Aalto-yliopisto
Aalto-universitetet
Aalto University

Master's Programme in ICT Innovation

Towards general end-to-end sensor fusion for robot localization: implementation of visual-inertial-wheel odometry

Giacomo Dario

Master's Thesis
2022

Aalto-yliopisto
Aalto-universitetet
Aalto University

Author Giacomo Dario

Title of thesis Towards general end-to-end sensor fusion for robot localization: implementation of visual-inertial-wheel odometry

Programme ICT Innovation

Major Autonomous Systems

Thesis supervisor Prof. Pietro Michiardi

Thesis supervisor Prof. Quan Zhou

Thesis advisor(s) Prof. Tammi Kari, PhD. Jari Vepsäläinen

Date 09.09.2022 **Number of pages** 60 + 4 **Language** English

Abstract

This thesis aims at generalizing a state-of-the-art end-to-end approach for robot localization in GNSS (GPS) deprived environments using a monocular camera, inertial sensors, and wheels encoder. The pipeline is trained and tested for autonomous vehicles, but the work aims to develop multimodal robot localization and observe how the method can be generalized. This thesis starts with an overview of the localization methods, structured to highlight the challenge of localization and sensor fusion, followed by a description of the state-of-the-art learning-based methods. Then, the data analysis and preprocessing are explained in the methods, as well as the structure of the pipeline, with a detailed analysis of its building blocks. Later, the results are shown and discussed, providing a comparison with the existing methods. To conclude the thesis, some observations about the presented methods and their future developments will be presented. This work has a solid industrial relevance since robustness in localization is an open problem and requires tailored engineering efforts, while having a strong research interest since it develops and expands the state of the art at the intersection between robotics and artificial intelligence.

Keywords Deep learning, Sensor fusion, Localization, Visual Odometry, Time-series regression, IMU, Wheel Odometry, Features Extraction

Contents

1 Introduction

In this thesis, a novel approach to sensor fusion for localization of autonomous vehicles will be described, a method that employs inexpensive sensors, such as cameras and inertial sensors, in GNSS-deprived environments and leverages the deep-learning methods aiming at achieving a robust end-to-end localization.

Odometry, or dead-reckoning, is a task that the human race has been performing throughout its existence, consciously or unconsciously, using more or less technologically advanced methods, for localization. According to the studies proposed in [1], the etymology of the term dead-reckoning is unclear. However, an interesting interpretation clarifies this term pictorially: the term *reckoning* recalls the process of calculating or estimating something, while the term *dead* refers to something that is still; so this term was first used for ships' navigation to calculate the ship position with respect to dead, unmoving, objects on the water. This mechanism, based on observing the relative motion of fixed reference with respect to us, lays the foundation for the problem of localization during navigation tasks. The word odometry is a synonym, it has greek roots, namely, it is composed of the words *odos*, which means way, or route, and *metron*, which means measure.

The odometry problem is the process of estimating an agent's position and orientation based on its movement. It is a fundamental task for many applications such as mobile robot navigation or object tracking systems; however, it remains challenging due to the inherent uncertainties in sensors used to measure relative motion. In particular, noisy measurements of linear and angular velocities can lead to errors that accumulate over time. This noise can come from different sources, such as mechanical limitations of sensors (e.g., IMU, encoders), changes in terrain conditions (e.g., slippery floors), or unpredictable external factors (e.g., wind). This work's main contribution lies in designing an algorithm to build a novel representation of odometry measurements that is robust against measurement noise. It autonomously fuses the sensors' raw data, selecting specific components of each sensor's raw data at each timestamp.

The studied approach is an evolution and union of several different methods proposed by the Department of Computer Science of Oxford University. It is based on a deep-learning-based feature extraction using CNN-based pipelines for the camera, IMU, and encoder data. The features are then fused together using an attentive mechanism, namely a selective mask that lets through only the features that do not correspond to noise and can positively affect the model's accuracy. After the selective mask, the features are

fed into an RNN (Recurrent Neural Network) that extracts the temporal relations between the past features. The three translations and the three rotations between two timestamps are estimated using different fully connected layers. The network parameters are trained using an optimizer called stochastic gradient descent. This method, although immature, since it has not been studied intensely in detail, and even if it is highly dependent on the quality and the quantity of data used to train the model, makes localization more robust, in particular to outliers, and enables the application of more inexpensive sensors.

This thesis starts with an introductory overview of localization methods and sensor fusion to let the reader understand the context of end-to-end selective sensor fusion for visual-inertial odometry. It is followed by the research methods, where the pipeline is explained with a particular focus on the data pre-processing and the used datasets. Then the results are shown and finally discussed to assess this method, elaborating on strengths and weaknesses, also from the point of view of industrial readiness.

2 Context and Literature Review

2.1 Localization

Localization is an extremely broad field. In this literature review, the reader will be guided through the different approaches, progressively narrowing down the scope of the problem to the end-to-end visual-inertial odometry, following a logical path that justifies why each branch of research has been studied.

There are two types of localization strategies: relative and absolute localization. Relative localization involves determining a robot's position relative to another object or agent, or the robot itself at different time steps, with respect to a coordinate frame within its working environment. On the other hand, absolute localization involves determining a robot's position with respect to an arbitrary coordinate system defined for its working environment. Most often, mobile robots use both relative and absolute localization together during their operation period. For example, while navigating through indoor environments, mobile robots may use relative localization to follow a human guide or track an object. As the definition suggests, the distinction between global and relative localization depends on what is considered the environment where the robot operates, and it depends on the application. For the scope of this thesis, we will consider the world as the environment, thus the position will be considered absolute with respect to the earth reference frame.

Another fundamental distinction to understand the problem of localization, which relates to the one just mentioned, is the distinction between *proprioceptive* and *exteroceptive* sensors. Namely, proprioceptive sensors measure the states of the robot with respect to the robot itself. On the other hand, exteroceptive sensors measure the states of the robot with respect to the environment. In general, in localization, exteroceptive sensors are more reliable and can be used to estimate the global position of the robot, while proprioceptive sensors suffer from a fundamental flaw. Since they can measure the state of the robot only with respect to the robot itself, they cannot estimate the robot's position, because the robot does not move with respect to its reference frame. On the other hand, sensors can be used to estimate other states and from those states, calculate the robot's relative position. Estimating a state based on the observation of a sensor leads to an additional source of errors added to those occurring on the observation. In some cases, the estimation process even increases the error. This is typically the case of INS or wheel encoders, which will be clarified further in

sections 2.1.4 and 2.1.5. These concepts will help the reader determine the advantages and disadvantages of each presented method.

2.1.1 GNSS

Ideally, the problem of localization is solved by using an accurate absolute source of localization, such as GNSS. In particular, high accuracy can be achieved using RTK technology, which not only uses signals transmitted from satellites but also from a calibrated ground station. However, RTK GNSS and GNSS localization becomes detrimental in urban areas, where the signals bounce between buildings, disrupting the triangulation computation performed by GNSS receivers. Moreover, the building walls completely block the GNSS signal in indoor scenarios. As a result, autonomous mobile robots and autonomous vehicles need to integrate sensors to compensate for the loss of accuracy (or loss of signal) of the GNSS in GNSS-deprived environments. From now on, the review will consider the research results that address the problem of localization in GNSS-deprived environments. Since localization is a vast field, some technologies might be missing in this review, but this presentation's goal is to follow a logical flow.

2.1.2 Beacon localization

A widely adopted solution for indoor environments is using beacon localization, where the absolute position of the robot is estimated with high accuracy using a similar strategy as the GNSS. For example, the system proposed in [2] uses a set of fixed beacons that emit known signals, in this specific case, Bluetooth signals. These signals are received by a mobile robot, which then uses triangulation to determine its position relative to the beacons. Using beacons for indoor localization is a well-established solution in the industry, but this method is not scalable to outdoor scenarios since it requires enormous infrastructure costs. These examples show how the idea of triangulating the position given the known position of some reference objects is typical of this approach and recalls the explanation of the term *dead-reckoning* in the introduction (section 1).

2.1.3 Vision

Taking a step further in the same direction, we can consider the localization based on vision as similarly performed. Vision-based localization is similar to the beacon-based one, but it does not require infrastructures since it uses visual landmarks [3]. The basic idea is that the robot can identify some

objects, or features, in its environment (e.g., doors, windows, corners, trees, road signs) and use these features as reference points for localization. The concept of feature is extremely important and it will play a key role in the end-to-end approach. In section 2.2.2 features will not only correspond to some characteristic, recognizable, points of the image, but also more complex features, such as patterns or bigger structures of pixels, or, in general, latent features, meaning that they cannot assume a direct meaning to humans.

The problem of localizing the objects and tracking their motion is well-studied in computer vision and it is known as optical flow. Optical flow is a fundamental concept that is utilized in one form or another in most video-processing algorithms aiming at estimating the dynamics of the objects in the scene. The dynamics of the object are represented through the optical flow field (or optical flow map), which corresponds to the apparent motion of objects in a given scene. In other words, it is a dense image-based motion estimation method that tracks optical flow vectors throughout an image, providing a dense set of velocity vectors. Although there are many different ways to formulate optical flow estimation methods, the simplest form of optical flow estimation is based on the brightness constancy assumption, which states that the brightness of a pixel in an image sequence remains constant over time. however, the brightness constancy assumption is not always valid, and therefore many optical flow estimation methods make use of more sophisticated assumptions, such as the brightness constancy assumption with a smoothness constraint, in order to more accurately estimate optical flow [4].

2.1.4 Inertial Navigation

Inertial navigation systems (INSs) are other sources of localization. INSs use a combination of accelerometers and gyroscopes to estimate the position, velocity, and orientation of a moving platform. The basic idea is that by measuring the acceleration and rotational velocity of the platform, one can determine its position and orientation relative to its starting point. For the sake of clarity further on in this thesis, INS contains IMU, which is the piece of equipment that observes the linear acceleration (with an accelerometer) and the angular velocity (with a gyroscope) of the sensor with respect to itself. From the output of the IMU, the INS integrates the accelerations and velocities and estimates the position of the robot. As a result, to determine the position of the robot, it is important to know where the sensor is located if the sensor is not placed in the center of the robot. According to the distinction made in 2.1, INSs are relative sources of odometry and they

are proprioceptive sensors. IMUs are widely used, especially thanks to the MEMS technology, which made the sensors extremely inexpensive, but likewise, more prone to errors. They are subject to drift over time, which limits their accuracy. Moreover, they do not work well in environments with large amounts of vibration or where the platform is not moving smoothly (e.g., over rough terrain). For INS, it is extremely important to limit the error to the neighborhood of 0, because a small consant error in the acceleration can lead to a quadratic error when the acceleration is integrated to estimate the position. Some research has been done ([5], [6]) to analyze the error and find proper calibration methods to minimize it. However, since the calibration cannot completely null the error in most cases, eventually the error will grow t,oo big to consider the localization as reliable and the robot will not be able to use that information or it could be danger,ous. This problem is shown in Figure 1. Although this sensor will lead eventually to an eventually lead, it works really well in detection at a high rate, for a few timesteps.So this feature makes the sensor extremely interesting to be integrated with GNSS, which is about 2 orders of magnitude slower than the IMU, but it bounds the localization error to a finite value. This is typically how the localization is performed in consumer smartphones. The integration of IMU and GNSS is studied in the broader field of sensor fusion, which will be explained more in detail with rein moreo the localization problem in 2.1.7.

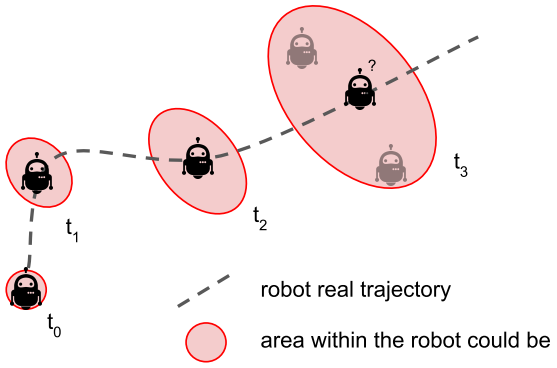

Figure 1: Covariance explosion using proprioceptive sensors: eventually (t_4) the robot could be anywhere inside the red ellipsoid.

2.1.5 Wheel Encoder

Wheeled robots can rely on another proprioceptive source of odometry. Wheel encoders count the precise number of rotations of the robot wheels, thereby estimating how far the robot has traveled. The kinematic model of the robot allows relating the rotation of the wheels with the motion of the robot. Typical models are the differential drive model or the Ackerman model in the case of four wheels autonomous robots, like autonomous cars. The Ackerman model is given by:

$$v_x = v\cos\theta - \frac{l}{2r}\tan\alpha\sin\theta$$

$$v_y = v\sin\theta + \frac{l}{2r}\tan\alpha\cos\theta$$

where l is the length of the robot, r is the radius of the wheels, α is the angle of the front wheels and v is the speed of the robot. With the kinematic model, the robot can be controlled by regulating the speed of the robot v and the angle of the front wheels α and the position of the robot can be determined based on the angles of the front wheels and on the velocity of the wheels.

Theoretically, assuming the model replicates perfectly the real robot kinematics, wheel odometry would solve the problem of localization but in real-world scenarios, wheels slip and manufacturing tolerances, variance in pneumatic pressure, and imperfect encoders, lead the robot's localization to be unreliable, as explained in Figure 1. Wheel odometry gives accurate observation for linear velocity but rather unreliable angular velocity, as opposed to the IMU, which senses a more accurate angular position since the gyroscope measurements (angular velocities) are integrated only once to estimate the orientation, while the linear accelerations are integrated twice for the linear position. The complementarity between these two sensors makes them good candidates for being fused with the GNSS, as explained in section 2.1.7.

