Bike Ego-Trajectory Mapping Using a CNN-LSTM With IMU and Monocular Camera

S. Redeker[§]
Pervasive Systems Group, EEMCS faculty, University of Twente
June 28, 2024

This paper presents a CNN-LSTM-based approach for mapping bicycle ego-trajectories based on inertial-camera data, capable of accurately capturing non-straight-line movements such as turns and curves. I introduce a novel bicycle trajectory dataset that integrates recordings from a 6 DOF IMU and two cameras, supplemented by GNSS ground truth data. The study systematically evaluates 25 different CNN-LSTM configurations, varying architectures and input parameters, with the top-performing model achieving a RMSE of 4.4 m on a 134 m trajectory. The best-generalised model achieved a RMSE of 33 m on a 598 m trajectory. Although the models are not directly usable for urban trajectories, they lay the foundation for the development and subsequent adoption of bike mapping algorithms. The code and dataset are open source to facilitate further research in this area.

I. Introduction

Intelligent transportation systems are attracting attention from academia and industrial sectors because of their possible solutions to different transportation problems, such as safety, congestion and energy usage [1]. Although bicycles are an important mode of transport, they are not yet well-integrated into current intelligent transportation systems. Trajectory mapping of bikes is a step in making bikes smarter and safer, because the cyclist's location can be shared with other traffic participants and infrastructure. Trajectory mapping can eventually be extended to trajectory prediction, which could foster the development of cycling assistance systems, such as lane keeping, navigation, obstacle avoidance, and collision prevention.

Currently, little research is performed on mapping bike trajectories. Existing bike-related mapping algorithms, such as [2], work only on straight lines and cannot correctly map turns and curves. Most state-of-the-art algorithms focus primarily on cars and Unmanned Aerial Vehicles (UAVs) as data sources. The resulting methods, such as those discussed in [3]–[5], may not be directly implementable in bikes. Bikes are cheaper, have less space, have no onboard battery, and can move more unpredictably [6]. In addition to that, data from cars or UAVs is less noisy due to manufacturers' stabilisation efforts.

The second problem with current trajectory mapping algorithms is that they are not widely adopted, properly documented, or standardised. Manufacturers of self-driving cars do not publish their algorithms, and the output of research papers is often constrained to a paper with results and an unmaintained codebase. Promising candidates for widespread algorithms, such as ORB-SLAM(1/2/3), [7], or VINS-Mono, [8], require specific operating systems and a large number of dependencies. This impedes quick and rapid adoption, especially on embedded platforms.

A. Research Questions & Contributions

This paper is part of the Smart Connected Bikes (SCB) project, [9], from the University of Twente. The SCB project aims to equip (e-)bikes with an Inertial Measurement Unit (IMU) and a low-resolution camera for trajectory mapping. This setup is significantly cheaper than measurement setups from other studies. I will investigate how an existing algorithm[1], i.e. a Convolutional Neural Network with Long Short-Term Memory (CNN-LSTM), can be deployed on the low-cost setup for bicycle ego-trajectory mapping. The main research question is:

1) What is the impact of different CNN-LSTM configurations on accuracy and generalisability for bicycle ego-trajectory mapping using data from an inexpensive IMU and a monocular camera?

Additionally, I set the following sub-research questions:

1) How can we design and implement a CNN-LSTM-based model that accurately maps bicycle ego-trajectories, considering key factors such as appropriate architectures, interpolation methods, input window lengths, and generalisation to unseen trajectories?
2) What are the key characteristics and requirements for creating a comprehensive dataset for bicycle ego-trajectory mapping, and how can this dataset be used to benchmark and evaluate the performance of the proposed CNN-LSTM models?

The primary contribution of this paper will be a CNN-LSTM-based model that can accurately map bicycle movements, including turns and curves. The secondary contribution will be a novel dataset that can be used for future development.

II. Algorithm Background

In this section, I will cover the background relevant to my research. First, I will present a high-level overview of the structure of common mapping algorithms. Then, I will discuss the two algorithms I analysed and elaborate on why I picked

[§]S. Redeker, student BSc Electrical Engineering, is the corresponding author for this work: s.redeker@student.utwente.nl. The members of the responsible BSc thesis committee are: Y. Huang, Assistant Professor at the Pervasive Systems Group, D. Yeleshetty, PhD candidate at the Pervasive Systems Group, and K. Niu, Assistant Professor at the Robotics and Mechatronics group.

[1]Another algorithm, LARVIO, was also considered but eventually not used. More can be read about LARVIO in Section II-B1.

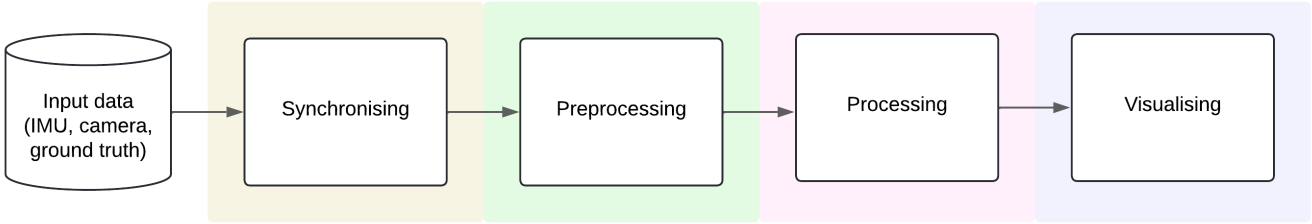


Fig. 1: High-level functional flow block diagram of an arbitrary trajectory mapping algorithm.

the CNN-LSTM architecture for further development. Lastly, I will analyse different benchmark options.

A. High-Level Overview

Most trajectory mapping algorithms share the same high-level structure. A visual representation of this structure can be seen in Fig. 1. Generally, a path prediction or path mapping program follows a four-step procedure:

1) **Synchronising.** If the various data sources, i.e. the IMU, camera, and ground truth, are not synchronised, this should be done first. Ideally, one would use a shared clock or another means of hardware synchronisation method for this. If a hardware synchronisation method is unavailable or not used, synchronisation is often done a posteriori, e.g. by analysing a calibration movement sequence.
2) **Preprocessing.** The data is preprocessed. Depending on the exact application and model constraints, this could mean that gravity is extracted, noise is removed, or quaternions are set up to visualise the orientation of an object in 3D space.
3) **Processing.** In the processing stage, one or more algorithms compute the expected trajectory, also referred to as the state. I will discuss two different processing algorithms, LARVIO and CNN-LSTM, in Section II-B.
4) **Visualising.** The last step is visualising. In this step, the trajectory of the bike can be visualised on a map. Also, one might calculate evaluation parameters, such as Root Mean Square Error (RMSE), in this step.

B. Considered Algorithms

In this section, I will discuss two existing algorithms that could be adapted for bike trajectory mapping. I picked the two algorithms based on a combination of different factors, of which the accuracy and the ease of implementation were the most important. All algorithms employ a fusion of IMU and camera data. A common approach is to feed the IMU data into a kinematic model, and camera data into a camera measurement model. Both models produce an intermediate state vector. These two state vectors are then combined into a definitive state vector based on a filter-based or optimisation-based strategy, also known as model-based or data-driven, respectively.

