UNIVERSITY OF CALIFORNIA

Los Angeles

Robust and Efficient Neural Inertial Localization and Complex Activity Recognition

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Electrical and Computer Engineering

by

Swapnil Sayan Saha

2021

ABSTRACT OF THE THESIS

Robust and Efficient Neural Inertial Localization
and Complex Activity Recognition

by

Swapnil Sayan Saha
Master of Science in Electrical and Computer Engineering
University of California, Los Angeles, 2021
Professor Mani B. Srivastava, Chair

Inertial complex activity recognition and neural inertial navigation are challenging due to missing samples, misaligned data timestamps across sensor channels, variations in sampling rates and high model deployment costs. In this thesis, we introduce a robust training pipeline for complex activity detection that handles sampling rate variability, missing data, and misaligned data timestamps using intelligent data augmentation techniques. Specifically, we use controlled jitter in window length and add artificial misalignments in data timestamps between sensors, along with masking representations of missing data. In addition, we exploit end-to-end sequential learning, $\alpha - \beta$ filters, Madgwick filters, hardware and quantization-aware Bayesian neural architecture search and a temporal convolutional neural network backbone to form the basis of scalable, real-time and sub-meter GPS-free inertial localization on wide spectrum of target resource-constrained hardware. We also provide a compact, ultra-low-power, environmentally resilient and modular sensor tag configuration that pushes the state-of-the-art in inertial odometry hardware. On average, the network found via our efficient pipeline provided $3\times$ peak activation and $6\times$ memory savings over the state-of-the-art neural inertial algorithms and taking at most 24 hours to train and search pareto-optimal models in the backbone search space. Moreover, we evaluate the complex activity pipeline on state-of-the-art complex activity recognition dataset, achieving test accuracies of 88% and 72% respectively for coarse and granular-activity classification while ranking 3rd in the 2020 Cooking Activity Recognition Challenge out of 78 submissions.

The thesis of Swapnil Sayan Saha is approved.

Jonathan Chau-Yan Kao

Xiang Chen

Mani B. Srivastava, Committee Chair

University of California, Los Angeles

2021

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I express my utmost gratitude to Prof. Mani B. Srivastava for being my M.S. thesis advisor and a mentor in both professional and academic settings. I would also like to thank Prof. Jonathan Chau-Yan Kao and Prof. Xiang Chen for providing their valuable and insightful feedback and constructive comments on the thesis.

I am grateful to Sandeep Singh Sandha (NESL, UCLA) and Prof. Luis Garcia (ISI, USC) for being core colleagues in my research journey, along with the rest of the members in the Networked and Embedded Systems Lab (NESL). I am also thankful to my soulmate from MIT for keeping me patient and supporting me at all times.

Special thanks to Urvi Shah, Junha Park, Josh Geronimo (undergraduate students at UCLA) and Suyog Vyawahare (consultant at Sincos Automation Technologies) for participating in the hardware design in this thesis. Sincere gratitude to Vivek Jain (XCOM Labs) and Siyou Pei (HiLab, UCLA) for taking part in the earable data collection in this thesis.

Parts of Chapter 2, 3, 5 and 6 are collectively a version of: Saha S.S., Sandha S.S., Srivastava M. (2020) "Deep Convolutional Bidirectional LSTM for Complex Activity Recognition with Missing Data." In: Ahad M.A.R., Lago P., Inoue S. (eds) *Human Activity Recognition Challenge. Smart Innovation, Systems and Technologies*, vol 199. Springer, Singapore. https://doi.org/10.1007/978-981-15-8269-1_4

Lastly I would like to acknowledge the funding agencies sponsoring the work, namely the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, the Army Research Laboratory (ARL) under Cooperative Agreement W911NF-17-2-0196 and the King Abdullah University of Science and Technology (KAUST) through its Sensor Innovation research program.