2.1.6 Simultaneous Localization and Mapping

A more general approach to localization is to use a map of the environment and compare it to sensor data. This approach, known as simultaneous localization and mapping (SLAM), has been used extensively in mobile robotics. The basic idea is that the robot builds a map of its environment as it moves through. This map can then be used for localization, path planning, and

navigation. SLAM algorithms have been developed for various sensors, including LIDAR and vision.

The main advantage of SLAM is that it does not require any special hardware. Any sensor that can be used to detect environment features can be used for SLAM. In addition, SLAM algorithms are generally very robust and can handle significant amounts of noise and uncertainty. The main disadvantage of SLAM is that it requires a significant amount of processing power. This is because the map must be updated constantly as the robot moves through its environment. In the scope of outdoors autonomous vehicles, such as autonomous cars, the advantages of a global map fade out since most often the trip does not start and end in the same location, so the vehicle does not come back on a known part of the map, which, through the so-called loop closure, would limit the localization error. However, a local map can be built locally to triangulate the position of the features in the environment. SLAM can be implemented using features from 3D sensors (such as lidar or stereocamera) or as visual SLAM, using visual features whose depth can be estimated deterministically or using neural networks. The problem of dead reckoning is tightly related to SLAM, however, the presented pipeline will not implement SLAM, but the deep learning pipeline could be improved to perform SLAM, this would likely increase the system accuracy.

2.1.7 Sensor Fusion for localization

Sensor fusion algorithms are a set of mathematical operations that are performed on data received from multiple sensors to provide a more accurate representation of the real world. Sensor fusion algorithms are used to estimate the state of a system by combining the data from multiple sensors. The sensors can be of different types, such as a camera, an infrared sensor, a sonar sensor, and others. The data from each sensor is processed independently to estimate the state of the system. The data from multiple sensors is then combined to produce a more accurate estimation of the state of the system. The data from each sensor is processed independently to estimate the state of the system. The data from multiple sensors is then combined to produce a more accurate estimation of the state of the system.

There are various types of sensor fusion algorithms, such as the Kalman filter, the particle filter, and the extended Kalman filter. The Kalman filter is a linear sensor fusion algorithm that is used to estimate the state of a system. The others are generalizations of the Kalman filter: the extended Kalman filter is a nonlinear sensor fusion algorithm that is used to estimate

the state of a system, it works as a Kalman filter but linearizes the kinematic model; the particle filter is a nonlinear sensor fusion algorithm that is used to estimate the state of a system lifting the restriction of the gaussian distribution of the sensors errors. Built on top of this method, numerous other filtering strategies can be found in the literature. As explained in [7] Kalman filtering, and the approaches derived from it, have some drawbacks which make the problem of localization not completely solved. In particular, Kalman filters lack robustness when the model of the system is inaccurate, which very likely is since most of the time, it is difficult to model complex systems. Moreover, Kalman filters are sensitive to outliers: if a sensor fails, providing a random value, the filter will weigh less than that value, but it will still be taken into account in the average that estimates the state, in the application of this thesis, the position of the vehicle. This problem is more visible with GNSS data, where outliers can be caused by the signal bouncing on buildings, affecting the filtered measurement. Furthermore, Kalman filter performance is affected by the initial parameters (covariance matrices of model and sensors and gains), which makes them less robust. These drawbacks can be minimized by fine tuning the initial parameters for the covariances and tuning the dynamic model, however, this process is extremely time-consuming and does not solve completely the problem of robustness against outliers. In the past years, research has developed methods to make Kalman filtering more robust to outliers, for example, [8], but the algorithm still requires appropriate tuning for achieving the desired performance. In this thesis, we are looking for a method that increases robustness and is end-to-end, meaning that no tuning should be performed, but all the parameters are learned. From this moment on, we will consider the solutions applying deep learning, to finally consider the end-to-end sensor fusion for Visual Inertial Odometry.

2.2 Learning methods

Deep learning has been studied for decades and it has gradually become more popular over the last few years as more and more businesses and organizations have started to realize its potential. However, deep learning became truly popular after the 2012 ImageNet Challenge, where a deep learning algorithm outperformed all other competitors in the task of image classification, this algorithm is known as AlexNet [9]. This event showed the world the power of deep learning and sparked a wave of interest and investment in the field. Deep learning is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data.

These algorithms are used to automatically extract features from raw data by learning patterns in order to obtain the desired output.

As opposed to deterministic approaches, it is able to learn complex patterns in data that would be difficult for a human to identify. Additionally, deep learning is able to learn from data that is unstructured or unlabeled, which is often the case with real-world data.

The problem of estimating the robot position and orientation can be seen as a regression problem, where, given the sensor data, the algorithm regresses 3 values of position and 3 values of orientation.

2.2.1 Hybrid methods

A few hybrid approaches integrating deep learning and deterministic sensor fusion have been implemented in the past years, leveraging the power of machine learning to model complex relations, while preserving a degree of interpretability of the algorithm.

One example is KalmanNet [10], which is a neural network that uses the Kalman filter to predict the future values of a time series. The KalmanNet is trained using a time series of data, such as stock prices or weather data. A similar method, specifically for GNSS and IMU is implemented in [11]. In both these papers, the Kalman filter is used to predict the future values of the time series, namely estimating the position of the robot. The predicted values are then compared to the actual value and the error between the predicted and actual values is used to update the weights of the neural network. These methods can be used to estimate the robot position, modeling more accurately the dynamic model, which, as explained in section 2.1.7, is one of the main flaws of Kalman filtering. However, the presented hybrid methods estimate the dynamic model, but require prior information about the sensors' covariance and consequent tuning, making the pipeline not end-to-end. However, hybrid solutions show outstanding results, both in terms of accuracy and computation time since the Kalman Filter is a low-computation optimal estimator.

2.2.2 DeepVO

Odometry is a common problem in the field of computer vision. It aims to recover the ego-motion of a camera, therefore a robot, by using only the visual input. It has been widely used in many fields such as navigation, augmented reality, robotics, etc. The VO algorithm computes the increment camera motion between two consecutive frames and accumulates the

estimated motion to obtain the absolute camera trajectory. Since the visual input is noisy, the incremental motion estimation is inherently ill-posed. The VO problem is even more challenging for monocular visual odometry (MVO), in which case the absolute scale is not available.

Existing VO algorithms are, in general, developed under a standard pipeline including feature extraction, feature matching, pose estimation, local optimization, etc. Although some of them have demonstrated superior performance, they usually need to be carefully designed and specifically fine-tuned to work well in different environments. Some prior knowledge, such as the intrinsic parameters of the camera, is also required by most of the traditional VO algorithms. As mentioned, the absolute scale of the camera trajectory is another problem for monocular visual odometry.

There are many methods that can be used to recover absolute scale in monocular visual odometry. Some common methods include using GPS data, using a known distance between two points in the scene, or using the height of the camera above the ground plane. Deep learning addresses this problem since the model can learn to estimate the scale of the features, as long as the environment is similar between training and testing. Therefore, as in most machine learning applications, the generalization performances of the model are tightly related to the variety and quality of the dataset.

The paper [12] presents a new end-to-end monocular VO algorithm, called DeepVO. This new paradigm uses Deep RCNNs and is able to achieve simultaneous representation learning and sequential modeling of the monocular VO by combining the CNNs with the RNNs. The main advantage of this paper is that there is no need to carefully tune the parameters of the VO system because it is trained in an end-to-end manner. However, the authors conclude, based on the results, that this method cannot be yet considered as an alternative to geometric approaches, but rather complementary to those methods, since geometric approaches can improve the accuracy. The pipeline of DeepVO is described in figure 2.

2.2.3 Selective Sensor Fusion for VIO

After DeepVO, the same research group at Oxford University focused on making the pipeline more accurate and robust. In order to achieve this result, they implemented a novel approach for sensor fusion, to fuse vision with IMU.

The first paper they published presents VINet [13] an end-to-end trainable system for performing monocular visual-inertial aided navigation. VINet pipeline concatenates the features extracted from the vision with the tem-

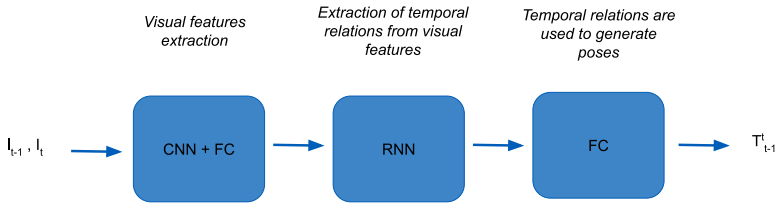

Figure 2: Overview of the end-to-end DeepVO system. It consists of an image encoder to extract features, a recurrent convolutional neural network (RCNN) to model the sequential dynamics and relations, a pose regressor to estimate the camera pose and a pose accumulator to recover the absolute camera trajectory. The trained network takes a sequence of two raw RGB images as input, and outputs the absolute transformation between timestep t and $t+1$ in the form of Euler angles and translation vectors.

poral features extracted from the IMU data. Then the concatenated features are plugged into the RCNN network that extracts temporal relations between features, and then the temporal relations are passed to a fully connected pose regressor, to output the transforms between timesteps. VINet has shown comparable results with traditional approaches, which require much hand-tuning during setup. The key advantage of VINet is that it can learn to become robust to calibration errors. The authors claim that VINet is the first step toward truly robust visual-inertial sensor fusion.

Shortly after, they published a new paper with the same pipeline but implementing a more sophisticated method to fuse the visual and inertial features together, this method is called Selective Sensor Fusion for Visual-Inertial odometry (VIO) [14], which form now will be referenced as Selective VIO. The pipeline is shown in Figure 3, it shows how the pipeline has evolved from the DeepVO pipeline.

If VINet was merely concatenating the features together, Selective VIO implements two different trainable masks: the hard mask that blocks or lets pass some features or the soft mask which associates a gain to each feature to amplify or reduce the effect that each feature will have on the final pose estimation. The mask can be seen as an attentive mechanism that allow the model to focus only on the latent features that positively affect the accuracy of the estimation. A more detailed explanation can be seen in section 3.2.4.

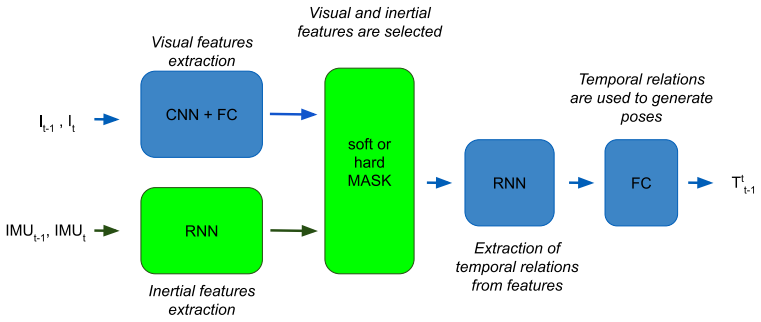

Figure 3: Pipeline of Selective Sensor Fusion for VIO. The figure shows that the visual features are now concatenated with the IMU features, and they go through a trainable soft or hard mask.

The authors claim that the results of their new pipeline show that, when

data is suddenly corrupted, the performance of learning-based fusion strategies degrades more gently, while filtering methods fail abruptly with the presence of considerable sensor noise and misalignment issues, as aforementioned in section 2.1.7.

2.2.4 SelfVIO

At this point, it is important to clarify some concepts about machine learning that can help the reader understand how the different model change and why. In particular, we must distinguish between supervised, unsupervised, and self-supervised learning.

- Supervised learning is when the model is given a set of training data, and the desired outputs for that data, and the model is then trained to produce the desired outputs for new data.
- Unsupervised learning is where the model is given a set of data but not the desired outputs, and must find patterns in the data itself.
- Self-supervised learning is a subset of unsupervised learning in which the training data is labeled by the algorithm itself, rather than by external sources. This usually means that the data is transformed in some way so that the labels can be generated from the transformed data. For example, a common self-supervised task is the prediction of the next frame in a video. The input data would be a video clip, and the labels would be the subsequent frames in the clip. The algorithm would learn to predict the next frame in the clip without any external labels. The main difference between self-supervised and unsupervised learning is that unsupervised learning does not use labels, while self-supervised learning uses labels that are generated by the algorithm itself. There are a few advantages of self-supervised learning against supervised learning. One is that self-supervised learning can be used when labeled data is unavailable, which is often the case. Another advantage is that self-supervised learning can be more efficient than supervised learning since it does not require data labeling. Finally, self-supervised learning can be more robust to overfitting since it does not rely on a fixed set of labels. However, these advantages come at the cost of finding a proper solution to extrapolate labels from the data, which is a task-dependent problem. To clarify these concepts in the scope of this thesis, supervised learning requires the ground truth poses, meaning that training data also provide the correct pose that

the model will estimate so that the model will tune the parameters to minimize the difference between the estimated and the real pose.

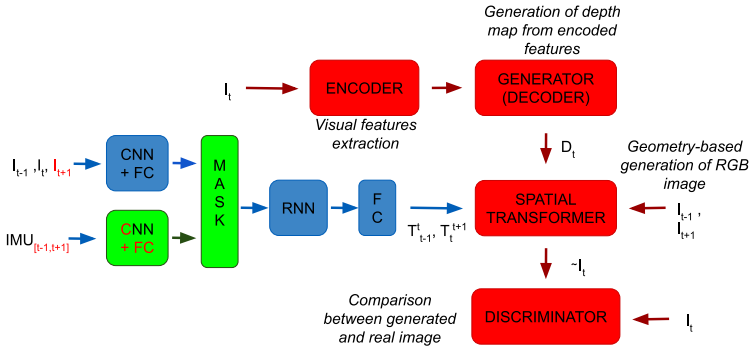

Figure 4: Pipeline of SelfVIO.

Both DeepVO and Selective VIO are supervised learning approaches since they use ground truth poses for training. Self-supervised approaches instead will output the poses without having the ground truth poses, instead using smartly the given input. This will be clearer soon when SelfVIO [15] is described.