A model-based algorithm maps the trajectory using predefined mathematical models and physical principles. On the other hand, data-driven algorithms rely on statistical methods and machine learning techniques. These models learn patterns directly from the data without assuming an explicit underlying system model.

1) LARVIO: LARVIO is short for Lightweight, Accurate and Robust monocular Visual Inertial Odometry. It is a model-based algorithm derived from the Multi-State Constraint Kalman Filter (MSCKF) architecture.

MSCKF combines IMU data and camera images to obtain a state estimation. The gyroscope data is integrated once to obtain rotation in 3D space, and the accelerometer data is integrated twice to obtain translation. The camera images are first preprocessed to obtain a set of features, which positions are then tracked overtime to obtain a camera pose estimation. This camera pose estimation is fused with the IMU estimation to obtain a final state estimation. [10]

At the time of its creation, MSCKF claimed to outperform all existing filter-based mapping architectures on accuracy and computational complexity [10]. LARVIO improves on MSCKF by adding better support for static scenes, implementing an analytical error transition equation, and introducing calibration parameters for the camera and IMU into the model. The creators of LARVIO boast a statistically significant increase in performance, and a decrease in computational complexity compared to off-the-shelf MSCKF. [11]

LARVIO is open source, and a Docker image is available at [12]. Despite that, LARVIO requires Rosbags as wrappers for input data, which are nontrivial to create. Although Rosbags are available for existing benchmarks, such as EuRoC – discussed in Section II-C – converting bike data to the Rosbag format proved not feasible within the scope of this research.

2) CNN-LSTM: A CNN-LSTM is not one particular algorithm, but rather an architectural description of a subset of data-driven methods. CNNs are a type of neural network with great spatial capabilities. LSTM adds memory cells to the network, increasing its temporal capabilities. Given the nature of a trajectory, a model benefits from both this spatial and temporal awareness.

The main motivations behind opting for a CNN-LSTM are its adaptability, great literature coverage, and ease of implementation in Python. Using off-the-shelf packages such as Tensorflow, one can set up and train a basic CNN-LSTM system within hours – facilitating easy development and expansion. The field of bike trajectory mapping is novel. Hence, I deem it more important to prioritise ease of development over

(a) The EuRoC dataset contains only indoor images, captured from a UAV, which are not comparable to outdoor images captured by a bike.

(b) Another cyclist moves through a calibration sequence of the SCB dataset, making it challenging to deploy movement-detecting tools, e.g. OpenCV.

(c) In some parts of the SCB dataset, there are many other – moving – traffic participants. This makes it difficult to extract camera features.

Fig. 2: Three flaws with the EuRoC and SCB datasets.

the relatively slow execution speed of Python. In later stages, the models can be upgraded to faster languages, which could aid the development of a real-time solution.

C. Benchmarks & Datasets

I analysed different benchmarks from existing works and found that all tests can be divided into four categories, based on the arbitrary classifications 'trajectory length' and 'trajectory difficulty'. An overview of the four categories can be found in Table I.

	Easy trajectory (ET)	**Difficult trajectory (DT)**
Short duration (SD)	Existing work within the SCB project, e.g. by [2]. The author uses two straight lines of at most 100 meters.	Any algorithm tested on the EuRoC dataset. EuRoC runs take about a minute, which allows for relatively little time to drift.
Long duration (LD)	Some papers doing trajectory mapping with cars, such as in the first publication about MSCKF [10]. Trajectories consist of mostly straight lines with some right or left turns and relatively empty streets.	To the best of my knowledge, there are no contenders in this category yet. Any algorithm able to map (or predict) a full run of a SCB field trial would fall under this.

TABLE I: Overview of the four benchmark categories based on two arbitrary classifications. A trajectory is 'short' if it is smaller than 500 m or shorter than 5 minutes. A trajectory is 'easy' if it contains mostly straight lines, little moving traffic, and few turns.

Currently, the tests within the SCB project are limited. Existing studies use mostly benchmarks of one or more quiet, straight road(s). J. Koornstra implemented a CNN-LSTM based on MobileNet and tested it on two straight-line trajectories [2]. M. Bessi planned on testing a MSCKF-based approach on a straight line and a trajectory with six turns, but did not succeed due to poor-quality camera images [13].

Most trajectory mapping algorithms are tested on a difficult trajectory (DT) for a short duration (SD), an easy trajectory (ET) for a long duration (LD), or both. The EuRoC MAV benchmark, [14], is often used as DT/SD. This dataset contains eleven trajectories recorded from a UAV. Each trajectory is approximately a few minutes long and is recorded using stereo images and an IMU. Additionally, a high-precision ground truth is available, recorded using either a Leica MS50 laser tracker or a Vicon motion capture system. Although of high quality and often used – enabling easy comparisons – the EuRoC dataset has downsides. A UAV is more stabilised than a bike. Also, all trajectories are recorded in indoor environments, see Fig. 2a, which results in little disturbances from, e.g. other traffic. Lastly, the EuRoC dataset is relatively small, which adds complexity when setting up a data-driven algorithm [15].

Another often-used dataset is KITTI [16]. The KITTI dataset is recorded from a car, which is also more stabilised than a bike. An advantage of the KITTI dataset is that it includes real-world traffic. However, the data is unsuitable for our study because of the lack of an IMU.

To summarise, the main problem with the current benchmarks is that they are not suitable for evaluating bike trajectory mapping performance. I hypothesise that a straight-line benchmark is insufficient, especially for data-driven methods. A CNN-LSTM, for example, will bias itself towards this straight line. The algorithm will ignore any movement to the sides and will, hence, always produce a straight-line outcome. As a result, the algorithm may not properly detect turns.

1) SCB Dataset As Promising Candidate: There is one promising candidate for a novel dataset. M. Boot et al., members of the SCB project, organised the SCB field trails [17]. During the field trails, 17 participants used a sensor-equipped e-bike to cycle a predetermined route, Fig. 4a, of approximately 4 km. Each participant cycled the same route 3 times. The researchers used a ProMove-mini Wireless Inertial Sensing Platform for collecting inertial information, an Akamduman camera for full colour, full HD video footage, and a phone GPS as ground truth.