CHAPTER 1

Introduction

An inertial measurement unit (IMU) reports mass-specific force and angular velocity of an object [76], often complemented by a magnetometer for reporting heading in the North-East-Down (NED) reference frame. Thanks to developments in micro-electro-mechanical system (MEMS) technology, inertial sensors are omnipresent and ubiquitous in modern day electronic systems as shown in Figure 1.1. Application spectrum of inertial sensors include autonomous vehicle attitude estimation and infrastructure-independent positioning on land, underwater, air and space [87][181] [87] for both military and civilian applications, pedestrian dead-reckoning (PDR), transportation mode, fitness and well-being, human gait and activity analysis [84][85] [182] [183] [184] [185] using smartphones and wearables, pose-estimation of smartphones and virtual-reality headsets coupled with full-body motion capture in haptics, gaming and augmented reality settings [181] [186] [187], robot joint state estimation [188] and multimodal sensing in smart homes [189] [190]. The market for inertial sensors was valued at USD 18.97 billion in 2019, with an expected value of 27.23 billion by 2025 at 6.49% CAGR [191].

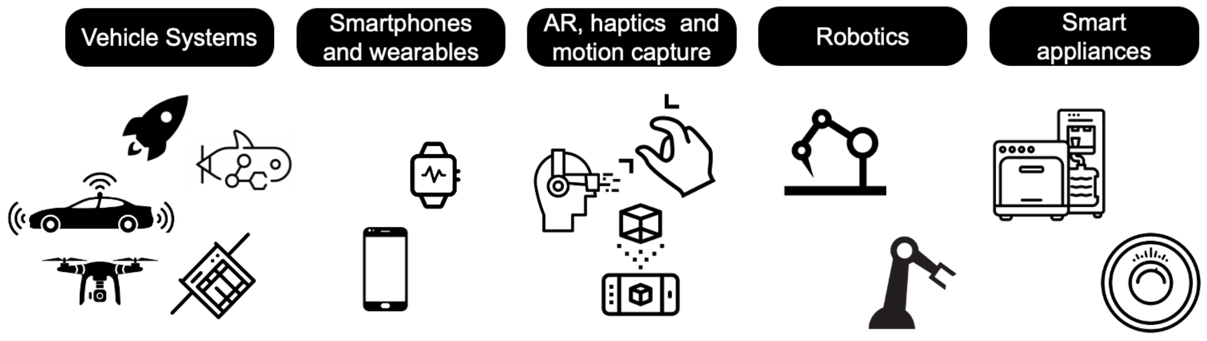


Figure 1.1: Inertial sensors encompass a broad domain spectrum with widespread ubiquity.

Two of the most prolific and widely-studied applications of inertial sensors include complex human activity recognition (HAR) and inertial odometry in GPS-denied environments. HAR is defined as the process of correlating a sequence of granular human action primitives with similar data from a datastore [192]. Complex events are anything that can be decomposed into a sequence of simpler events [3]. On the other hand, inertial odometry (often referred to as dead-reckoning) is the fusion of on-board inertial sensors for indirect infrastructure-free estimation of an object's position (and attitude) for navigation [193][194]. Among the current odometry and activity recognition techniques, inertial sensors provide a small footprint, low cost, low access delay, low power, high sampling rate and high resolution solution for complex event processing and localization [85][149].