As mentioned, a new paper was presented by the Oxford research group, which implements a self-supervised method, this pipeline is indeed called SelfVIO [15]. This pipeline adopts a novel method to generate labels and uses generative networks known as GANs.

GANs became famous in 2016 after a paper [16] was published by Ian Goodfellow and his colleagues. The paper showed how GANs could be used to generate realistic images. The applications of GANS are many and varied. They have been used to generate images, videos, and even 3D models. GANS are generative adversarial networks, a type of neural network used for unsupervised learning, namely self-supervised learning. They consist of two parts: a generator network and a discriminator network. The generator network creates samples, while the discriminator network tries to distinguish between actual and generated samples. The two networks are trained together, with the generator network trying to fool the discriminator network and the discriminator network trying to classify samples correctly.

Considering how this powerful tool is used for the SelfVIO pipeline is

extremely interesting. As the figure 4 shows, the generator network is a CNN that is trained to build the depth map from the extracted visual features from the frame at timestamp t while the discriminator network is trained to determine which, between the real image at timestep t and the generated at timestep t, is the generated. A deterministic spatial transformer is used to generate the image at timestep t. It constitutes a set of geometric rules that, based on the estimated map (represented by a point-cloud) and the camera frames at timestamp $t-1$ and $t+1$, guesses what could be the most similar camera frame to the real camera frame at timestamp t. Thus, the spatial transformer takes as input the depth map at timestep t coming from the generator, the estimated transformation (poses) between t and $t-1$ and t and $t+1$, coming from a pipeline with a structure similar to Selective VIO, and the input frames at $t-1$ and $t+1$, and outputs the generated image. The idea of this method is that the more accurate the transforms given input to the generator, the more accurate will be the generator outcome, generating an image that will be more similar to the real one, making the discriminator's job harder. On the other hand, the discriminator will learn to improve the ability to distinguish the real and generated image, making the generator's job harder. This adversarial learning strategy will improve the prediction quality without needing ground-truth poses.

3 Research Design and Methods

3.1 Dataset

3.1.1 Input data and labels

To better understand the decision and the operations that were performed in relation to the dataset, it is important to clarify what are the inputs and outputs of our system. The inputs are:

- Consecutive camera frames: the model takes two consecutive 512×256 images. In the initial phases of the development, the frames were RGB (KITTI dataset) but due to reasons that are explained in section 3.1.2, the used images are grayscale (ComplexUrban dataset). However, to keep the same model, the grayscale input data has 3 channels, copying the grayscale channel in the R, G, and B channels. The two images are concatenated along the horizontal axes of the camera frame, as shown in figure 5 constituting an input tensor of $3 \times 1024 \times 256$, which is redundant in the first dimension. The framerate of the camera is 10 Hz.

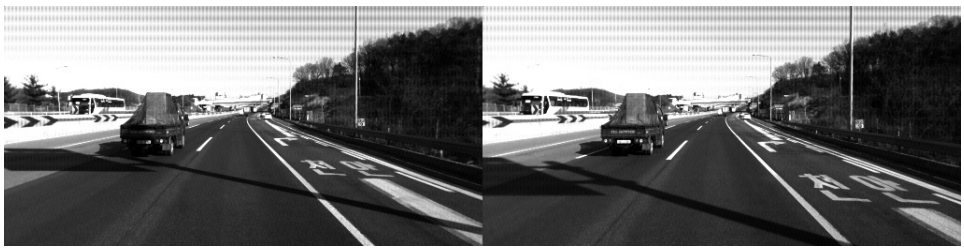

Figure 5: Visual input: Consecutive images concatenation (timestamp t-1 and t).

- IMU data: the IMU data is divided into 3 linear accelerations (acc_x, acc_y, acc_z) and 3 angular velocities $(\omega_x, \omega_y, \omega_z)$. The IMU is sampled at 100 Hz, which is 10 times higher frequency than the camera frame, thus, for every pair of consecutive camera frames, there are 10 IMU measurements in between. As a result, the IMU data is fed into the network as a matrix of size 10x6, populated with the 3 accelerations and the 3 angular velocities for the 10 timestamps. An example of an IMU sample is:

ts	acc_x	acc_y	acc_z	ω_x	ω_y	ω_z
0	0.12	0.003	9.83	0.03	0.024	0.96
1	0.11	0.003	9.83	0.06	0.024	0.91
2	0.17	0.006	9.86	0.06	0.024	0.84
3	0.14	0.009	9.85	0.07	0.024	0.87
4	0.16	0.003	9.85	0.09	0.024	0.81
5	0.17	0.003	9.84	0.03	0.024	0.81
6	0.11	0.008	9.83	0.05	0.024	0.86
7	0.11	0.003	9.82	0.05	0.024	0.92
8	0.09	0.012	9.79	0.07	0.024	0.91
9	0.17	0.008	9.77	0.08	0.024	0.95

- Wheel encoder data: raw wheel encoder data are fed into the system, in particular, the system takes in the difference between the ticks that occurred between timestamp $t-1$ and t. As explained in section 2.1.5, the wheels encoder increases the number of ticks proportionally with the angle by which the wheels spin. More simply, $n_{ticks} \propto \theta_{wheel}$. Feeding in the system the difference of ticks between consecutive timestamps is equivalent to feeding in the system the angular velocity by which wheels spin. Wheel encoder data are recorded on the rear wheels of the vehicle, so it is a 2D input. Moreover, wheel encoders, as well as IMU, are sampled at 100 Hz, so there are 10 data points for every pair of consecutive images. As a result, the wheel encoder data is fed into the network as a 10×2 matrix populated with the left and right wheel $\delta ticks$ for 10 timestamps.

ts	w_L	w_R
0	512	508
1	512	508
2	512	508
3	512	508
4	512	508
5	512	508
6	512	508
7	512	508
8	512	508
9	512	508

The output, or rather, the label, is the homogeneous 3D relative transformation between timestamp $t-1$ and t. In the preprocessing, we will explain

further in detail the operation of extraction of the relative transformations from the absolute transformations.

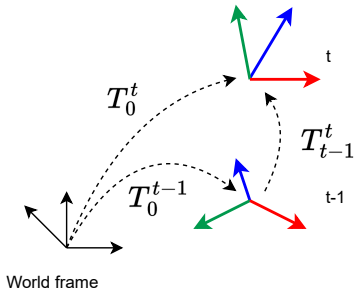

Figure 6: Relative pose from absolute poses.

The relative poses can be calculated as $T_{t-1}^t = T_0^{t-1} \cdot T_0^t = (T_0^{t-1})^{-1} \cdot T_0^t$, using the notation depicted in figure 6. However, the relative transforms are transformed into two vectors of 3 translations and 3 orientations that are the real labels in the network. The translation vectors $[t_x, t_y, t_z]$ correspond to the distance in meters covered by the vehicle in one timestamp, that is 10 ms, while the orientation vector represents the roll, pitch, and yaw angles, the angles obtained by consecutive rotations about X, Y, Z, typically used for aircraft and ships orientation (R, P, Y). The model is meant to estimate the two vectors, which define the position of the robot at timestamp t with respect to the robot position at timestamp $t-1$. However, given the type of motion performed by the car, only the longitudinal translation (x) and the yaw angle (θ_z) will actually be non-zero. For this reason, to guarantee a faster convergence of the model and to simplify the visualization of the results, we decided to simplify the problem to a 2D problem, so the two labels to be predicted are x and θ_z.

3.1.2 ComplexUrban vs KITTI

The research that has been carried out until this point and in particular the paper that in this thesis has been thoroughly investigated uses the KITTI dataset [17], a dataset that contains sensors such as monocular cameras, stereo cameras, IMU, GPS, LIDAR, etc, recording a car driving in the area of Karlsruhe, Germany, providing ground truth of the car position and ground truth for classification tasks. This dataset is excellent for purely vision/lidar-based tasks, however, it is not the best option when other sensors need to

be used, due to the lack of time synchronization between ground-truth and sensors measurements and the smaller set of different sensors.

In particular, the main reason to change the dataset is that the KITTI dataset does not provide timestamps associated with the sensor's measurements, and the authors of the dataset claim that ground truth poses and camera frames are synchronized, however, for every sequence, the cardinality of the poses and the frames are different (sometimes by even hundreds of samples), which makes it impossible to pair the frame with the exact matching ground truth.

When the first tests using the KITTI dataset were carried out during the research, data was preprocessed so that the cardinality of frames and poses corresponded, simply trimming one end of the list of camera frames to match the size of the list of ground-truth poses, but most probably a temporal offset between GT and camera frame was in place, which could not be corrected, due to the fact that there no timestamps are provided in the dataset, but camera frames, GT and sensor measurements are ordered.

Another reason to change the dataset is related to the IMU. The IMU data is supposed to be synchronized with the frames, but with no timestamps, it is impossible to pair them correctly. Moreover, a frequency of 10 Hz is in general too low, since IMU is affected by high-frequency noise, which at 10 Hz is very difficult to be filtered out since 10 Hz is closer to the main frequency of the system rather than the noise. As the Nyquist-Shannon theorem explains, the sampling frequency needed to reconstruct a signal has to be twice the frequency of the signal. Thus, a higher frequency is required in order to reconstruct the signal and successfully separate the noise from the main signal, either if the filtering is performed deterministically or is aimed to be performed through the regression, as we aim to achieve in this thesis.

Last, KITTI does not provide wheel encoder data, which we are planning to fuse in the pipeline in the same manner as the IMU is fused.

These observations, in addition to raising some doubts about the methods and results of the paper we have investigated, lead to the decision of using a different dataset. After analyzing the options, the best choice resulted in the ComplexUrban Dataset [18], a dataset for vehicles or robots operating in urban environments. The dataset includes data from multiple LiDARs and stereo cameras. The ComplexUrban dataset provides a whole set of consumer sensors that can be used in the future to extend the scope of the end-to-end sensor fusion for odometry, pursuing the task of localization using easily available sensors. Among the consumer sensors featured in ComplexUrban Dataset, IMU and wheel encoder will be fused to the visual odometry to achieve higher accuracy and robustness.

The dataset provides the vehicle's ground-truth position, estimated using simultaneous localization mapping (SLAM).

The most important feature of this dataset is the presence of timestamps, that can be used to perform accurate analysis of data and deterministically preprocess the data to make sure the model learns from the correct distribution, which can be later perturbated by adding noise to make the model generalize, yet having control over the data. The process of data preprocessing is thoroughly explained in section 3.1.3 The major drawback of this dataset is that the frames are only black and white, providing less information than the RGB images. As seen in 3.2.1, some modifications will need to be performed to use grayscale images as input.

Furthermore, this dataset is more recent (2017); as a result, it is more compatible with current methods of research and development in robotics. In particular, data is gathered using ROS, a robotic middleware widely used in Accademia and is increasing its popularity in the industry. Using ROS bags, streams of data are collected with their timestamp and stored in an organized way that allows reutilizing the data. ROS bags allow replaying the stream of data live, allowing asynchronous testing of real-time algorithms. This feature allows generating a package for end-to-end localization that can be reutilized by the community, getting data from ROS topics (streams of data) and estimating the position. If the model proves to outperform the existing methods, the package would be a valuable alternative for the robotics community.

3.1.3 Data preprocessing

A preliminary analysis was performed to determine the actions to take to preprocess the data. Thus, information has been plotted, reporting the frequency at which the sensors and the ground truth are gathered. Moreover, as the images show, each sensor's initial and last timestamps have been analyzed. As images 7 and 8 show, there is often an offset in the initial timestamp that could be explained by sensors being started at different times. Moreover, it is noticeable that most frequently, all the sensors show a constant period with the average matching the nominal period described in the paper with a pronounced yet acceptable standard deviation, probably due to the data not being gathered using an RTOS (real-time operative system), as Ubuntu. However, as the plot of section 39 shows, compared to section 21, in some scenarios, it is evident how the ground truth presents long sleeping periods of the order of magnitude of tens of seconds. The initial hypothesis for this behavior was that since section 21 is a highway

section while 39 is an urban section, the ground truth was not updated in GNSS-deprived scenarios, which is the case of urban scenarios, in particular in a city like Seoul, where high skyscrapers make GNSS signal detrimental. After further analysis, it was clear that the ground truth is not updated when the car is still, which often happens in urban scenarios with many traffic lights or crossroads.

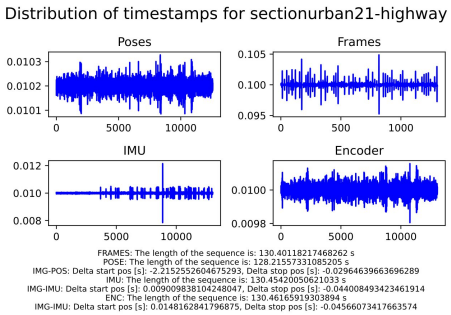

Figure 7: Analysis of section 21.

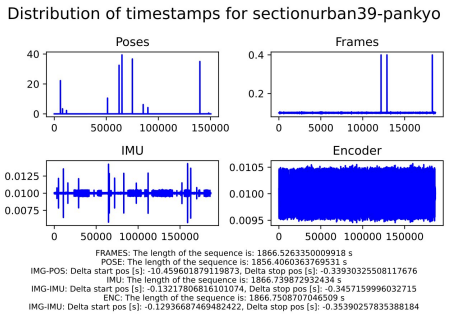

Figure 8: Analysis of section 39.