The resulting dataset contains approximately 70 GB of highly relevant and high-quality data. However, the data are not yet usable due to the lack of proper synchronisation. There is no hardware synchronisation between the different types of sensors. Hence, all synchronisation should be done afterwards. This is a time-consuming process. Firstly, the IMU data should be aligned with the camera images by analysing a calibration sequence of predetermined movements. Then, the camera should be aligned with the GPS ground truth based on timestamps. Even with partly automated tools, such as the

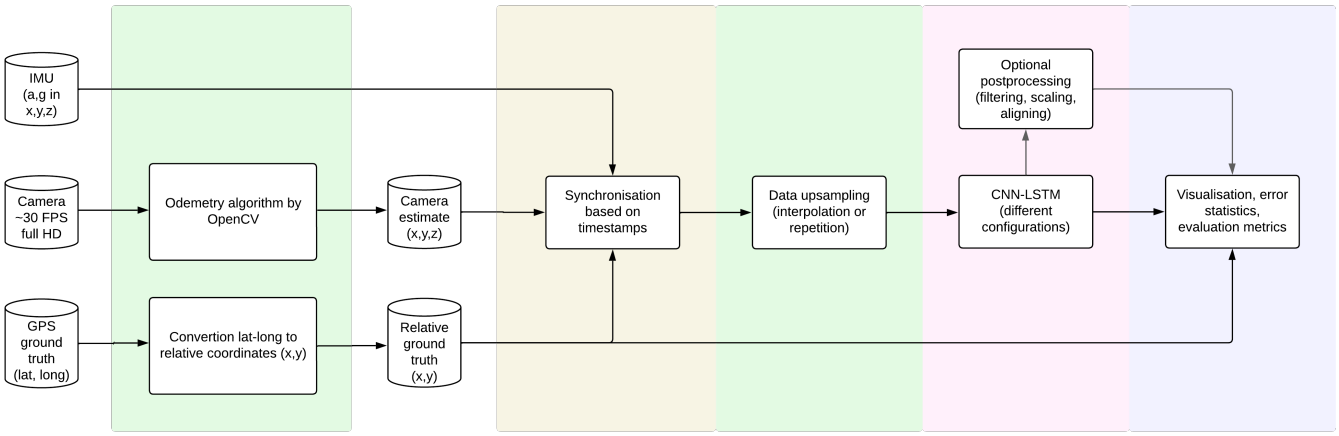
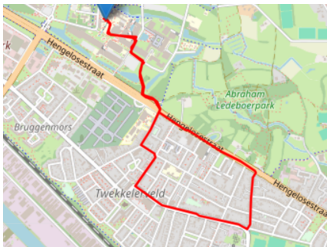


Fig. 3: High-level functional flow block diagram of the proposed CNN-LSTM-based trajectory mapping algorithm. I only used the high-resolution camera images, since these yielded better results than the low-resolution ones.

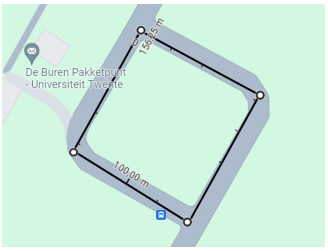
one I proposed in [18], it is a labour-intensive process prone to errors – see Fig. 2b.

In the future, when the SCB dataset is properly synchronised, it will be a valuable contribution to possible benchmarks for bike trajectory mapping. I hypothesise that the benchmark is challenging due to large quantities of other traffic (see Fig. 2c) in each frame and unstabilised data recording, but the large size of the dataset would make it possible to deploy data-driven methods confidently.

In conclusion, a new benchmark is needed to evaluate the performance of bike tracking algorithms discussed in this paper. I will propose a novel benchmark in Section III-B.



(a) The SCB trajectory. 4 km. DT/LD.



(b) The proposed trajectory. 130-150 m. ET/SD.

(c) The proposed zero-shot trajectory. 600 m. ET/LD.

(d) The proposed urban trajectory. 3.9 km. DT/LD

Fig. 4: The SCB trajectory and the newly proposed trajectories in Section III-B. The lengths and the classifications (see Table I) are given.

III. Method

In this section, I will first discuss the CNN-LSTM architectures and configurations evaluated in this study. Then, I will propose a novel benchmark that can be used. Lastly, I will elaborate on the exact training and testing procedures.

	Topology 1, based on [19], [20]	**Topology 2, based on [2], [5]**
Layer 1	Conv1D	Sequential
Layer 2	BatchNormalization	LSTM
Layer 3	MaxPooling1D	Dropout
Layer 4	Dropout	LSTM
Layer 5	LSTM	Dense
Layer 6	BatchNormalization	Dense
Layer 7	LSTM	
Layer 8	Dropout	
Layer 9	Dense	
Layer 10	Dense	

TABLE II: Low-level structure of the two baseline neural network topologies. The topologies in this table make up the 'CNN-LSTM (different configurations)' block in the high-level functional flow diagram as seen in Fig. 3.

A. CNN-LSTM Architectures & Configurations

There are multiple approaches to setting up a CNN-LSTM-based system. One approach – taken by e.g. [2] – is directly inputting camera images into a CNN layer. This approach has the advantage of requiring little data preprocessing. Another, more common approach is to preprocess the camera images first. The main advantage of this approach is that it simplifies the input to the neural network. Preprocessing could entail detecting and extracting features and feeding these features into the neural network, as done in [5], [20]. The proposed CNN-LSTM architectures used in the remaining part of this paper will follow the latter approach.

I started with two baseline topologies used in other studies and designed a pipeline, Fig. 3, for adaptation to bicycle data. The low-level structure of both topologies can be seen in Table II. Then, I devised 25 different versions on the baseline topologies, varying the following parameters:

1) **Model architecture.** I tested different architectures, and different hyperparameters within these architectures.
2) **Data input.** I varied which parts of the proposed benchmark were used for training, validation, and testing.
3) **Input window length.** I varied the input window length between 100 and 3200 samples.
4) **Data upsampling method.** The IMU data has the largest sample rate, i.e. 139 Hz, whereas the GPS ground truth has the smallest sample rate, i.e. 1 Hz. I explored two different methods to handle this sampling rate difference. The first method is to repeat copies of the last known data point, until a new data point is received. This is most comparable to a real-world, real-time application, but might cause unwanted inaccuracies in the output of the algorithm. The second method is interpolation, where I interpolate between two known data points using linear interpolation.

These 25 versions are compared in this study. An overview of the versions and their hyperparameters can be found in Section VIII-D.

B. Proposed Benchmark

A proper benchmark is important to test the performance of trajectory mapping algorithms. Since bike path trajectory mapping is novel, no standardised benchmark has been proposed. Given that existing benchmarks are suboptimal, as discussed in Section II-C, there is a need for a suitable benchmark that can be used. In Fig. 4, I propose a novel benchmark. The largest part of the benchmark, Fig. 4b, consists of a square trajectory. The main motivation behind this square trajectory is that any model can no longer ignore side movements. Another advantage is that each trajectory ends at the starting position. Thus, a value for the model drift can be computed, independent of the ground truth quality. The last advantage is that this square is less traffic-dense than the SCB trajectory, simplifying the camera feature extraction part of the algorithms.

The proposed benchmark also consists of a larger (600 m) square trajectory, Fig. 4c, that can be used as a zero-shot test case to check for overfitting on the specific small square. Lastly, I included a realistic urban trajectory of 3.9 kilometres, see Fig. 4d. The motivation for this last test is to discover possible improvement strategies to move to real-world applications in the future.