1.1 Challenges in Complex Activity Detection and Inertial Odometry

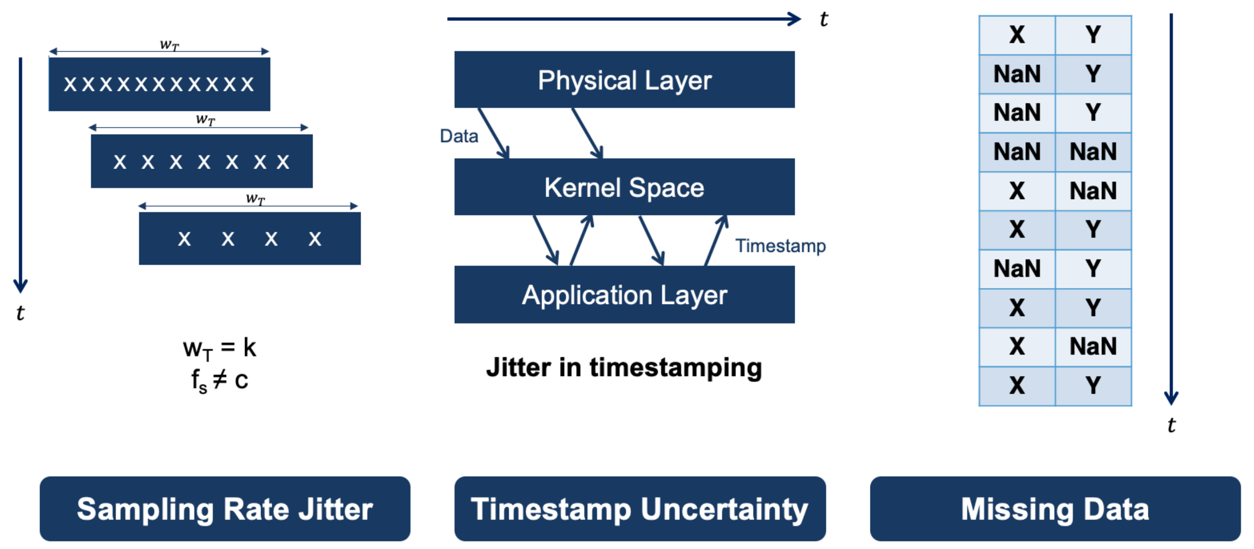


Figure 1.2: Challenges in deploying inertial complex activity recognition systems in the wild.

There are several challenges in deploying inertial complex activity detection system in the wild, illustrated in Figure 1.2. Ubiquitous inertial systems are known to have sampling rate jitter, cross-channel timestamp misalignment and occasional sample loss (otherwise known as missing data) [1][132][133], which can hurt the performance of vanilla complex event processing pipelines [6][7][15][132]. This can lead to disastrous consequences in critical settings, particularly in healthcare monitoring. For example a nurse may need to wash her hands before administering an injection to a patient. The temporal dynamics in complex activities such as nurse-care recognition have been shown to be severely hampered by loss of information caused by sensing and timing uncertainties in inertial streams [195][196]. Furthermore, detecting complex events at various space and time scales requires injection of conditional logic and hierarchy in the detection framework, as some granular events may be dependent on the presence of macro events [3][15]. Without appropriate pre-processing and architectural encodings, even artificial intelligence (AI) based complex event processors suffer from accuracy loss during test time [15][154].