Moreover, as the figures 7 and 8 show, the ground truth is sampled at 100 Hz, while the camera frames are sampled at 10 Hz. Since the model is meant to estimate the transforms between consecutive images, aided by other sensors, it is crucial to match the frequency between camera frames and ground truth poses. To solve this problem, the nearest ground truth neighbor in the time domain was selected for every camera frame. As a result, the preprocessed samples have the same amount of camera frames and ground truth and all the imu and wheel encoder samples between the first and the last timestamp of the sequence, namely $10 \cdot (N-1)$ with N being the number of camera frames (and GT poses). Each training sample has a length of 3 images, meaning that the model can learn temporal relationships up to 0.3 s (3 samples times 0.10 seconds) in the past since the model is supposed to estimate the relative position of the robot with respect to itself, as explained in section 2.1. On the other hand, the test sample has the size of the whole sequence, that is, the same amount of images and poses. The downsampling operation is described by figure 9. The final sample is shown in figure 10.

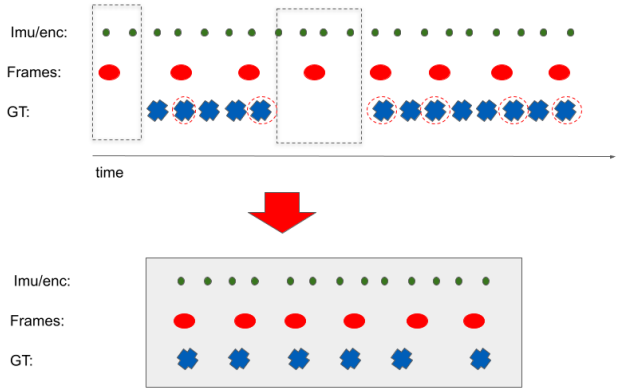

Figure 9: Downsampling of poses and generation of the samples.

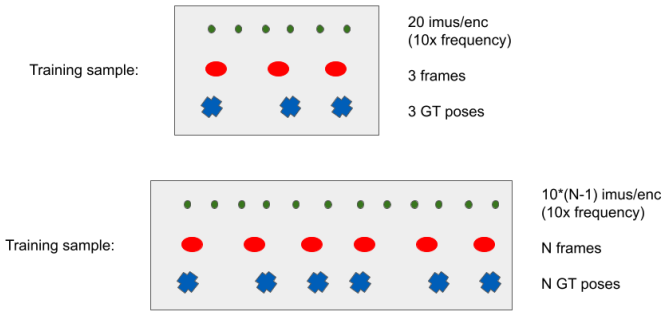

Figure 10: Training and test samples.

Moreover, since the dataset provides absolute poses in the MGRS (Military Grid Reference System), all the poses had to be transformed into sequential relative poses. This can be achieved using homogeneous transforms. Homogeneous transforms [19] are used to represent points in the 3D space, describing the position and the orientation. The advantage of using homogeneous transformations is that by simply chaining some matrix operations, we can describe multiple transformations of a point in space. In particular, the performed operation is described in section 3.1.1. Moreover, since the model uses images, normalization is applied. Image normalization

is used in machine learning to improve the convergence of an optimization algorithm by scaling the data so that it is in a range that is more amenable to the algorithm. The idea is to scale and shift the input parameters so that they all belong to the same range (usually 0 to 1) so that the parameters are in the range of the activation functions where the gradient is non-zero, which leads to faster learning. The camera frames are resized to 512×256 to keep compatibility with the pretrained Flownet.

3.1.4 Data augmentation

The ComplexUrban dataset shows less variability than the KITTI dataset, but with the shortcomings already described. As a result, data augmentation needed to be performed. Data augmentation improves the performance of a machine learning model by increasing the number of training examples and by increasing the diversity of the training dataset. Data augmentation can also improve the generalization performance of a machine learning model by making the model more robust to changes in the data distribution. For this task, different techniques of data augmentation have been applied. These methods will briefly be described and explain why they were applied.

- **2x SPEED**: the dataset has been augmented by adding samples with twice the speed. The goal was to increase the number of samples and obtain a wider range of speeds, particularly at lower speeds, since higher speeds are filtered out as explained in 3.1.6. This augmentation is performed by selecting every two images and every two ground-truth global poses, summing up two consecutive wheel encoder measurements, and summing up two consecutive imu samples.

- **MIRROR**: to increase the number of curving sequences in the dataset and to have the same amount of left and right turns, all the samples were mirrored. This was achieved by mirroring all the camera frames, then by mirroring the relative homogeneous transforms across the plane generated by the x and z axes (the vertical plane on the longitudinal axis) through the following transformation:

$$\begin{bmatrix} r_{xx} & r_{xy} & r_{xz} & t_x \\ r_{yx} & r_{yy} & r_{yz} & t_y \\ r_{zx} & r_{zy} & r_{zz} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{xx} & -r_{xy} & r_{xz} & t_x \\ r_{yx} & -r_{yy} & r_{yz} & t_y \\ r_{zx} & -r_{zy} & r_{zz} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  Furthermore, the left and right wheel encoder data were swapped ($w_L = w_R$, $w_R = w_L$), the sign of imu data for angular velocity about

the z-axis, and the size of the acceleration along the y-axis was reverted ($acc_y = -acc_y$, $\omega_z = -\omega_z$).

- **BACKWARD**: this augmentation is mainly meant to avoid overfitting and make the model learn the more general rule that relates sensors and output, but it can be helpful in case the robot actually moves backward. This augmentation was achieved by reverting the order of the images and the order of the absolute ground poses in every sample, by reversing the sign of the wheel encoder measurements, and reversing the sign of imu data for angular velocity about the y-axis (longitudinal) and the sign of acceleration along the x-axis ($acc_x = -acc_x$, $\omega_y = -\omega_y$).

- **STILL**: this augmentation provides more still samples since the raw training set has almost zero sequences where the car is still. It is easily visible how adding still sequences improved the ability of the model to predict more accurately when the car is still. This augmentation was achieved by populating the sample with the same image, the same absolute pose, and zeros in both the wheel encoder data and imu data (except for 9.81 on the acceleration along the z-axis to simulate the pure effect of gravitational acceleration). Figures 11 and 12 show the impact of adding still sequences to the dataset used for training the Wheel Encoder Network (3.2.3): the model reduced the bias term when the ground-truth translation and orientation are zero.

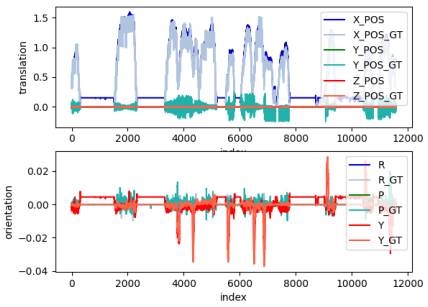

Figure 11: Wheel Encoder Net before STILL augmentation.

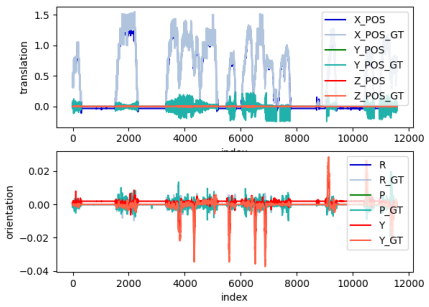

Figure 12: Wheel Encoder Net after STILL augmentation.

3.1.5 Data degradation

In order to achieve robustness, the model needs to be trained with perturbated input data that resemble possible real degradations. The probability of the sample being degraded can be set in the training and test phase so that the performances of the model in the different scenarios can be accessed.

The degradations of the visual data that can occur either singularly or combined are:

- **BLANK**: this degradation method resembles the absence of an image, for example, due to an unreliable connection, leading the image to be blanked out. In practical terms, this means zeroing out all the elements of the image matrix.

Figure 13: Image degradation: blank image.

- **BLUR**: the image is blurred through convolution with a kernel of size 15, moreover the images are deteriorated to resemble the salt and pepper effect. The salt and pepper noise of images is caused by the presence of white and black pixels in the image. These pixels are usually caused by the presence of dust or other contaminants in the camera sensor. Practically, with a probability of 0.005, a pixel can be black or white.

Figure 14: Image degradation: blurred image.

- **OCCLUSION**: this degradation method is meant to resemble the obstruction of an object in front of the lens or in from of the camera, such as a water drop or dirt stain on the camera lens. This method, as well as the blur, is also beneficial to avoid overfitting some specific features that might be typical of the training set (e.g., guard rails or landscape features).

Figure 15: Image degradation: occluded image.

Moreover, the IMU and wheel encoder data are degraded too to achieve higher robustness. In particular, two methods are used:

- **BLANK**: as well as for the visual data, the sensor's data can be blanked out in case the communication with the sensor ceases for several reasons, such as sensor failure, unreliable electrical connection, or the sensor driver failing. Practically, this is achieved by zeroing all the sensors' data out.

- **NOISE**: this degradation adds a gaussian noise of amplitude proportional to the sensor measurement to the sensors' measurements and it is supposed to resemble any disturbance that can occur on the sensors, such as vibrations in the case of IMU data or wheel slipping in case of wheel encoder data.

3.1.6 Data balancing

The dataset distribution is about 40% open highway, 30% urban highway, and 30% urban. This proportion, combined with the data augmentation which makes the proportion of highway sequences increase, would make the model overfit the highway sequences giving poor results on urban sequences. For this reason, the samples were selected to generate a more balanced dataset. In particular, the longitudinal distance between two poses, meaning the x component of the translation in the relative homogeneous

transformation, which corresponds to the linear speed, was used to discriminate the highway samples. A longitudinal distance of 2 meters at a 10 Hz sampling rate responds to about 70 km/h, while 4 meters correspond to about 140 km/h. The logistic function shown in figure 16 determines the probability of dropping the sample. This function can be tuned to let more or fewer highway samples through.

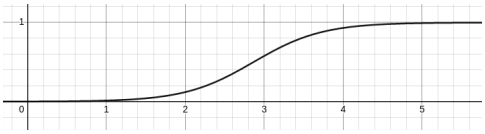

Figure 16: Probability of dropping a sample based on the longitudinal distance covered between two timestamps.

Furthermore, the dataset shows an imbalanced distribution of straight roads and curves, as a result, the training would maximize the accuracy on the straight roads penalizing the angular accuracy. To achieve a more balanced dataset, samples were discriminated based on the yaw angle extracted from the homogenous relative transforms. If the yaw angle was smaller than 1 degree (corresponding to an angular rate of 10 deg/s at 10 Hz), the sample was dropped with a probability of 0.7, meaning that 30% of the straight sequence was let through.

3.2 Model Building Blocks

3.2.1 DeepVO

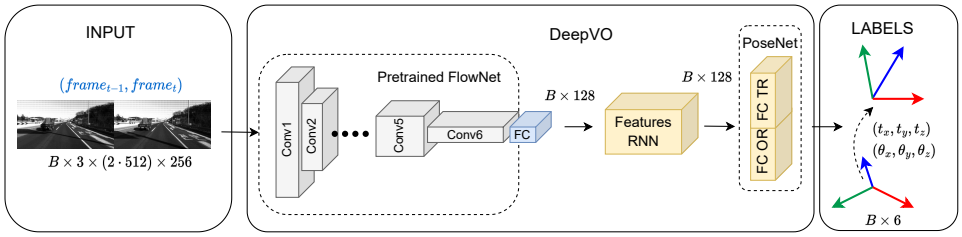

Figure 17: DeepVO pipeline.

As described in 2.2.2, the basic initial structure is the structure of DeepVO. In particular, it is made of a network that takes in the concatenated images and extracts the features through the encoder that is part of the FlowNet pipeline [20], a network to estimate optical flow, that is, for every area of the image (e.g. pixel) it is associate a vector that represents how that area has moved within two camera frames. In order to be adapted to the new dataset, which uses grayscale, Flownet was retrained with grayscale images. The retraining was performed by turning the Flying Chairs Dataset used in [20] to grayscale. The loss of information from RGB to grayscale did not affect since the EPE (Expected Prediction Error) between RGB and grayscale was comparable, however, higher accuracy can be achieved by retraining FlowNet with KITTI Dataset. The FlowNet encoder used in the DeepVO pipeline is made of 6 consecutive stages of a convolutional neural network. The structure of the neural network can be briefly described as follows:

- Conv1: input planes = 6, output planes= 64, kernel=7, stride=2;
- Conv2: input planes = 64, output planes= 128, kernel=5, stride=2;
- Conv3: input planes = 128, output planes= 256, kernel=5, stride=2
- Conv3_1: input planes = 256, output planes= 256, kernel=3, stride=1;
- Conv4: input planes = 256, output planes= 256, kernel=3, stride=2;
- Conv4_1: input planes = 512, output planes= 512, kernel=3, stride=1;
- Conv5: input planes = 512, output planes= 512, kernel=3, stride=2;

- Conv5_1: input planes = 5122, output planes= 512, kernel=3, stride=1;
- Conv6: input planes = 512, output planes= 512, kernel=3, stride=2;
- Conv6_1: input planes = 1024, output planes= 1024, kernel=3, stride=1;

Every convolutional layer is two-dimensional, uses batch-normalization, and is followed by a Leaky ReLU (Leaky Rectified Linear Unit) activation layer with a negative slope of 0.1.

The 1024 features extracted by the FlowNet CNN go through a fully connected layer to reduce the dimensionality of the features and adapt it, in case of fusion, to the selected number of features. The number of tested features is mainly 128 and 256. The linear layer is trained with different dropouts to make the model more robust to external modifications and avoid overfitting, which is typically a problem of camera-based solutions, as will be shown in section 4.

Afterward, the features flow in a recurrent neural network that is supposed to extract the temporal relationships between features extracted in different timestamps. The model was trained with different sequence lengths, mainly ranging between 3 and 5 camera frames, that is from 2 to 4 poses to be estimated, which correspond to the model keeping in memory the last 0.3 to 0.5 seconds. Further experiments can be performed with longer sequences. The RNN, which features dropout, is made of a bidirectional 2-layers LSTM (Long Short-Term Memory) with both input size and hidden size equal to the number of features (e.g., 128 or 256).

Finally, the temporal features go through two separate fully connected layers, one for orientation and one for translation, this is what the authors of DeepVO call PoseNet. Respectively, the linear layers, featuring dropout, output the Euler angles and the three translations in the space. As explained in section 3.1.1, at this point the model was trained to output only 2D poses for simplicity and faster convergence so the output of the network is now one orientation and one translation.