For the benchmark data, I connected an MPU-6050 to the handlebars of a bike. I used an Akamduman camera for high-resolution (1080p, full HD) video data and a Raspberry Pi camera for low-resolution (720p) video. I used a ZED-F9P GNSS module as ground truth. The IMU data, low-resolution camera data, and ground truth were hardware synchronised using the Raspberry Pi platform designed by G. S. Salustiano [21]. In total, I cycled the small square trajectory 120 times – of which 60 times clockwise and 60 times counterclockwise. I cycled the zero-shot test trajectory 3 times and the urban trajectory 1 time. The entire dataset is available at [22].

C. Training & Evaluation Procedure

I trained between 10 and 15 networks in parallel, using Jupyter Notebook environments on the cloud compute servers of the University of Twente. All used code is available at [18]. The models are set to train at most 50 epochs – early stopping enabled – with the RMSE as the evaluation parameter. After training, the performance on the test set is quickly evaluated, both on the RMSE, and by visual inspection. Promising algorithm candidates are put forward to the evaluation procedure.

I evaluated each promising algorithm twice – before and after post-processing – on the specified test set. I allowed three post-processing operations, in the following order:

1) **Filtering using a moving average filter.** The inputted IMU data is unfiltered, which causes a noisy prediction. I use a moving average (MA) filter to reduce high-frequency noise.
2) **Scaling.** The current models are unaware of any intrinsic camera parameters; hence, obtaining the correct scale is difficult. Additionally, the filtering operation will alter the scale. I allow a basic scaling operation – i.e. multiplying two constants c_x and c_y with the x and y output values – to correct for both problems.
3) **Shifting.** A low-pass filter causes delay. Furthermore, the starting location is not known by the model. The predicted x and y trajectories may be shifted by a constant in both horizontal and vertical directions.

The models will first be tested on a part of small square benchmark data (Fig. 4b) that was not used for training. Then, I will test all models on the zero-shot square trajectory as seen in Fig. 4c. Lastly, the models that generalise best are tested against the realistic urban trajectory in Fig. 4d. I do not expect the algorithms to perform well on this last test; the urban trajectory test data is not comparable to anything in the dataset used to train the models. An overview of the evaluation parameters can be seen in Section VIII-C, Table IV.

IV. Results

Based on manual inspection, of which the observations can be found in Section VIII-D, I selected seven algorithms for the evaluation procedure as described in Section III-C. Table IIIa and Table IIIb show the results of the selected neural network architectures for the standard test and the zero-shot trajectory, respectively. For the standard test case, Fig. 5a shows the best-performing model without interpolated data input, i.e. model 11. Fig. 5b shows Model 10i, the best-performing model with interpolated data input. Subsequently, for the zero-shot test case, the best performing models – models 9 and 7i – can be seen in Fig. 5c and Fig. 5d, respectively.

Since Models 9 and 7i generalise best based on the zero-shot test case, these models are tested against the 3.9 km urban trajectory in Fig. 4d. The results can be seen in Fig. 5e and Fig. 5f. For model 9, the RMSE of the unfiltered output is 1749 m, the RMSE of the filtered output is 940 m, and the drift error after filtering is 3003 m. For model 7i, the RMSE of the unfiltered output is 1746, the filtered is 1165 m, and the drift error is 3129 m.

Model	Length full test set	RMSE without post-processing	MA window size	Length best round	RMSE best round		Drift error best round		Turn accuracy metric	Total score
	in m	in m	in samples	in m	in m	in %	in m	in %		
7i	2901	20	1600	130	13	10.2	5	3.8	23	41
8	2882	20	1200	133	14	10.1	10	7.7	22	45
9	2837	18	1000	134	9	6.4	11	8.4	12	32
9i	2834	20	1000	133	9	6.5	7	5.3	16	31
10	2812	10	300	130	5	4.1	4	2.8	8	17
10i	2811	12	400	133	7	5.2	13	9.4	11	31
11	2764	9	100	134	4	3.3	1	0.4	8	13

(a) Standard test trajectory as seen in Fig. 4b.

Model	Length full test set	RMSE without processing	MA window size	Length best round	RMSE best round		Drift error best round		Total score
	in m	in m	in samples	in m	in m	in %	in m	in %	
7i	1818	156	7750	604	45	7.5	15	2.5	60
8	1818	152	7750	599	72	12.0	13	2.2	85
9	1818	154	10000	598	33	5.5	41	6.9	74
9i	1818	152	10000	602	37	6.1	97	16.1	134
10	1818	153	5000	598	96	16.1	100	16.7	196
10i	1818	155	9000	610	73	11.9	91	14.9	164
11	1818	152	7750	598	73	12.2	166	27.7	239

(b) Zero-shot test trajectory as seen in Fig. 4c.

TABLE III: The seven selected network architectures tested against different trajectories. Lower scores are better. The best scores for models without interpolation are marked in blue, and those for models with interpolation are marked in green. The overviews of the evaluation metric calculations and model topologies can be found in Section VIII-C and Section VIII-D, respectively.

V. Discussion

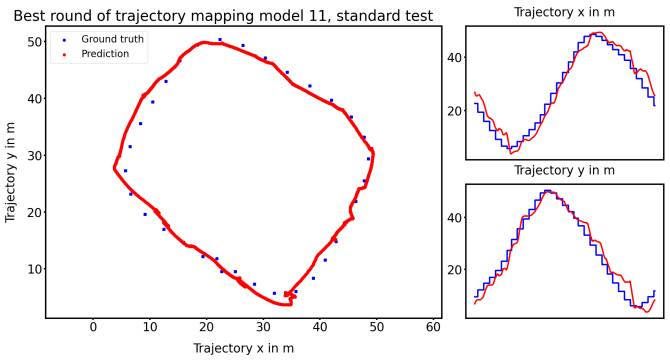
The first observation one can make is that the best models for the standard test and the zero-shot test are not the same. Model 11 maps an almost perfect trajectory on the standard test, as seen in Fig. 5a, but is the worst-performing model on the zero-shot test. This indicates overfitting on the specific small square trajectory. The main difference between the well-performing models on both tests is the model size. The best-performing model on the zero-shot is 7i, of which the first convolutional layer has 8 filters, whereas 10i and 11 – the best-performing models on the standard test – have 16 and 32 filters, respectively. The other layers of 10i and 11 are also larger than 7i and 9, as seen in Section VIII-D. Although larger models can perform better than smaller ones, they are also more prone to overfitting – which is observed here.

Nevertheless, the obtained results are useful. For the first time in literature, algorithms have been proposed to map non-linear bike trajectories, including turns and curves. Models 9 and 7i show the possibilities of generalising a model to work on unseen trajectories. Models 11 and 10i show practical maximums to performance on known trajectories. The drift error of model 11 during the standard test is on par with LARVIO [11]. Both the smaller and larger neural network configurations are worth further researching. The smaller models can be tested against different trajectories to discover their limitations. The larger models can be trained with more data, e.g. the SCB dataset, to reduce the possibility of overfitting.