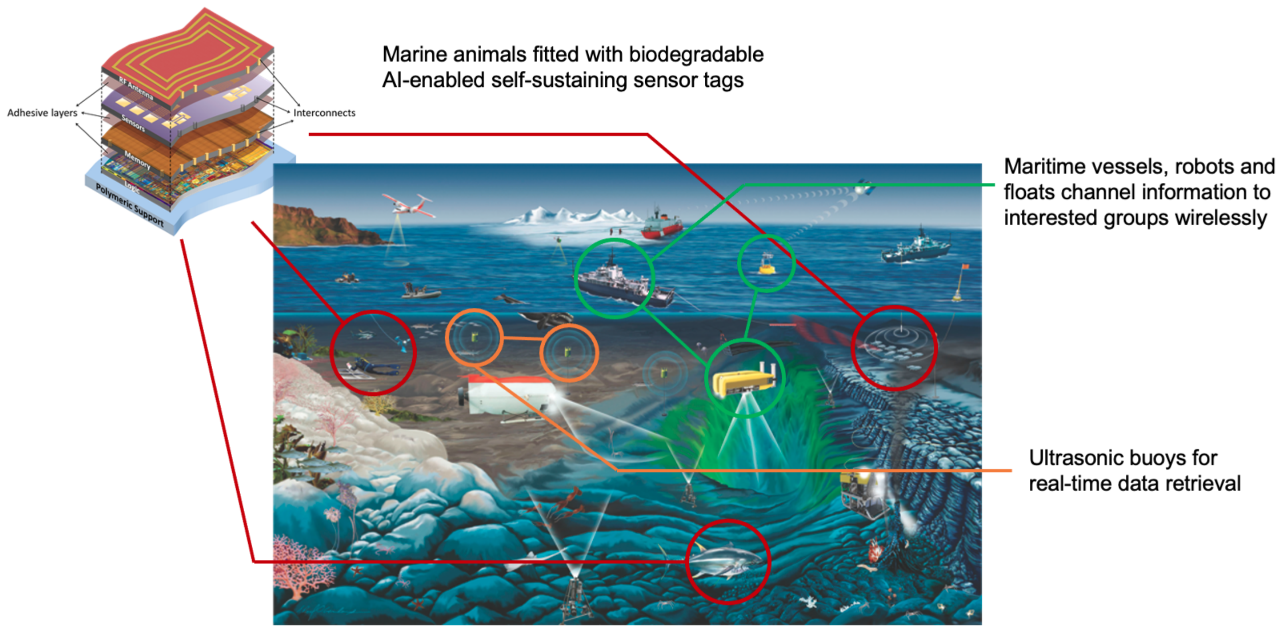

Marine animals fitted with biodegradable AI-enabled self-sustaining sensor tags
Maritime vessels, robots and floats channel information to interested groups wirelessly
Ultrasonic buoys for real-time data retrieval
Adhesive layers
Interconnects
RF Antenna
Sensors
Memory
Polymeric Support

Figure 1.3: Example application requiring real-time neural inertial odometry [197].

In addition, MEMS inertial sensors suffer from time-varying drift and noise [73][82][120][76]. Dead-reckoning involves cumulatively summing the estimated position at current timestamp relative to the position at previous timestep in conjunction with heading information from attitude estimation algorithms. While inertial sensors are generally available continuously without interruption to the object interested in localizing itself, the error bound in the estimated position explodes cubically with each timestep [78][75]. Classical techniques to constrain errors such as Bayesian filters, man-made system models and heuristics are not robust to domain variance and ambient dynamics. Although these techniques are computationally inexpensive, they are accurate only under the system model that they have been designed for, with no "one-size-fits-all" scenario for system model and designed heuristics due to non-optimal parametrization via linear approximation of the real-world [82][129]. On the other hand, AI-based inertial odometry frameworks have been shown to outperform model-based techniques under domain variance and stochasticity [82][102][103][120][131][117][125][129][123], but suffer from high training and model optimization costs, constraining their application in the offline world or on smartphones, cloud and workstations [131]. Existing off-the-shelf hardware-agnostic neural optimization techniques alone cannot drastically reduce model size without incurring performance loss, with most of the tools geared towards smartphones and image processing frameworks [24]. A broad spectrum of localization applications in GPS or network-denied environments mandate accurate yet lightweight and real-time deployment of neural-inertial models where smartphone or cloud-based evaluation are unusable. These include:

- Terrestrial and marine search and rescue missions [198][199].
- Underwater sensor networks, oceanic biodiversity and marine health tracking [72][200][197]. An example is shown in Figure 1.3.
- Wildlife monitoring [201][91].
- Deep-space small satellite localization [202].
- Coalition situational understanding [203][204].
- Patient tracking and healthcare monitoring [205].
- Localizing micro unmanned vehicles and robots [102][123][124][206].

For the above applications, it is not possible to use conventional infrastructure-dependent localization systems such as GPS or radio telemetry, as such systems are subject to both spatial and temporal limitations due to signal obfuscation and coverage [207][85][208], location error [209][210], access delays with sporadic accessibility [176][90] and hardware-software challenges for ubiquitous edge localization [171][210][211][85][74]. Non-inertial odometry techniques such as visual, LIDAR and RADAR egomotion suffers from sub-optimal and understated trajectory projections during local costmap updates. Real-world is stochastic, unstructured and complex, with changes in ambient and environmental conditions (such as lighting, weather, obfuscations and path loss) affecting non-inertial odometry sensors [212]. Furthermore, non-inertial odometry sensor algorithms are domain and platform specific, generalizing poorly across eclectic use-cases. In addition, they may not be suitable for integration in ubiquitous localization platforms given usability, power, compute, weight and space requirements. The shortcomings are summarized in Figure 1.4.