3.2.2 IMU Net

The IMU network resembles the same structure as DeepVO, but the way features are extracted from the sensor data makes it different. The feature extraction is performed through a convolutional network and a recurrent neural network. The CNN is meant to filter the IMU signal which is sampled at 10 Hz and is affected by high-frequency noise and low-frequency bias. This CNN takes the 10 timestamps for the 6 channels in and outputs 16

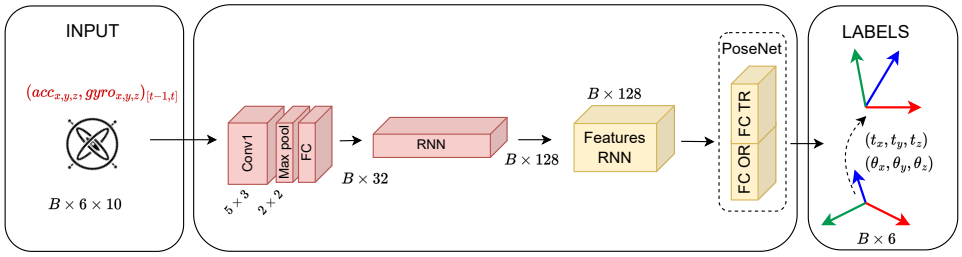

Figure 18: IMUNet Pipeline.

channels using a kernel size of 5 and stride of 3. This layer features batch normalization and is followed by a Leaky ReLU with a negative slope of 0.1. The output goes through a max pooling layer with a kernel size of 2 and a stride of 2 to amplify those features that mostly convey signal information. Afterward, the 32 features that are extracted by the max pooling layer are fed to a fully connected layer to match the desired feature size. Then the features go through a recurrent neural network, specifically an LSTM, that takes in 10 samples per timestamp for as many channels as the number of features. It features 2 layers and both the input size and the hidden size equal the feature dimension. The rest of the network follows the same structure as DeepVO as shown in the image.

3.2.3 Wheel Encoder Net

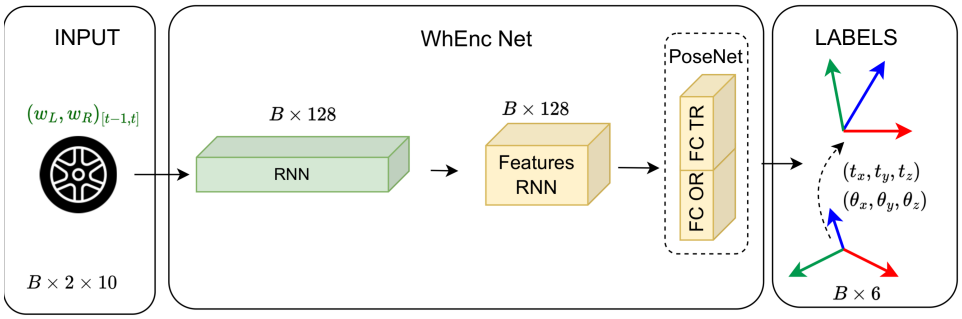

Figure 19: Wheel Encoder Net pipeline.

The wheel encoder uses a recurrent neural network with a similar structure as the one used to encode the time series of the IMU. The wheel odometry from the Complex Urban dataset is pre-calibrated, meaning that the

dataset contains the parameters to reconduct the wheel odometry to the differential drive kinematic model, as explained in section 2.1.5. The input to the LSTM is bidimensional, the LSTM takes in 10 samples per timestamp. The rest of the network follows the same structure as DeepVO, as shown in image 19.

3.2.4 Selective sensor fusion

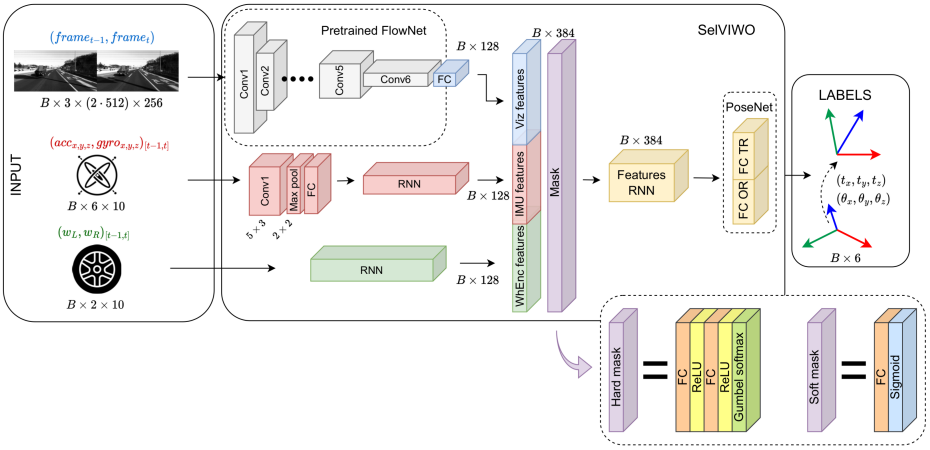

Figure 20: Full pipeline of End-to-end Sensor Fusion for Visual-Inertial-Wheel Odometry.

As seen in figure 20, the entire pipeline is a combination of the first tree, with the addition of a mask to fuse the sensors' features. Features' fusion plays a prominent role in the development of robust sensor fusion. Unrolling all the complexity and underlying ideas and maths behind the fusion strategy would consist itself in a research topic, thus, in this thesis, we are focusing on a proof of concept of generalization of the approach implemented in [14]. The main idea of fusion is to develop an attention mechanism that selects what latent features need to be used to determine the robot's position, choosing from the space of all the latent features extracted from different sensors with the previously presented methods. In this work, two types of fusion have been studied, using a mask that selects the features that are let through the pipeline: soft and hard masks. The two strategies will now be explained as presented in the paper [14].

The soft mask is a mask of the same size as the concatenated visual, IMU, and wheel encoder features, which is made of a fully connected layer

and sigmoid activation function. The idea is to train the mask to resize the features according to the effect their forward pas would have on the accuracy of the estimation, thus, the features are element-wise multiplied by the mask before entering the temporal modeling, meaning the features' RNN.

The features can be symbolically represented as a_v, a_i, a_w, while the mask is

$$s = \sigma([a_v; a_i; a_w])$$

Making the output of the soft mask fusion strategy:

$$g_{soft}([a_v; a_i; a_w]) = [a_v; a_i; a_w] \odot s$$

The author describes the soft mask as deterministic since the model chooses, based on the magnitude of the input features, which are worth passing through. However, this approach does not leave room for randomness and noise in the input both in training and testing time, making the system less robust and open to handle more variable scenarios.

For this reason, the authors designed a fusion strategy that, as a hard mask (acting as a switch), draws the selected feature from a Bernoulli probability distribution. However, to make back propagation possible in such a probabilistic layer, the Gumbel-softmax trick was adopted to make the layer differentiable. The Gumbel-softmax trick [21] is a way to sample from a categorical distribution by first sampling from a Gumbel distribution and then using the softmax function to map the samples to a probability distribution. The Gumbel distribution is used to model the distribution of the maximum values in various distributions, in layman's terms, to represent the distribution of the highest values of the features. The function represents the Gumbel distribution is:

$$Z = onehot(argmax_i G_i + log(\pi_i))$$

However, the argmax function is not differentiable, so the softmax is used as a differentiable alternative to argmax, thanks to the temperature parameter that controls how closely the function approximates discrete vectors. In particular, the equation for the samples vectors is:

$$y_i = exp((G_i + log(\pi_i))/\tau) / \sum_i exp((G_i + log(\pi_i))/\tau)$$

In this case, the mask is resampled from a probabilistic Bernoulli distribution, which is conditioned to the feature magnitude but adds random noise to achieve higher robustness. As a result, the masks are built as follows:

$$s_v \sim p(s_v|a_v, a_i, a_w) = Bernoulli(\alpha_v)$$

$$s_i \sim p(s_i|a_v, a_i, a_w) = Bernoulli(\alpha_i)$$
$$s_w \sim p(s_w|a_v, a_i, a_w) = Bernoulli(\alpha_w)$$

In the same fashion as the soft fusion, the mask multiplies the features, but in this case, the mask is discrete and not continuous, letting the whole magnitude of the feature pass or zeroing it.

$$g_{hard}([a_v; a_i; a_w]) = [a_v \odot s_v; a_i \odot s_i; a_w \odot s_w]$$

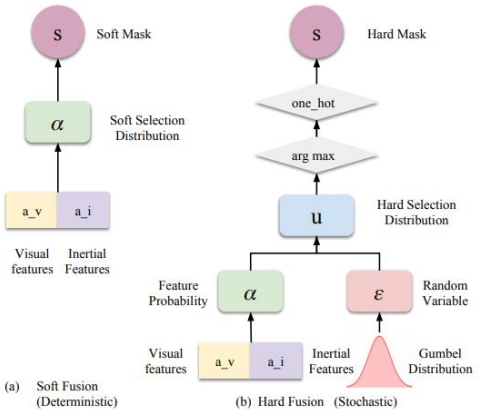

Figure 21: Rapresentation of the selective hard and soft fusion strategies. Both the masks are trainables, the hard mask is stochastic to achieve robustness, and some random noise has been added. Figure from [14].

Non-official yet very informative clarifications about the Gumbel-softmax trick can be found at resource [22] listed in the bibliography list.

3.2.5 Training and test setup

The network is implemented using the open source framework PyTorch started by Meta AI. It is trained using an NVIDIA GeForce RTX 2080 Ti (11 GB RAM). The model was trained using an Adam optimizer, varying the hyperparameters for up to 200 epochs, however, practically, early stopping is applied when the best model is selected based on the validation loss. The loss function used to supervise the model is the sum of the linear and angular mean squared error, with the latter scaled by a multiplication

factor k = 1000, since the angle is expressed in radians while the linear orientation is in meters.

$$L = MSE_X + k \cdot MSE_\theta = \frac{1}{n}\sum_{i=1}^{n}((X_i - \hat{X}_i)^2 + k \cdot (\theta_i - \hat{\theta_i})^2)$$

With X being the vector of x,y, and z translations and θ the vector of roll, pitch, and yaw orientation.

The code will be publicly available as soon as we decide if this work can be submitted for publication, and it will contain all the information to be easily reproduced.

4 Results

The separate networks were tested separately before testing the compound network to assess the role of every single component of the network.

4.0.1 DeepVO

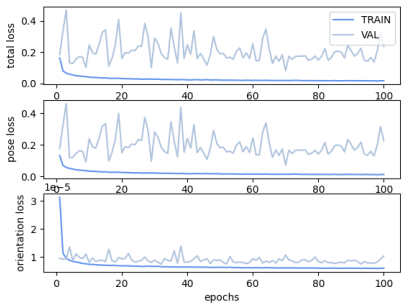

Figure 22: DeepVO training and validation loss

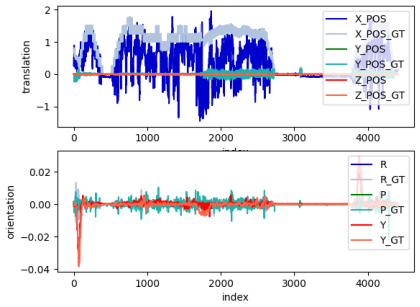

Figure 23: DeepVO results for sequence 29

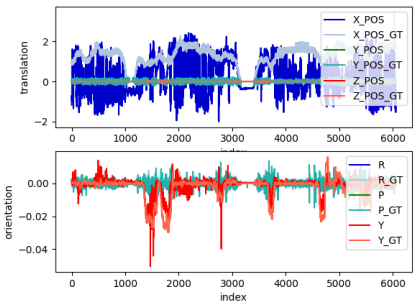

Figure 24: DeepVO results for sequence 34

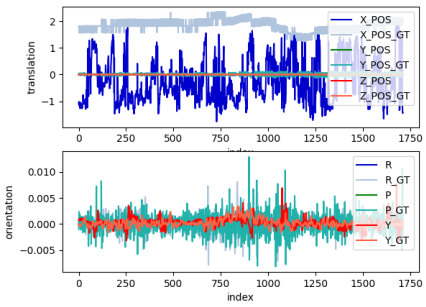

Figure 25: DeepVO results for sequence 35

The network that regresses the position using visual information from monocular camera, which was presented in [12] as DeepVO, was tested separately to assess the strengths and weaknesses of this specific sensor. Image 22 shows that the models strongly tend to overfit since the validation loss converges in about 5 epochs while the training loss keeps decreasing. Although the

attempts of reducing the overfit by applying dropout between 0.2 and 0.4, increasing the probability of data corruption (in order to learn the macro features instead of the particular features that are specific to each environment), the model has always shown overfit, even though the authors of [12] address the problem and show in their results that they were able to reduce the overfit and improve the performance of the model. The results achieved in that paper show that there is room for improvement in tuning the model, and using different methods to reduce the overfit. The results presented by the authors of DeepVO are shown in figure 26. From these results, we can notice how they managed to achieve better training since the loss function shown in figure 22 looks more similar to the overfitted model loss function shown by DeepVO authors.