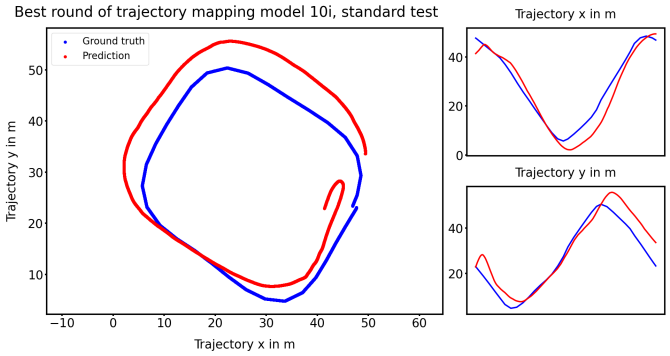
Currently, the tested algorithms perform poorly on real-world urban trajectories, as seen in Fig. 5e and Fig. 5f, which confirms the initial hypothesis. The primary reason for this poor performance is that the urban trajectory significantly differs from the training data: it consists of moving traffic, speed bumps, traffic lights, etc. The SCB dataset could again be of use here. Training a CNN-LSTM model on this more varied dataset could help make a more generalised model. An interesting observation is that model 7i tries to output a square in Fig. 5f, which it has been trained on. Model 9, Fig. 5e, generalises better, with parts of the outputted trajectory showing similarities to the ground truth. I hypothesise that the large anomalies on the x_{pred} and y_{pred} halfway through the trajectory are caused by the presence of other moving traffic participants.

I identified various limitations and flaws of the methods used. The first limitation is the accuracy of the ground truth. By comparing the camera images to the recorded GNSS data, I estimate that the approximate accuracy is $\pm$ 1 m. This accuracy could further drop in urban environments. Future work could explore integrating different ground truth systems, e.g. LIDAR, which is used in [16].

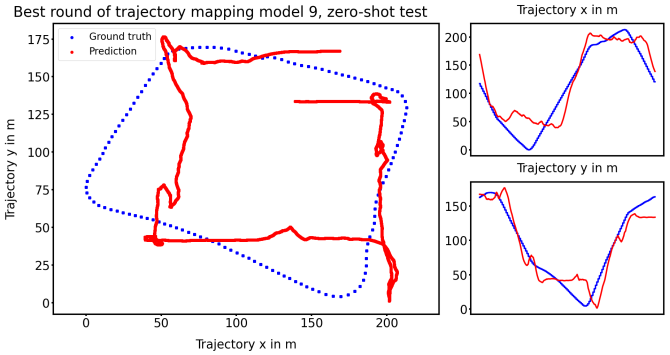
The second flaw is the synchronisation error between the different data sources. Although the IMU, low-resolution camera, and GNSS module are hardware synchronised, I used the data from the high-resolution camera as the source for the OpenCV odometry toolbox. The high-resolution videos yielded significantly better results than the low-resolution ones. Synchronisation was based on timestamps, accepting a synchronisation error of at most 1 s. I limited the influence of this error by reducing the number of different dataset runs. The entire dataset has been recorded over 6 continuous sessions,



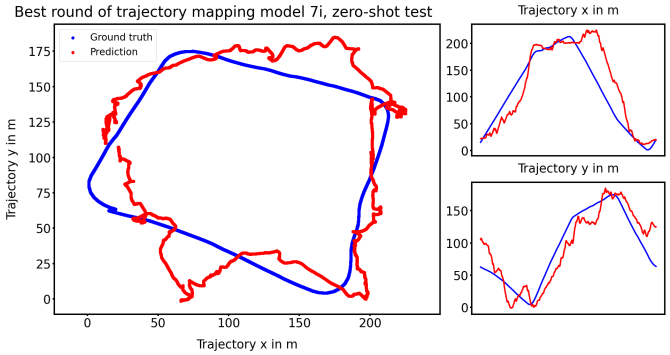
(a) Best round of model 11, tested against the standard test trajectory as shown in Fig. 4b.



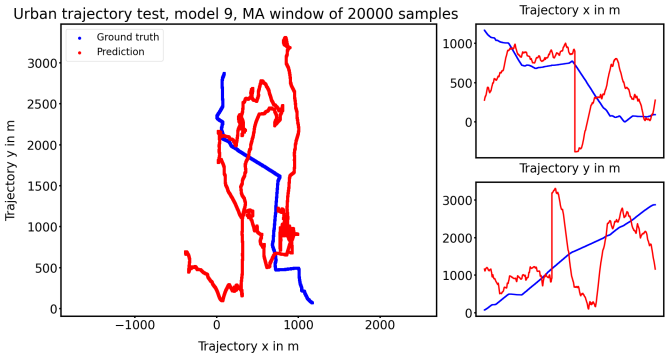
(b) Best round of model 10i, tested against the standard test trajectory as shown in Fig. 4b.



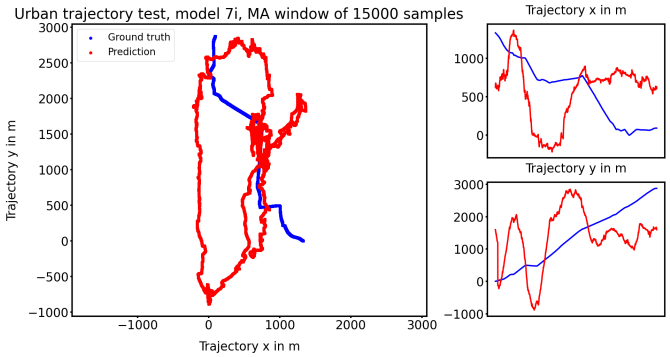
(c) Best round of model 9, tested against the zero-shot test case as shown in Fig. 4c.



(d) Best round of model 7i, tested against the zero-shot test case as shown in Fig. 4c.



(e) Model 9 tested against the urban trajectory as shown in Fig. 4d.



(f) Model 7i tested against the urban trajectory as shown in Fig. 4d.

Fig. 5: Comparison of the best-performing models on the three test trajectories. All models without interpolation can be seen on the left, and all models with interpolation on the right. The ground truth is blue, and the model output is red.

yielding at most 6 different synchronisation errors. In the future, the synchronisation error can be solved by including the high-resolution camera in the hardware synchronisation platform. The error can also be mitigated entirely by building a version of the model that uses only the low-resolution camera.

The third flaw is that – depending on the algorithm – significant post-processing operations might be required, as described in Section III-C. It is more efficient to filter the data prior to feeding it through the neural network because this reduces the number of calculations the model should perform. Future work could investigate the impact of different sampling rates and filter strategies.

The last flaw is the limited diversity of cyclists used for the novel dataset. I cycled all the recorded trajectories, potentially causing the models to be overfitted to my cycling patterns and behaviours. This flaw can be solved by adding data from more participants, e.g., by using the SCB dataset.

VI. Conclusion

This paper presents a pioneering approach to bicycle ego-trajectory mapping using CNN-LSTM models, demonstrating their ability to map turns and curves accurately. I also proposed a novel dataset that was used to train the current models and can be used as a benchmark for future systems.