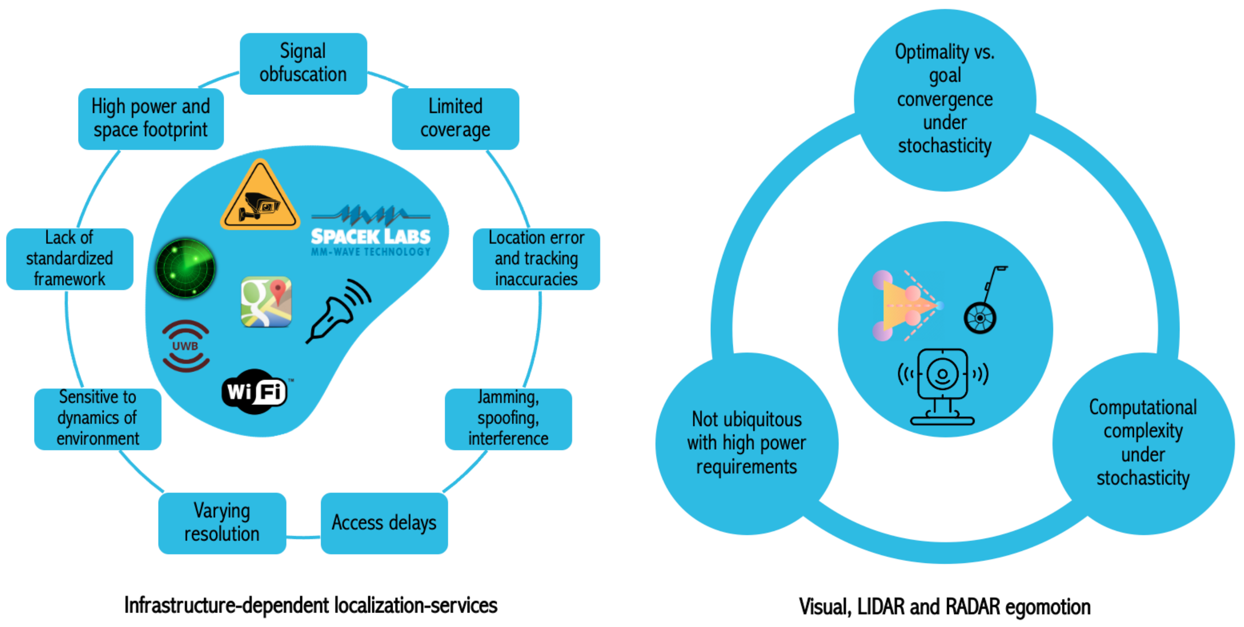

Signal obfuscation
Limited coverage
High power and space footprint
Location error and tracking inaccuracies
Lack of standardized framework
Jamming, spoofing, interference
Sensitive to dynamics of environment
Varying resolution
Access delays
SPACEK LABS
MM-WAVE TECHNOLOGY
UWB
Wi Fi
Infrastructure-dependent localization-services
Optimality vs. goal convergence under stochasticity
Not ubiquitous with high power requirements
Computational complexity under stochasticity
Visual, LIDAR and RADAR egomotion

Figure 1.4: The shortcomings in conventional localization techniques.

1.2 Contributions

In this thesis, we introduce techniques to handle runtime sensing uncertainties in inertial complex activity detection systems, and also provide a a framework for deploying real-time neural inertial models on resource-constrained (popularly known as TinyML) hardware for diverse applications. To handle the challenges of missing and misaligned sensor samples and variable sampling rates of multimodal data, we augment additional sensor channels in the training pipeline through masking representations of missing samples. Additionally, we perform data augmentation through artificial misalignments and jitter during window creation, effectively creating time-aware deep neural architectures. Our neural inertial navigation framework exploits end-to-end sequential learning, hardware and quantization-aware Bayesian neural architecture search (NAS) and a temporal convolutional neural network (TCN) backbone to form the basis of scalable sub-meter inertial localization. Our algorithms run on a wide spectrum of target TinyML hardware without requiring GPS or human-engineered system models. Our main contributions are follows:

- We provide a comprehensive survey on the recent advances in uncertainty aware complex event processors, inertial odometry and efficient deep-learning (DL), outlining their strengths and weaknesses (Chapter 2).
- We propose using a mixture of mask metadata channels and pop-ahead samples to quantify missing data. To handle sampling rate jitter and timestamp misalignment, we apply controlled artificial time shifts to the training data and timing jitter in window length (Chapter 3).
- We devise a hierarchical deep recurrent-convolutional architecture that can perform complex activity processing at varying spatial and temporal granularities, infused with conditional logic (Chapter 3).
- To the best of our knowledge, our proposed efficient neural inertial navigation framework is the first work that explores real-time AI-based inertial navigation for resource-constrained hardware (Chapter 4).