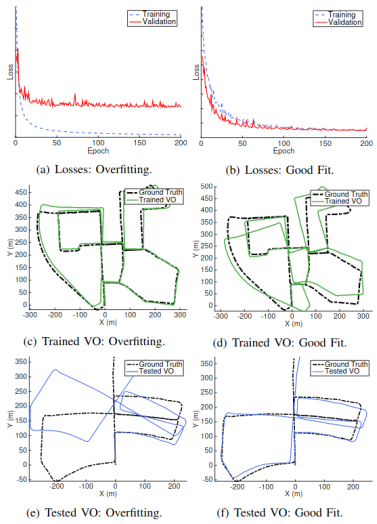

Figure 26: Comparison of overfitted and non-overfitted model from DeepVO paper [12]

Concerning the accuracy (in this case accuracy and loss are exactly equivalent since both are evaluated by using MSE) of the model, the translation loss converges rapidly to about 0.2 while the orientation loss converges at about $1e10^{-5}$. Considering that the error is calculated as mean squared error, the average validation error for every translation is about $0.45m$ while the average orientation error is about $0.2deg$. The performance of the model is shown in figures 23, 24, and 25. The figures show that the estimation tends to be very noisy and rather inaccurate and it is evident how the model strug-

gles to estimate the localization from the highway sections (figure 25) since the highway scene is monotone, all the frames look similar to the model.

4.0.2 IMU Net

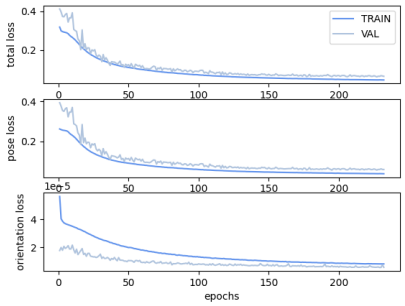

Figure 27: IMUNet training and validation loss.

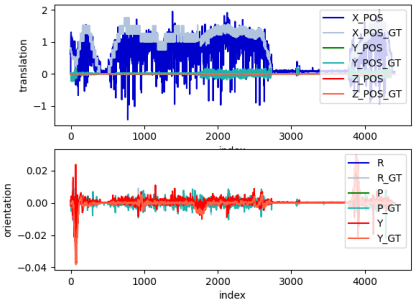

Figure 28: IMUNet results for sequence 29.

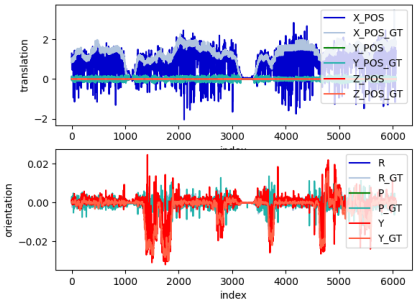

Figure 29: IMUNet results for sequence 34.

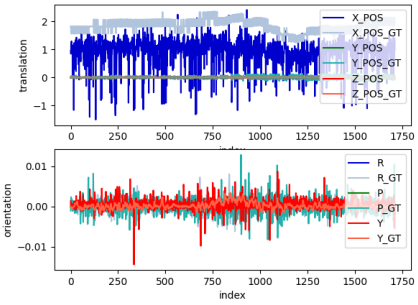

Figure 30: IMUNet results for sequence 35.

The network that regresses the position merely using inertial information, which we call IMU net for simplicity, was tested separately to assess the strengths and weaknesses of this specific sensor. Image 27 shows that the model's loss converged in about 150 epochs, showing a total validation loss of 0.2, but it is more interesting to break it down and observe separately the translation and orientation loss: the translation loss ("pose loss" in the figure) for the validation converges at about 0.1 while the orientation loss

converges at about $1e10^{-5}$. Considering that the error is calculated as mean squared error, the average validation error for every translation is about $0.32m$ while the average orientation error is about $0.2deg$. The performance of the model is shown in figures 28, 29, and 30. The figures show that the model performs well where the sequences have higher variability since the IMU sensor works by integrating the small variations perceived but those variations are often difficult to be separated by noise. In particular, figure 29, which presents sharp turns and rapid accelerations, shows the best accuracy compared to figure 30 which is a straight highway section. In this case, the model learns the small variations in linear acceleration but not the small variations of orientation, however, the model can be considered as performing well since the function that maps the linear acceleration and angular velocity to the linear translation and yaw angle is an integration, which requires the knowledge about the initial velocity, which this sensor alone cannot achieve. This explains the constant offset between ground truth and estimated linear translation because, without information about the initial velocity, the best guess for the model is to take the average of the training set velocity as the initial velocity. This problem is solved in real-life scenarios because the localization starts with the knowledge that the vehicle is still. Again, this feature is typical of pure inertial localization, either deterministic or end-to-end. Moreover, the results show how noisy the estimation is compared to the ground truth: the convolutional filtering explained in section 3.2.2could be improved, in particular acting on longer sequences to filter out the noise. This will be further addressed in the discussion, section 5.

4.0.3 Wheel Encoder Net

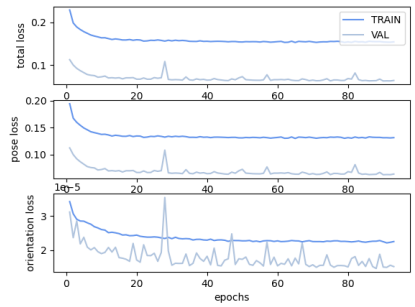

Figure 31: WhEncNet training and validation loss.

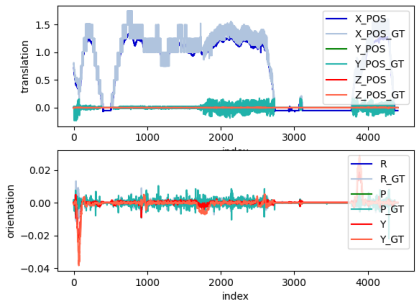

Figure 32: WhEncNet results for sequence 29.

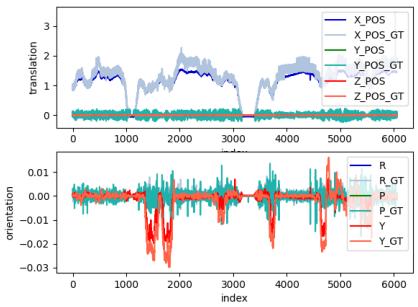

Figure 33: WhEncNet results for sequence 34.

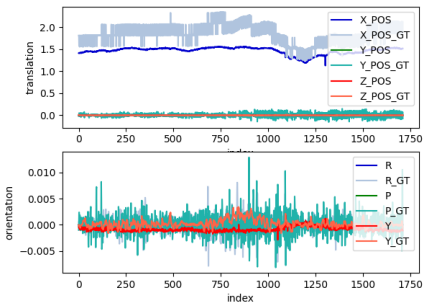

Figure 34: WhEncNet results for sequence 35.

Figure 31 shows that the model's loss converged early, in about 30 epochs, showing a total validation loss lower than 0.1, but it is more interesting to break it down and observe separately the translation and orientation loss: the translation loss ("pose loss" in figure) for the validation converges at about 0.05 while the orientation loss converges at about $1.6e10^{-5}$. Considering that the error is calculated as mean squared error, the average validation error for every translation is about $0.22m$ while the average orientation error is about $0.23deg$. The offset between training and validation loss can be explained by the data augmentation performed on the training set while the plain dataset is used for validation. As the lower validation loss shows,

the data augmentation allows better generalization in this case.

The performance of the model is shown in figures 32, 33, and 34. The plots show high performances of the model as long as the magnitude of the label to be estimated is small, and it is particularly visible in the linear translation of figure 34 and the orientation in 33. This is probably due to the fact that the training set does not contain enough of these extreme situations, so the data balancing explained in section 3.1.6 could be further tuned, letting faster sequences through while it is impossible to generate sharper corner, but more data containing sharp corners need to be acquired.

Since the wheel encoder is not affected by high-frequency noise the inferred positions are much less noisy than those achieved by using the IMU.

4.0.4 Soft fusion without data degradation

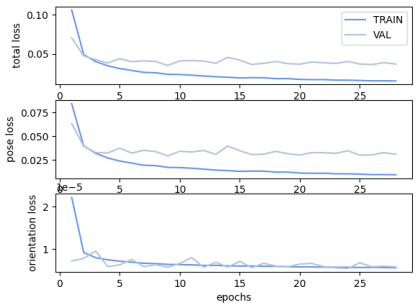

Figure 35: Soft fusion training and validation loss.

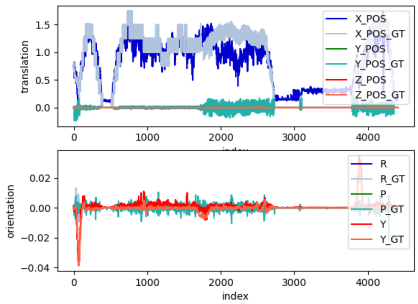

Figure 36: Soft fusion results for sequence 29, no data degradation.

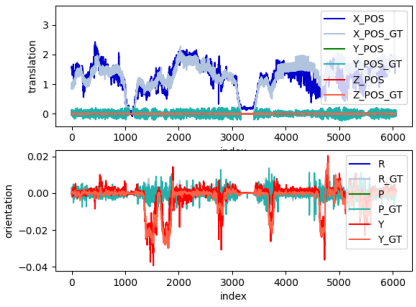

Figure 37: Soft fusion results for sequence 34, no data degradation.

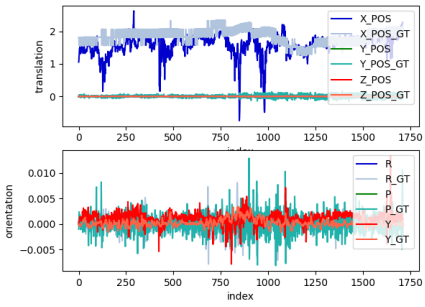

Figure 38: Soft fusion results for sequence 35, no data degradation.

The model of the soft fusion is first assessed without data degradation in the testing sequence. Concerning the accuracy of the model, the translation loss converges rapidly to about 0.04 while the orientation loss converges at about $0.3e10^{-5}$. Considering that the error is calculated as mean squared error, the average validation error for every translation is about $0.2m$ while the average orientation error is about $0.1deg$. The performance of the model is shown in figures 36, 37, and 38. It is interesting to notice that the accuracy of the fusion is closer to the accuracy of the wheel encoder alone but the estimated pose is noisier. In order to train the fusion mask, some noise has been injected into the sensor's measurement, this could explain the variance of the signal.

4.0.5 Soft fusion with data degradation

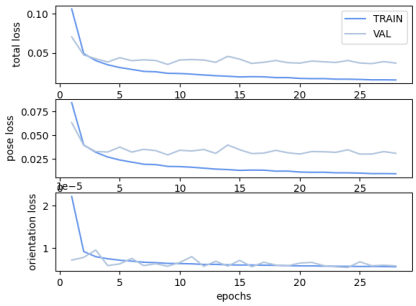

Figure 39: Soft fusion training and validation loss.

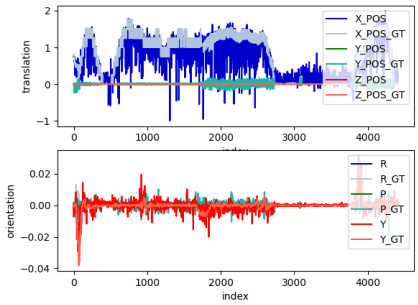

Figure 40: Soft fusion results for sequence 29, with data degradation.

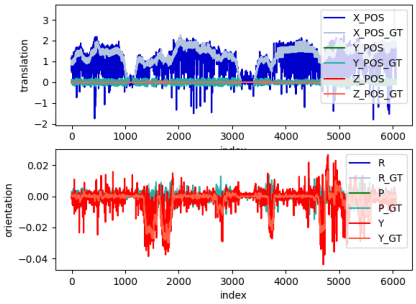

Figure 41: Soft fusion results for sequence 34, with data degradation.

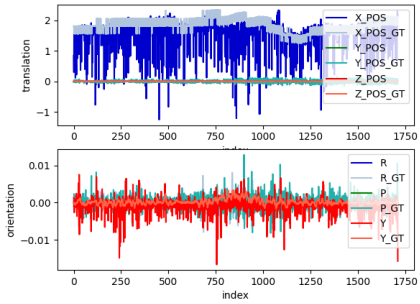

Figure 42: Soft fusion results for sequence 35, with data degradation.

When the sensors data are degraded, the estimation variance increases, making the signal more turbulent, as figures 40, 41, and 42 show. The sensor degradation is performed as explained in section 3.1.5. In particular, in this specific case, every visual degradation method can occur with a probability of 0.3, while the IMU and wheel encoder sensors degradation can occur with a probability of 0.2.

4.0.6 Hard fusion without data degradation

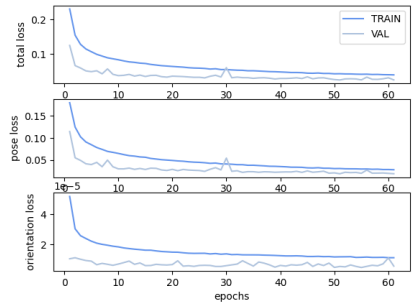

Figure 43: Hard fusion training and validation loss.

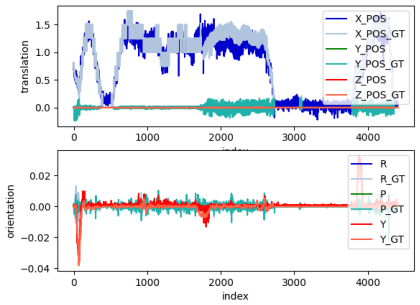

Figure 44: Hard fusion results for sequence 29, no data degradation.

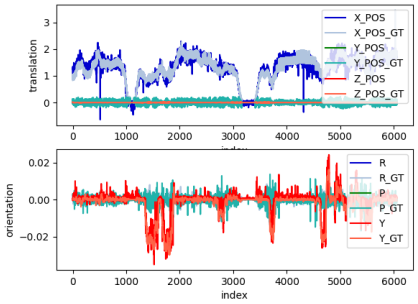

Figure 45: Hard fusion results for sequence 34, no data degradation.

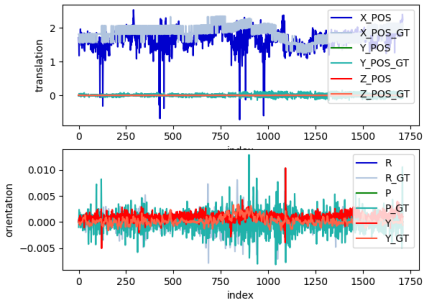

Figure 46: Hard fusion results for sequence 35, no data degradation.