All flaws of the currently proposed system – as discussed in Section V – can be solved by extending the dataset or improving the model. Additionally, one can use the proposed model to implement prediction functionality, e.g. by shifting the ground truth values of the dataset and retraining the model. An interesting study would be investigating the time-ahead prediction versus the model accuracy. Other studies could investigate the robustness of the model when sensor data is missing or unreliable. For example, consider a case where a raindrop falls on the camera sensor. This raindrop will be tracked as a feature but will not move. Hence, the odometry system might report that the bike is not moving, whereas the IMU data could show otherwise.

It is important to remember the bigger picture. In the future, the aim is to deploy trajectory mapping and prediction systems on bikes in real traffic. For this to work, the datasets used to train models should be extended with more varied data, e.g. longer routes, different weather conditions, different road surfaces, etc. The SCB dataset could be of use for this. Then, the system should be able to handle moving traffic, which is undesirable for getting an accurate camera state estimation. A solution proposed in [10] is using a Mahalanobis distance test. Another solution could be to deploy an object detection and recognition system that masks out often-moving objects, e.g. pedestrians, other bikes, and cars.

An important step in widespread adoption is developing an easy-to-use, easy-to-contribute framework. As stated in the introduction, most state-of-the-art algorithms require specific operating systems and many dependencies. In Section VIII-E, Fig. 6, I propose a Python-based processing pipeline to facilitate faster development. The main novelty of the proposed pipeline is that different processing steps can be executed independently, and all data is inputted and outputted in easy-to-access file formats, i.e. CSV files.

Lastly, a useful experiment would be to compare the performance of model-based trajectory mapping algorithms with data-driven ones. If a tool was made to convert the IMU and camera data into a usable Rosbag format, one could compare the proposed CNN-LSTM system with existing algorithms such as LARVIO.

Ultimately, this research should be seen as one of many puzzle pieces to the widespread adoption of bike mapping algorithms. The proposed algorithm and dataset can be used to develop other models, e.g., ones that are faster, more accurate, or better suited for embedded platforms. The code, model weights, and dataset are open source to endorse this further development.

VII. Acknowledgments

I want to thank G. S. Salustiano for helping me set up the data collection platform. I also thank J. K. Buursink and K. G. J. Martens for proofreading this document. Lastly, I am grateful to my committee members, Y. Huang, D. Yeleshetty, and K. Niu, for their advice during the process.

References

[1] Y. Huang, J. Du, Z. Yang, Z. Zhou, L. Zhang, and H. Chen, "A survey on trajectory-prediction methods for autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 7, no. 3, pp. 652–674, 2022. DOI: 10.1109/TIV.2022.3167103.

[2] J. Koornstra, *Predicting ego-bicycle trajectory: An lstm-based approach using camera and imu*, https://essay.utwente.nl/96462/7/Koornstra_BA_EEMCS.pdf, University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science, Enschede, The Netherlands, 2023.

[3] E. A. I. Pool, J. F. P. Kooij, and D. M. Gavrila, "Context-based cyclist path prediction using recurrent neural networks," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, 2019, pp. 824–830. DOI: 10.1109/IVS.2019.8813889.

[4] H. Guo, Q. Meng, D. Cao, H. Chen, J. Liu, and B. Shang, "Vehicle trajectory prediction method coupled with ego vehicle motion trend under dual attention mechanism," *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1–16, 2022. DOI: 10.1109/TIM.2022.3163136.

[5] R. Jiang, H. Xu, G. Gong, Y. Kuang, and Z. Liu, "Spatial-temporal attentive lstm for vehicle-trajectory prediction," *ISPRS International Journal of Geo-Information*, vol. 11, no. 7, 2022, ISSN: 2220-9964. DOI: 10.3390/ijgi11070354. [Online]. Available: https://www.mdpi.com/2220-9964/11/7/354.

[6] E. A. I. Pool, J. F. P. Kooij, and D. M. Gavrila, "Using road topology to improve cyclist path prediction," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 289–296. DOI: 10.1109/IVS.2017.7995734.

[7] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021. DOI: 10.1109/TRO.2021.3075644.

[8] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018. DOI: 10.1109/TRO.2018.2853729.

[9] University of Twente, Accell Group, TNO, Delft University of Technology, and Saxion University of Applied Science, *Smart connected bikes*, https://www.smartconnectedbikes.nl/, Accessed: 2024-06-03, 2022.

[10] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 3565–3572. DOI: 10.1109/ROBOT.2007.364024.

[11] X. Qiu, H. Zhang, and W. Fu, "Lightweight hybrid visual-inertial odometry with closed-form zero velocity update," *Chinese Journal of Aeronautics*, 2020.

[12] H. Z. Xiaochen Qiu and W. Fu, *Larvio: A lightweight, accurate and robust monocular visual inertial odometry*, https://github.com/PetWorm/LARVIO, Accessed: 2024-06-03, 2020.

[13] M. Bessi, *Trajectory prediction on an ego-bike with imu and camera using a multi-state constraint kalman filter*, https://essay.utwente.nl/96435/1/Bessi_BA_faculty.pdf, University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science, Enschede, The Netherlands, 2023.

[14] M. Burri, J. Nikolic, P. Gohl, *et al.*, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016. DOI: 10.1177/0278364915620033. eprint: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.full.pdf+html. [Online]. Available: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract.

[15] C. Li and S. L. Waslander, "Towards end-to-end learning of visual inertial odometry with an ekf," in *2020 17th Conference on Computer and Robot Vision (CRV)*, 2020, pp. 190–197. DOI: 10.1109/CRV50864.2020.00033.

[16] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[17] M. Boot, D. Yeleshetty, A. Jutte, and G. Kapousizis, *Smart connected bikes*, https://www.smartconnectedbikes.nl/Newsletter/Fieldtrials/, Accessed: 2024-06-04, 2024.

[18] S. Redeker, *Bike trajectory mapping*, 2024. [Online]. Available: https://github.com/StachRedeker/BikeTrajectoryMapping.

[19] S. Alsanwy, H. Asadi, M. R. C. Qazani, S. Mohamed, and S. Nahavandi, "A cnn-lstm based model to predict trajectory of human-driven vehicle," in *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2023, pp. 3097–3103. DOI: 10.1109/SMC53992.2023.10394243.

[20] S. Sharma, A. Singh, G. Sistu, M. Halton, and C. Eising, *Optimizing ego vehicle trajectory prediction: The graph enhancement approach*, 2024. arXiv: 2312.13104 [cs.CV].

[21] G. S. Salustiano, *Bike collector*, Accessed: 2024-06-24, 2024. [Online]. Available: https://github.com/guissalustiano/bike-collector.

[22] S. Redeker, *Bike trajectory mapping dataset*, Dataset, 2024. DOI: 10.5281/zenodo.12517266. [Online]. Available: https://doi.org/10.5281/zenodo.12517266.