- We devise a systematic framework for deploying data-driven inertial odometry on a wide vareity of target hardware. We formulate a fast and parallelizable hardware-in-the-loop (HIL) quantization-aware Bayesian NAS operating on a TCN backbone to yield sub-meter neural-inertial architectures regularized by device constraints (Chapter 4).

- We formulate lightweight and computationally tractable altitude/depth and orientation filters (Chapter 4).

- We provide a compact, ultra-low-power, environmentally resilient and modular sensor tag configuration that pushes the state-of-the-art in inertial odometry hardware (Chapter 5).

- We describe a large-scale inertial dataset that we collected, called the *NESL Earable Mobility Corpus*, for benchmarking orientation and head-pose estimation filters in context of spatial audio (Chapter 5).

Figure 1.5 and Figure 1.6 provides an overview of our designed "robust and efficient" neural inertial localization and complex activity detection frameworks.

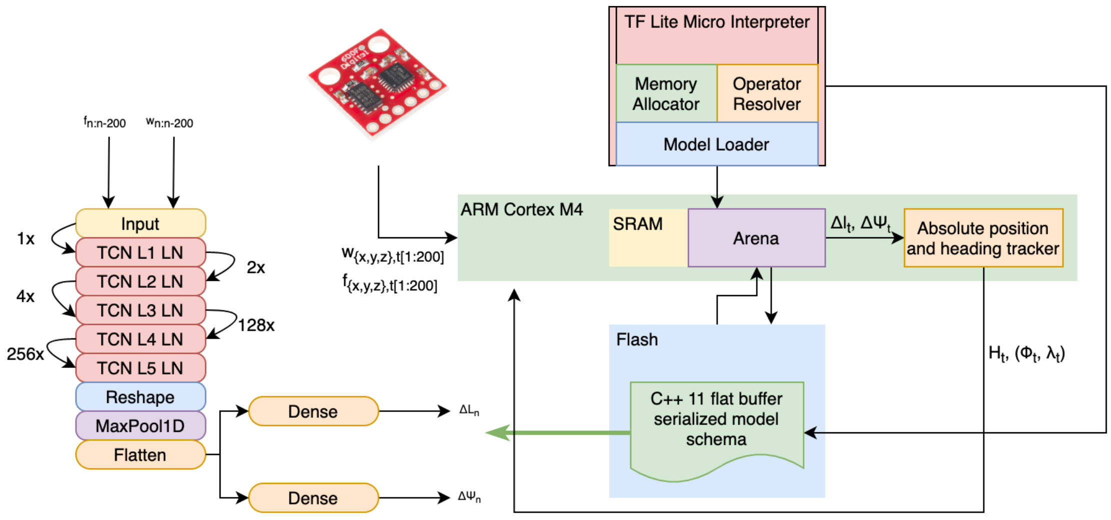
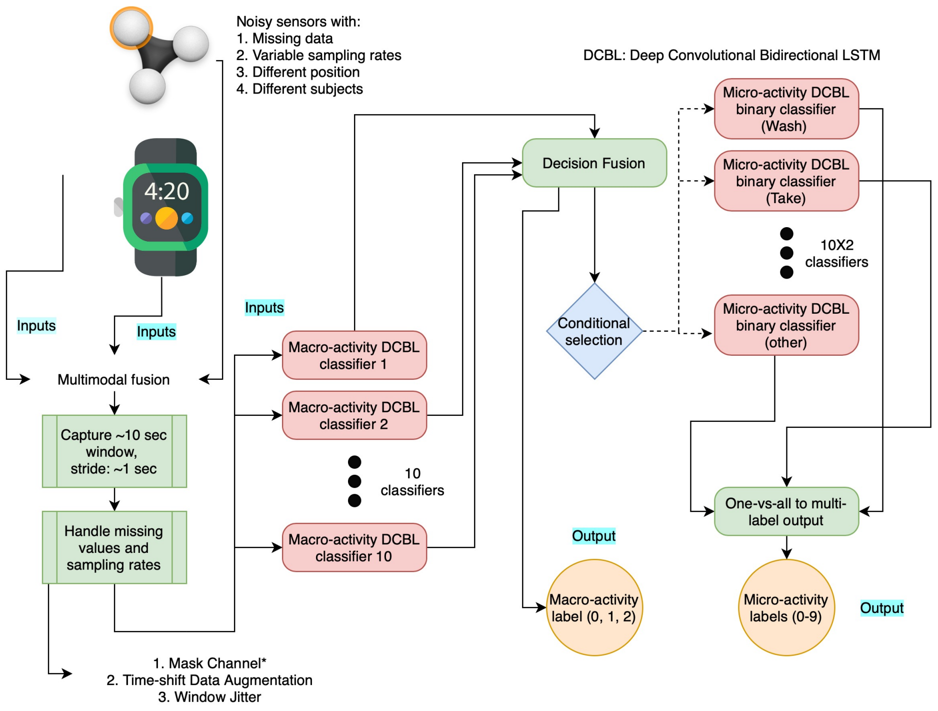