Figures 44, 45, and 46 show the performance of the hard fusion without data degradation while figure 47 shows the training and validation loss curves. We can observe that the translation loss converges rapidly to about 0.02 while the orientation loss converges at about $0.3e10^{-5}$. Considering that the error is calculated as mean squared error, the average validation error for every translation is about $0.14m$ while the average orientation error is about $0.1deg$. We can also notice that the learning rate is lower in this case since the curves decrease more slowly. We can also notice an improvement in the accuracy between hard and softmax, which can be noticed both from

a lower validation loss (the translation loss in particular) and from the test sequences graphs.

4.0.7 Hard fusion with data degradation

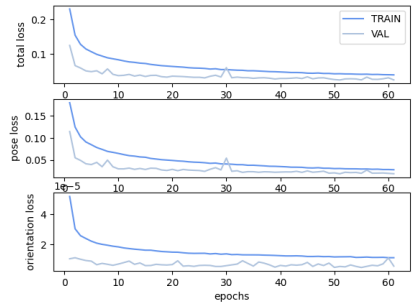

Figure 47: Hard fusion training and validation loss.

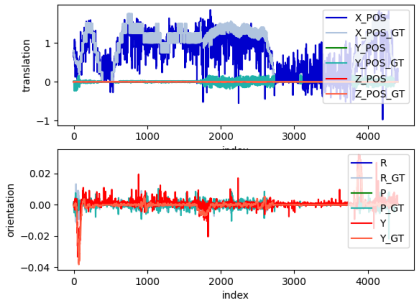

Figure 48: Hard fusion results for sequence 29, with data degradation.

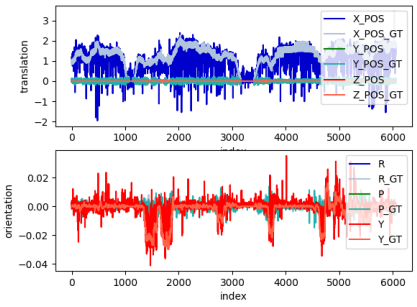

Figure 49: Hard fusion results for sequence 34, with data degradation.

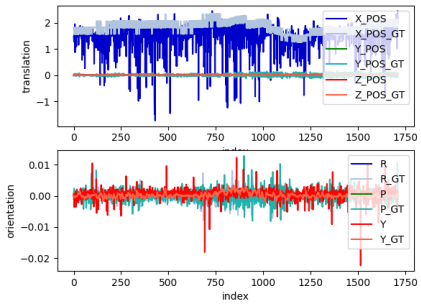

Figure 50: Hard fusion results for sequence 35, with data degradation.

As well as the soft fusion, when the sensors data are degraded, the estimation variance increases, making the signal more turbulent, as figures 48, 49, and 50 show. The sensors are degraded with the same probability of soft fusion. The results are similar to the soft fusion but show an improved translation estimation. Thus, we can conclude that the hard fusion model performs better than the soft fusion model; hence, it will be used to compare with the Kalman Filter benchmark in the next section.

4.1 Robustness to noise

4.1.1 Benchmark: Kalman Filter

A Kalman Filter was implemented to compare the method's robustness with the existing filtering approaches. The Kalman filter is purely based on inertial sensor and wheel odometry, using the calibrated kinematic model of the vehicle provided by the Complex Urban Dataset. The basic behavior of the Kalman Filter is:

1. Prediction $x_{t+1} = x_t$ and $\omega_{t+1} = \omega_t$
2. Update based on measured IMU and wheel odometry
3. Repeat

The Kalman Filter performs poorly with non-gaussian noise (e.g., the typical offset error in the gyroscope or accelerometer), and, to limit the effect of the bias term on some sensors, the corresponding covariance can be tuned to weigh less the information coming from that sensor. Otherwise, a static calibration can be performed to zero the bias term out. The kinematic model of the Kalman Filter assumes that linear and angular velocities stay constant if sensor updates are not available and will rely on this knowledge in case the model covariance is smaller than the observation covariance. For this task, the chosen kinematic model was accurate enough to model the vehicle, however, in the case of a non-linear vehicle model or non-linear measurement model, the extended Kalman Filter should be used, which increases complexity. In general, the model shows a high accuracy after properly tuning the parameters (model covariance matrix and sensor covariance matrix), however, adjusting the parameters can be time-consuming. However, interesting comparisons can be performed to assess the relative robustness of the Kalman Filter with respect to the End-to-end Selective Sensor Fusion presented in this thesis.

4.1.2 Results

The comparison between the presented Selective Sensor Fusion and the Kalman Filter is performed using a custom dashboard that allows visualizing, frame by frame, the input and output of the system, as well as the values of the selected features and some statistics about the test session. An example of one frame of the dashboard is shown in figure 51, however, we provide, as follows, the list of the links to the video showing a live sequence

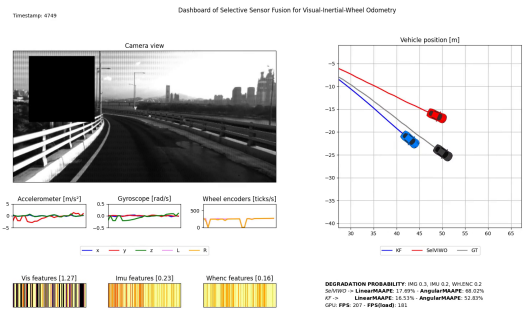

Figure 51: Example of the dashboard used to compare the localization methods.

of some example scenarios where the system has been tested, followed by some relevant observations:

- Video showing the robustness of the model in urban area
- Video of the model outperforming the Kalman Filter in highway sequence with data degradation
- Video of the model in curvy highway road

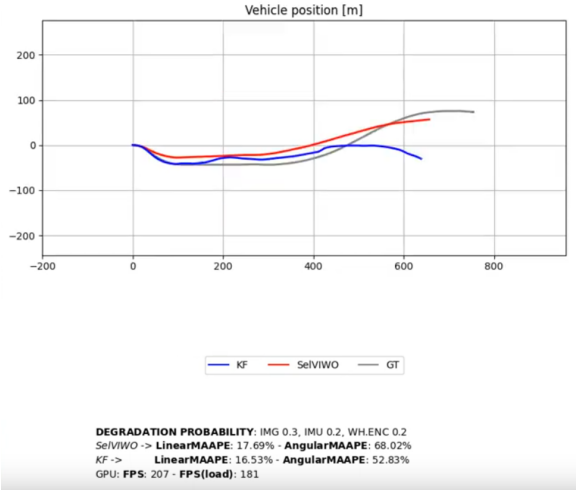

Figure 52: Dashboard comparing the methods in a section of curvy highway with degraded sensors' data.

- Video of the model failing to compensate for poor lighting conditions

Figure 53: Dashboard showing the model failing to compensate for the loss of visual information when no degradation is applied to the visual data.

From the linked videos, we can draw the following observations:

- These videos exemplify the concept depicted in figure 1 since the error in the vehicle's position can be seen growing. This increase is because the position is the sum of all the estimated relative transforms; since every transform is affected, to a certain extent, by error, the error in the absolute localization corresponds to the sum of the errors of the relative transforms, therefore, the error will grow larger without a chance to converge.
- We can notice how the ground truth is slightly shaky due to the inaccuracy of the sensors used to estimate the ground truth. Interpolating the poses would improve the performance of the whole model, but this will be discussed in section 5.2.
- The dashboard shows a color map of the values of latent features encoded for the different sensors after they pass through the fusion mask. This visualization aims to keep track of the neural activity of the sensors' networks; the higher the value of a feature, the more it will contribute to the regression down the pipeline. It is interesting to notice that the visual features correspond to high neural activity in a typical scenario (e.g., figure 51), but we can notice how the intensity of the color of the visual features drastically decreases when the sunlight flashes the camera in figure 53.
- The information printed in the video shows that the average error of SelVIWO is similar to the error of the Kalman Filter. Still, the

SelVIWO model is affected by a small constant error (MAAPE, Mean Arctangent Absolute Percentage Error). It manages to compensate for the noisy measurements, while the Kalman Filter, being a feed-forward method, does not have any way to compare the sensors' information and determine whether they are genuine. As a result, the accuracy could be improved by further augmenting the data and interpolating the ground truth, or in general, using an accurate localization dataset. Thus, we can conclude that when the sensors' data is noisy or lacking, SelVIWO achieves higher robustness. As a side note, MAAPE is used since the sensor measurements range includes 0 (the orientation is even centered in 0), and the standard percentage error would be undefined if the ground truth is 0 or lead to undesired explosions to infinity in the neighbor of 0.

4.2 Comparison with previous results

As mentioned, the presented pipeline is inspired by the state of the art in end-to-end sensor fusion for localization [14]. In the paper, only IMU and visual odometry are fused using the KITTI dataset, however, the author states that the model uses primarily the camera features against the IMU features; this is probably because of the KITTI dataset not being correctly aligned in the time since, as explained in section 3.1.2, no timestamps are provided, making the time series regression more prone to error. Moreover, since we are using a different dataset, it is impossible to compare the results of the two methods.

5 Discussion

5.1 General observations

The studied end-to-end sensor fusion proves to be a field where machine learning can play an essential role in inferring and encoding the observation model (mapping IMU and wheel encoder to the robot's position). However, as explained in reference [23], it shows a flaw typical of multimodal approaches: building networks that exploit supplementary, redundant information to achieve robustness is more complex than implementing a network that extracts only complementary information. The model would be expected to reach at least the accuracy of the best of the three input sources but instead seems to rely in particular on the vision data, although it is the most unreliable. The reason could be that the camera provides more information due to the quantity and diversity of the pixels; hence, the model searches for any correlation between all this information and the correct label.

Moreover, particular care must be paid to adding enough variability and noise in the dataset to increase the model's ability to detect the noisy signal and focus attention on the correct sensors. Based on the training, the loss landscape in such a dynamic environment seems shallow and has numerous local minima. Studying and analyzing the effects of different loss functions could be beneficial to overcoming this issue and could be focused on performing a balanced training of positions and orientation. Indeed, due to the different scales of angular and linear components, and in particular since the angle estimation and ground truth are in the neighborhood of zero, using a relative error is inappropriate since the error would explode with ground truth approaching zero. Thus, the scale of the sensors' measurement is essential. A way to overcome this issue could be by using MAAPE (Mean Arctangent Absolute Percentage Error) as a loss function: it would be scale-independent, allowing the comparison of the two measurements.

As briefly explained in the results, the sampling frequency of the raw sensors has a critical role. To recap the sensors' frequencies, the camera frames are captured at 10 Hz, the wheel encoders and IMU are sampled at 100 Hz, while the ground truth poses are sampled at 10 Hz (after preprocessing, section 3.1.3. However, the IMU signal is affected by high-frequency noise (due to electrical fluctuations in MEMS sensors) and lower frequency (due to vehicle springiness and vibration resonance due to road conditions). In this implementation, a convolutional network is used to perform the filtering, but it can only filter out higher frequencies (between 10 and 100

Hz). For this reason, in future developments, the model could be adapted to perform the fusion at a further stage in the pipeline (before or after the PoseNet's fully connected layers) so that filtering can be performed on the time series at a lower frequency. This modification would not only improve the sensor's time series filtering but also would make the fusion more homogeneous and explainable: instead of different features extracted from each sensor, the model would fuse spatiotemporal features that are more relatable to the label.

5.2 About the ground truth

As this Thesis presents a supervised learning approach, the ground truth poses play a prominent role in the model's accuracy.

The ground truth poses are themselves estimated, meaning that they had been acquired by using a more accurate localization method, in particular SLAM. As a result, the poses are affected by noise; in fact, the authors of the dataset [18] suggest not to use the poses for localization purposes since the accuracy depends on the environment's complexity. With ComplexUrban being chosen for the reasons presented in 3.1.2, the dataset has been preprocessed to perform the proper time alignment between the sensors, but the performances could be improved by interpolating the ground truth poses in the 3D space to remove the noise, thus, smoothening the trajectory. A stable and accurate ground truth would improve the estimation of the poses since they are relative poses: the relative transformations are smaller than the global poses; as a result, their magnitude can be similar to the noise magnitude, losing the information. This problem can be seen particularly in the orientation; since the rate of rotation is usually slow for four wheels vehicles, noise can make the model's performance very detrimental. Moreover, interpolating the poses would smooth down the loss landscape, thus, improving the SGD's ability to reach a real minimum.

Considering the ground truth is generated using deterministic methods like SLAM, so the estimated position will be in the best case scenario as accurate as the ground truth, one question needs to be answered to justify the study of this approach: what is the advantage of using a learned model? If looking at the mere short-term period, this answer cannot be successfully answered, but the answer can be formulated with a bit of foresight. This work aims at studying and generalizing how to learn and latently represent state estimation (the states are position and orientation in this specific case) and to fuse the latent features in an intelligent, robust way. In future developments, as explained further on in this discussion, the training

paradigm will be unsupervised or self-supervised (e.g., the self-supervised method presented in 2.2.4), therefore, the performances could improve compared to the legacy non-learning methods. To rephrase the concept: using supervised learning is a temporary, intermediate step to studying how to design a model to perform robust localization.

5.3 Gain points

The presented methods, although still in an initial phase and more work needs to be carried out to develop a full-fledged end-to-end sensor fusion for localization, aim at generalizing a sensor fusion approach as a skill that can be learned by any robot requiring localization without human intervention and tuning of the parameters. We see this generalization as a small step further towards general intelligence since it provides a structure to solve the problem of localization. The benefits for the industry lay in the zeroed costs for development and tuning while achieving robustness that vanilla fusion methods cannot guarantee since the development of custom deterministic methods for robust localization can be expensive and difficult to generalize. By using End-To-End Selective Sensor Fusion at the current stage of development, generalization can be easily handled by acquiring more data. Moreover, adapting the pipeline described in reference [15] and depicted in figure 4, the same methods can be trained using a self-supervised approach, which does not require ground-truth data acquisition.