VIII. Appendix

A. AI statement

During the preparation of this work, I used Grammarly Premium to check the document for language mistakes and improve fluency. Furthermore, I used ChatGPT 4o for implementing the OpenCV odometry toolbox in Python, generating code for plotting trajectory predictions, setting up sub-figure environments in LaTeX, and converting the tables in this paper from Excel to LaTeX. After using these services, I thoroughly reviewed and edited the content as needed, taking full responsibility for the final outcome.

B. Provided Code and Documentation

The following scripts and programs are the output of this research paper. They are provided under the MIT licence at [18].

1) **LocationMapper.** A Python script that takes coordinates and places these on a map.
2) **SyncSystem.** The first version of a system that could be used to automatically synchronise camera, IMU, and GPS data from the SCB dataset.
3) **MapAndVideo.** A Python script that takes location and camera data, and plots both on an interactive map.
4) **RosbagExtraction.** A tool that can be used to extract Rosbag output data from LARVIO.
5) **OpenCVFeatures.** A system that uses the odometry toolbox from OpenCV to find trackable features in a video.
6) **OpenCVOdometry.** A system that used the odometry toolbox from OpenCV to create a state estimate based on only camera data.
7) **ProposedFramework.** The first version of a Python processing pipeline which can be used to facilitate further development and research.
8) **CNN-LSTM.** This contains all scripts related to the development of the CNN-LSTM mapping algorithm, including but not limited to:
    a) Code to synchronise the input data.
    b) Code to convert GPS coordinates to relative coordinates.
    c) Code to train the neural networks.
    d) The weights and biases of the trained neural networks.
    e) Code to evaluate the performance and compute the relevant metrics.

Additionally, the novel dataset is made public at [22].

C. Evaluation Metrics

Test case	Evaluation parameter	Post-processing operations	Explanation
Standard test (small square trajectory)	RMSE full test set	None.	Gives an indication of the out-of-the-box performance.
	RMSE, best single round	At least three consecutive rounds are selected and post-processed.	The best single round is selected. The RMSE is calculated for that single round.
	Drift error, best single round		Since every round starts and ends in the same place, one can compute the error caused by drift. This excludes the influence of any ground truth errors.
	Turn accuracy metric, (best) single round		Identical to the RMSE, however, errors during a turn are multiplied by 2. This penalises models that do not work properly on turns and curves.
	Total score		The sum of the RMSE for the best single round, the drift error, and the turn accuracy metric.
Zero-shot test case (larger square trajectory)	RMSE full test set	None.	Gives an indication of the out-of-the-box performance.
	RMSE, best single round	All post-processing operations, on all rounds.	The best single round is selected. The RMSE is calculated for that single round.
	Drift error, best single round		The drift error of the best single round.
	Total score		The sum of the RMSE for the best single round and the drift error.
Urban traffic test	RMSE full test set	None.	Gives an indication of the out-of-the-box performance.
	RMSE, best single round	All post-processing operations.	
	Drift error end position		

TABLE IV: Evaluation parameters for the different test cases.

D. Analysed CNN-LSTM Configurations

Model	Architecture	Training parameters				Data input	Observations
		TR data	VAL data	TEST data	Window length		
2	Conv1D(16) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(25, return sequences) - BatchNormalization - LSTM(25) - Dropout(0.3) - Dense(25) - Dense	1	0,5*2	0,5*2	200	repetition	Overfitted, but showed signs of periodicity on the test data.
3	Conv1D(32) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(50, return sequences) - BatchNormalization - LSTM(50) - Dropout(0.3) - Dense(50) - Dense	1	0,5*2	0,5*2	400	repetition	Overfitted.
4	Conv1D(16) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(25, return sequences) - BatchNormalization - LSTM(25) - Dropout(0.3) - Dense(25) - Dense	1	0,5*2	0,5*2	400	repetition	Did not move away from the starting location.
5	Conv1D(16) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(25, return sequences) - BatchNormalization - LSTM(25) - Dropout(0.3) - Dense(25) - Dense	1,2,4	0,5*5	0,5*5	200	repetition	Shows some signs of periodicity, is able to successfully predict one corner location of the test trajectory (most likely due to overfitting).
5i	Conv1D(16) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(25, return sequences) - BatchNormalization - LSTM(25) - Dropout(0.3) - Dense(25) - Dense	1,2,4	0,5*5	0,5*5	200	interpolate	Shows some signs of periodicity, is able to successfully predict one corner location of the test trajectory (most likely due to overfitting).
6	Conv1D(8) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(12, return sequences) - BatchNormalization - LSTM(12) - Dropout(0.3) - Dense(12) - Dense	1,2,4	0,5*5	0,5*5	200	repetition	Does not converge.
6i	Conv1D(8) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(12, return sequences) - BatchNormalization - LSTM(12) - Dropout(0.3) - Dense(12) - Dense	1,2,4	0,5*5	0,5*5	200	interpolate	Does not converge.
7	Conv1D(8) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(12, return sequences) - BatchNormalization - LSTM(12) - Dropout(0.3) - Dense(12) - Dense	1,2,4	0,5*5	0,5*5	100	repetition	Only converges for the y trajectory, and not for the x trajectory.
7i	Conv1D(8) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(12, return sequences) - BatchNormalization - LSTM(12) - Dropout(0.3) - Dense(12) - Dense	1,2,4	0,5*5	0,5*5	100	interpolate	Results after heavy post-processing seem promising.
8	Conv1D(8) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(12, return sequences) - BatchNormalization - LSTM(12) - Dropout(0.3) - Dense(12) - Dense	1,2,4	0,5*5	0,5*5	800	repetition	Slightly overfitted, good periodicity, proper idea of scale even without post-processing.

TABLE V: Analysed CNN-LSTM configurations, part I. Promising candidates are marked in green.

Model	Training parameters						Observations
	Architecture	TR data	VAL data	TEST data	Window length	Data input	
8i	Conv1D(8) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(12, return sequences) - BatchNormalization - LSTM(12) - Dropout(0.3) - Dense(12) - Dense	1,2,4	0,5*5	0,5*5	800	interpolate	By visual inspection, good performance on the y trajectory, poor performance on the x trajectory. Good periodicity.
9	Conv1D(8) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(12, return sequences) - BatchNormalization - LSTM(12) - Dropout(0.3) - Dense(12) - Dense	1,2,5	0,5*4	0,5*4	800	repetition	Slightly overfitted, good performance on the y trajectory, medium performance on the x trajectory. Good periodicity.
9i	Conv1D(8) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(12, return sequences) - BatchNormalization - LSTM(12) - Dropout(0.3) - Dense(12) - Dense	1,2,5	0,5*4	0,5*4	800	interpolate	Strong x and y performance, especially after post-processing.
10	Conv1D(16) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(25, return sequences) - BatchNormalization - LSTM(25) - Dropout(0.3) - Dense(25) - Dense	1,2,5	0,5*4	0,5*4	1600	repetition	Strong x and y performance, also without post-processing. Might be overfitted to the specific trajectory, further testing should show this.
10i	Conv1D(16) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(25, return sequences) - BatchNormalization - LSTM(25) - Dropout(0.3) - Dense(25) - Dense	1,2,5	0,5*4	0,5*4	1600	interpolate	Strong x and y performance, also without post-processing. Might be overfitted to the specific trajectory, further testing should show this.
11	Conv1D(32) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(48, return sequences) - BatchNormalization - LSTM(48) - Dropout(0.3) - Dense(24) - Dense	1,2,5	0,5*4	0,5*4	3200	repetition	Excellent performance on both x and y, also without post-processing. Might be overfitted to the specific trajectory.
11i	Conv1D(32) - BatchNormalization - MaxPooling1D(2) - Dropout(0.3) - LSTM(48, return sequences) - BatchNormalization - LSTM(48) - Dropout(0.3) - Dense(24) - Dense	1,2,5	0,5*4	0,5*4	3200	interpolate	Excellent performance on both x and y, also without post-processing. Might be overfitted to the specific trajectory. Is not able to run all tests.
12	Sequential - LSTM(100, return sequences) - Dropout(0.2) - LSTM (100) - Dense(50) - Dense(2)	1,2,5	0,5*4	0,5*4	400	repetition	Good periodicity, proper scaling even without post-processing. Medium prediction results.
12i	Sequential - LSTM(100, return sequences) - Dropout(0.2) - LSTM (100) - Dense(50) - Dense(2)	1,2,5	0,5*4	0,5*4	400	interpolate	Good periodicity, proper scaling even without post-processing. Medium prediction results.
13	Sequential - LSTM(100, return sequences) - Dropout(0.2) - LSTM (100) - Dense(50) - Dense(2)	1,2,5	0,5*4	0,5*4	200	repetition	Poor prediction results, even after post-processing. Signs of periodicity.
13i	Sequential - LSTM(100, return sequences) - Dropout(0.2) - LSTM (100) - Dense(50) - Dense(2)	1,2,5	0,5*4	0,5*4	200	interpolate	Poor prediction results, even after post-processing. Signs of periodicity.
14	Sequential - LSTM(100, return sequences) - Dropout(0.2) - LSTM (100) - Dense(50) - Dense(2)	1,2,5	0,5*4	0,5*4	100	repetition	Poor results, even after post-processing. Little signs of periodicity. Proper idea of scale.
14i	Sequential - LSTM(100, return sequences) - Dropout(0.2) - LSTM (100) - Dense(50) - Dense(2)	1,2,5	0,5*4	0,5*4	100	interpolate	Poor results, even after post-processing. Little signs of periodicity. Proper idea of scale.
15	Sequential - LSTM(50, return sequences) - Dropout(0.2) - LSTM (50) - Dense(25) - Dense(2)	1,2,5	0,5*4	0,5*4	800	repetition	DNF - not enough memory available to train.
15i	Sequential - LSTM(50, return sequences) - Dropout(0.2) - LSTM (50) - Dense(25) - Dense(2)	1,2,5	0,5*4	0,5*4	800	interpolate	DNF - not enough memory available to train.

TABLE VI: Analysed CNN-LSTM configurations, part II. Promising candidates are marked in green.

E. Proposed Python Processing Pipeline

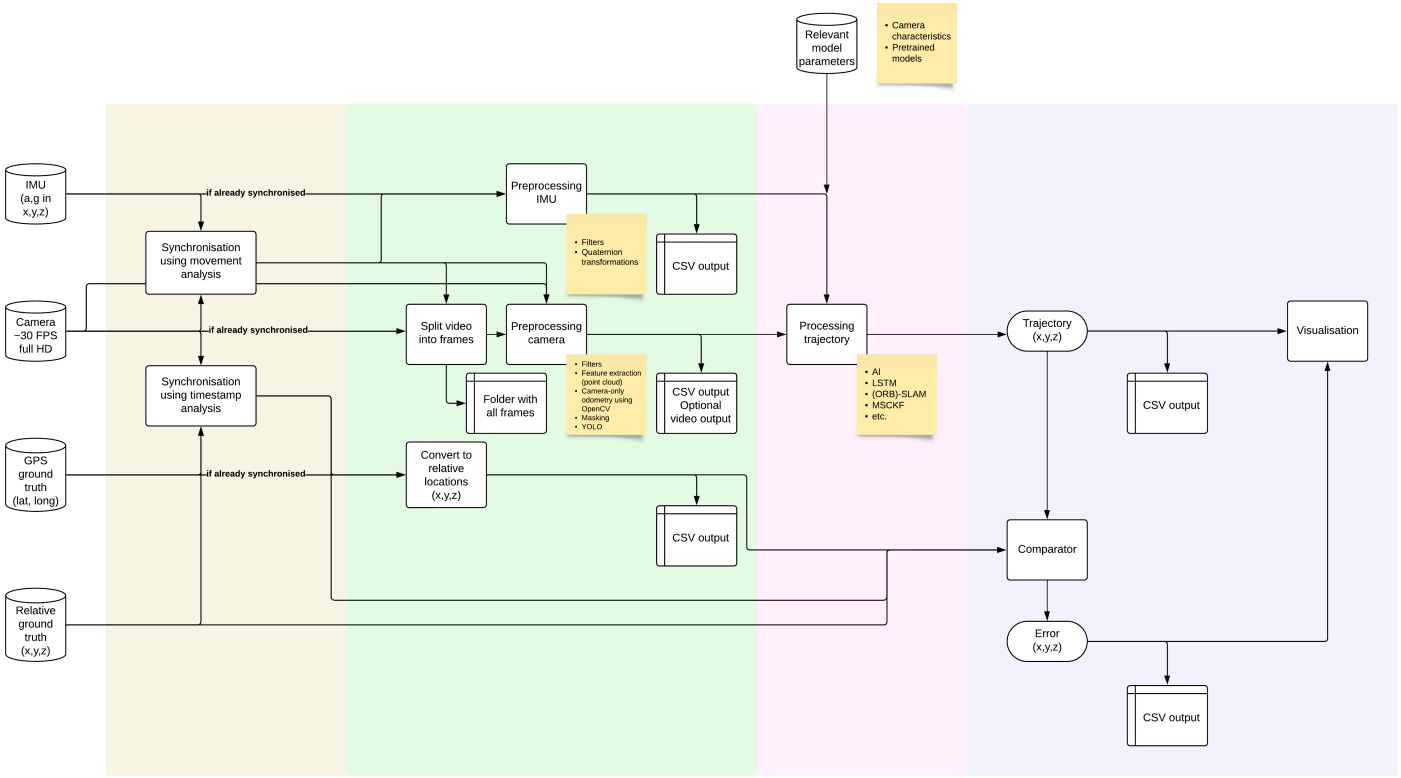


Fig. 6: Low-level block diagram of the proposed Python processing pipeline for using the SCB dataset. The main novelty is that every step produces intermediate CSV output files, facilitating platform-independent development and object-oriented programming strategies, such as abstraction and encapsulation.