5.4 Pain points

The pain points of the presented methods are presented in the following list:

- Computation-heavy and memory intensive: although the model runs at about 200 Hz on GPU and about 40 Hz on CPU, the whole model takes approximately 6 GB of memory, which is a considerable weight compared to methods like Kalman Filter, which make small matrices that multiplied for a few times at every timestep, according to the number of used sensors and the size of the considered state-space.

- Relatively low data efficiency. The results show the importance of a balanced and diverse dataset. The diversity cannot often be increased by augmenting the data, hence, the required dataset is relatively large. To give a reference, the dataset used for this work totals 255 GB (about which approximately 60% of it is used for training) before even applying the data augmentation.

- Explainability and reproducibility. These two problems appear to be common problems of Deep Learning, fostering some suspicion about this field of science. The explainability issue arises because representing latent features is, as the name hints, difficult, and their interpretation is rather conjectural; moreover, the large amount of features does not simplify their examination. This issue is horizontal to the DL field and involves mainly the understanding, debugging, and development of the model. In contrast, reproducibility is an issue that is typical of (at least) the DL research in the field of robotics, according to the experience gathered in this work. Usually, source code, when published, is not well documented, mostly buggy, and the papers cannot explain in detail the parameters that could be found in a published code. Moreover, datasets like the mentioned KITTI and Complex-Urban can be accessed only by researchers who are affiliated with an educational institution, which not only cuts the individual researchers out of this field but also forces the other researchers who want to reproduce the code to download and organize the data in such a way as the authors did. Hence, we will carefully publish all the required steps to reproduce the results and improve the code.

- Requires ground-truth: although the labeling itself is not labor-intensive since it is achieved by using specific sensors, gathering the data requires time and the data accuracy plays an important role as explained in section 5.2.

5.5 Future improvements

Although some ideas for future improvements have been hinted at in this discussion, the following list aims to summarize them and include additional ideas to give a research direction to whom will continue this work, based on months of work in this field:

- Implement and test the fusion mask at different stages of the pipeline. We expect the late fusion (right before or even after the fully connected layers that constitute the network) to improve the model's performance since, by that stage of the pipeline, the attentive fusion mask would be trained to add the time information to the latent features. Moreover, the later fusion would make the features more homogeneous and easier to represent since they are more similar to the labels. A representation of this is shown in figure 55

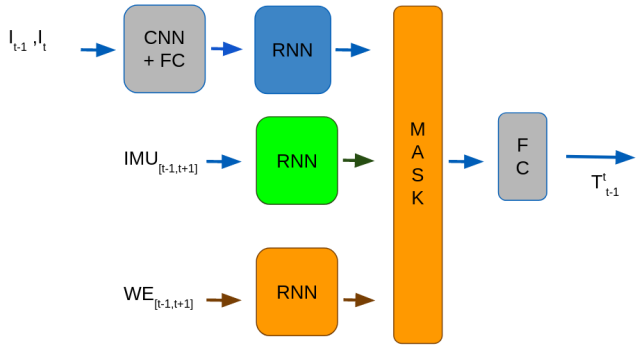

Figure 54: Pipeline featuring late fusion. The gray boxes are the component that does not change position from the current implementation to the late fusion architecture.

- Perform "individual" sensors training and fusion: to better focus the study on the selection alone, the separate sensors pipelines can be trained separately and the fusion can be trained later using the pre-trained sensors processing networks. This approach would allow us to study the fusion more in detail and test it by injecting noise into the sensors.

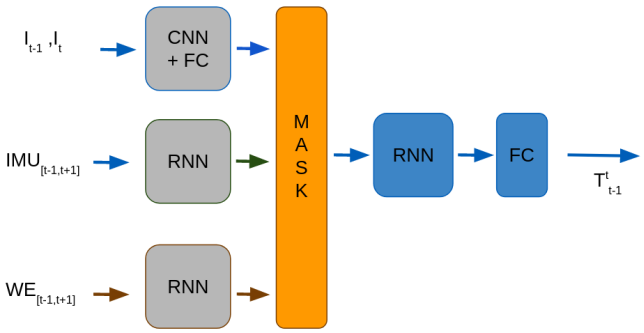

Figure 55: Pipeline training the isolated fusion. The gray boxes are the pre-trained models when training the fusion network.

- Study and implement hybrid methods, for example by deterministically extracting the models that map the raw sensors measurements into a position estimation and fuse the processed information in a

learning-based fashion. This design choice would make the model leaner, reducing the computation load.

- Further improve the data in quantity and quality. Based on the results obtained in this work, we suggest improving the data augmentation (e.g., by combining the augmentation methods) and interpolating the ground-truth poses.

- Develop and test the self-supervised methods such as the discussed SelfVIO 2.2.4.

- Generalize the method to other sensors. For example, interesting results could be evaluated by testing the pipeline with multiple cameras (e.g., using the Waymo dataset [24], featuring five camera streams). Another idea worth exploring is to fuse the latent features extracted from the LiDAR data, for example using a pipeline such as the one presented in reference [25].

5.6 Digression about biological localization

As a digression, it is interesting to consider how human localization works and use such considerations to improve the machine learning model.

Humans fuse information from different sensors (sight, touch, hearing, inertial sensors located in our ear, and even smell) and can often select what sensors to trust in case of incoherence between our senses, because we have built our model for localization based on our experience, being able to determine which sensor is the most likely to be correct, hence, constantly tuning our model with real-time feedback, using a reinforcement learning approach. For example, an aging person whose sight is deteriorating would be aware that their sight has deteriorated, as a consequence, they would learn to use other senses. An edge case is represented by blind people, who learn outstanding localization skills by improving their sense of touch, hearing, and even smell. As a result, to achieve human-like localization or improve our AI models learning from the human biological world, the machine learning model should be able to perform continuous learning, using the reinforcement provided by the real-life feedback, without supervision, as opposed to the presented supervised learning approach.

Another relevant observation regards the accuracy humans can achieve with their unsupervised, continuous reinforcement learning, particularly how it relates to the environment where the human agents need to localize themselves. The more information is in the environment, the more accurate the

localization will be. For example, assuming the environment has already been visited, an urban environment contains more information than a plane desert, let alone the desert features are dynamic due to the wind. This issue seems to represent a drawback of human-like localization. However, it is the opposite since the same approach would allow the encoding and fusing of the information from sensors humans are not equipped with, such as GNSS sensors. In the same way, as humans already do with their senses, the model would learn when to rely on GNSS data and when to discard it.

Furthermore, human localization is based on experience and some map representation that we build in our minds when we visit new places; knowing how this map is long-term stored and accessed in the biological domain would help improve the robot model in terms of accuracy and efficiency. In particular, we remember specific features of the environment we have already explored and the distance between them for a long time. Based on the available tools that deep learning offers, the best tool that allows focusing on specific features and remembering them over time is the transformer network. As a result, an exciting development for sensor-fusion localization could be achieved by taking as an example the work presented in [26], where a transformer network is used to focus on some specific features of the environment, and those features are used to build a latent map representation of the environment.

The topics of this digression, although not strictly relevant to the industry-related research and based on subjective conjectures, widen the horizons of the current localization methods almost to the extent of artificial general intelligence; as a result, they foster the research in the field by making it more ambitious and more relatable to researchers.

6 Conclusions

This thesis studied, developed, and expanded the state of the art for end-to-end sensor fusion for robust localization, specifically the fusion of visual odometry with inertial sensors and wheel encoders in GNSS-deprived circumstances, such as urban environments or tunnels for vehicles.

This work constitutes a step further in generalizing the approach for different sensors, studying the strengths and weaknesses, and keeping the legacy methods as a reference. In particular, this method allows to infer the model that estimates the localization of an agent with inertial sensors, wheel encoders, and camera feed as the input sources. The presented methods leverage Deep Learning, particularly by implementing a pipeline that generates a latent representation of the information from the sensors (features extraction), selects such features by an attentive mechanism, and fuses them to provide the most robust and accurate estimation of the agent's egomotion.

In this work, we presented how developing end-to-end localization, as in general for Deep Learning, puts fundamental stress on the data and its preprocessing. This thesis provides an example of data preprocessing to maximize the results and the variety of the dataset.

As far as the results are concerned, the presented model achieved similar results compared to the Kalman filtering approaches when the sensors are affected by noise, proving the method can achieve robust localization. However, the approach is outperformed by a well-calibrated Kalman Filter when the sensor data is not affected at all by noise. The results confirm that the presented methods can increase the performances of localization models, but, at the current state of the art, they should be intended as complementary components rather than entirely stand-alone.

The vastity of this field makes it impossible for this problem to be treated with the level of detail needed to solve it completely. This work can be improved in many directions suggested in the discussion section. Some ideas are proposed to enhance and expand the model according to the experience gathered in these months of work, but any attempt is highly encouraged. The code will be refactored and made publicly available to unleash the power of the deep learning and robotics community.

References

[1] Cecil Adams. *Is "dead reckoning" short for "deduced reckoning"?* 2002. URL: https://www.straightdope.com/21343189/is-dead-reckoning-short-for-deduced-reckoning (visited on 05/11/2022).

[2] Sudarshan S. Chawathe. "Beacon Placement for Indoor Localization using Bluetooth". In: *2008 11th International IEEE Conference on Intelligent Transportation Systems*. 2008, pp. 980–985. DOI: 10.1109/ITSC.2008.4732690.

[3] S. Atiya and G.D. Hager. "Real-time vision-based robot localization". In: *IEEE Transactions on Robotics and Automation* 9.6 (1993), pp. 785–800. DOI: 10.1109/70.265922.

[4] Deqing Sun et al. "Learning optical flow". In: *European Conference on Computer Vision*. Springer. 2008, pp. 83–97.

[5] Guoqiang Xu and Xiuyun Meng. "The MEMS IMU Error Modeling Analysis Using Support Vector Machines". In: *2009 Second International Symposium on Knowledge Acquisition and Modeling*. Vol. 1. 2009, pp. 335–337. DOI: 10.1109/KAM.2009.287.

[6] Elder M Hemerly. "MEMS IMU stochastic error modelling". In: *Systems Science & Control Engineering* 5.1 (2017), pp. 1–8.

[7] He Song and Shaolin Hu. "Open Problems in Applications of the Kalman Filtering Algorithm". In: *Proceedings of the 2019 International Conference on Mathematics, Big Data Analysis and Simulation and Modelling (MBDASM 2019)*. Atlantis Press, 2019/10, pp. 185–190. ISBN: 978-94-6252-811-6. DOI: https://doi.org/10.2991/mbdasm-19.2019.43. URL: https://doi.org/10.2991/mbdasm-19.2019.43.

[8] E. Navon and B.Z. Bobrovsky. "An efficient outlier rejection technique for Kalman filters". In: *Signal Processing* 188 (2021), p. 108164. ISSN: 0165-1684. DOI: https://doi.org/10.1016/j.sigpro.2021.108164. URL: https://www.sciencedirect.com/science/article/pii/S0165168421002024.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[10] Guy Revach et al. "KalmanNet: Neural Network Aided Kalman Filtering for Partially Known Dynamics". In: *IEEE Transactions on Signal Processing* 70 (2022), pp. 1532–1547. DOI: 10.1109/TSP.2022.3158588.

[11] Siavash Hosseinyalamdary. "Deep Kalman Filter: Simultaneous Multi-Sensor Integration and Modelling; A GNSS/IMU Case Study". In: *Sensors* 18.5 (2018). ISSN: 1424-8220. DOI: 10.3390/s18051316. URL: https://www.mdpi.com/1424-8220/18/5/1316.

[12] Sen Wang et al. "Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 2043–2050.

[13] Ronald Clark et al. "Vinet: Visual-inertial odometry as a sequence-to-sequence learning problem". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017.

[14] Changhao Chen et al. "Selective sensor fusion for neural visual-inertial odometry". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10542–10551.

[15] Yasin Almalioglu et al. *SelfVIO: Self-Supervised Deep Monocular Visual-Inertial Odometry and Depth Estimation*. 2020. arXiv: 1911.09968 [cs.CV].

[16] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. DOI: 10.48550/ARXIV.1406.2661. URL: https://arxiv.org/abs/1406.2661.

[17] Andreas Geiger et al. "Vision meets robotics: The kitti dataset". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237.

[18] Jinyong Jeong et al. "Complex urban dataset with multi-level sensors from highly diverse urban environments". In: *The International Journal of Robotics Research* 38.6 (2019), pp. 642–657.

[19] Kevin M Lynch and Frank C Park. *Modern robotics*. Cambridge University Press, 2017.

[20] Alexey Dosovitskiy et al. "Flownet: Learning optical flow with convolutional networks". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2758–2766.

[21] Chris J Maddison, Daniel Tarlow, and Tom Minka. "A∗*Sampling*". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014.

[22] *DataScience, StackExchange Gumbel-Softmax trick vs Softmax with temperature*. Gumbel-SoftmaxtrickvsSoftmaxwithtemperature. Accessed: 2010-09-30.

[23] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. "Multimodal machine learning: A survey and taxonomy". In: *IEEE transactions on pattern analysis and machine intelligence* 41.2 (2018), pp. 423–443.

[24] Pei Sun et al. "Scalability in perception for autonomous driving: Waymo open dataset". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2446–2454.

[25] Huan Yin et al. "Rall: end-to-end radar localization on lidar map using differentiable measurement model". In: *IEEE Transactions on Intelligent Transportation Systems* (2021).

[26] Yoli Shavit, Ron Ferens, and Yosi Keller. "Learning multi-scene absolute pose regression with transformers". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 2733–2742.