Figure 1.5: Overview of efficient neural inertial navigation.



Noisy sensors with:
1. Missing data
2. Variable sampling rates
3. Different position
4. Different subjects
DCBL: Deep Convolutional Bidirectional LSTM
4:20
Inputs
Inputs
Inputs
Multimodal fusion
Capture ~10 sec window, stride: ~1 sec
Handle missing values and sampling rates
1. Mask Channel*
2. Time-shift Data Augmentation
3. Window Jitter
Macro-activity DCBL classifier 1
Macro-activity DCBL classifier 2
10 classifiers
Macro-activity DCBL classifier 10
Decision Fusion
Conditional selection
Output
Macro-activity label (0, 1, 2)
Micro-activity DCBL binary classifier (Wash)
Micro-activity DCBL binary classifier (Take)
10X2 classifiers
Micro-activity DCBL binary classifier (other)
One-vs-all to multi-label output
Micro-activity labels (0-9)
Output

Figure 1.6: Overview of robust complex activity detection (specialized to operate on *Cooking Activity Dataset with Micro and Macro Activities* [155]).

CHAPTER 2

Background and Related Work

A prolific set of scholarly manuscripts and books exist addressing the issues of handling sensing uncertainties in DL pipelines, inertial localization and efficient DL (also known as TinyML). Common techniques to handle runtime uncertainties include feature extraction, matrix factorization and generative models [15]. Inertial localization can be categorized into model-based systems using system models, Bayesian filters and heuristics, or learning-enabled systems exploiting recent advances in DL [82]. Software advances in TinyML include the use of pruning, quantization and model compression, lightweight neural blocks, hardware-aware NAS and the rise of commercial off-the-shelf (COTS) tools [17][32]. In this chapter, we provide a comprehensive survey of the strengths and shortcomings of existing techniques in inertial localization and complex activity detection.

2.1 Handling Sensing Uncertainties in Temporal Streams

Several techniques have been proposed in the literature to handle missing data in temporal streams. Hossain et al. [1][2] proposed extracting handcrafted spatial features to diminish individual effects of sample points, using traditional machine learning (ML) algorithms for activity recognition [149][150]. However, the constituent primitives of complex events might generate features that evolve with time [3], requiring careful a trade-off between differentiability and generalizability during feature selection [4], ultimately leading to poor deployment performance for traditional classifiers compared to DL approaches [5]. Feature extraction leads to loss in high-dimensional spatial and temporal context for temporal sequences, while traditional interpolation techniques require implicit modelling assumptions

[15]. This is evident in [6], where the authors propose using masking and time interval channels as representations of missing patterns to improve the prediction performance of deep recurrent architectures. Their evaluations illustrate that DL architectures offer better generalizability in test cases over traditional interpolation (e.g., MICE, MissForest, matrix completion, spline, k-nearest neighbors (kNN), forward imputation and statistical imputation) and classification techniques (e.g., logistic regression (LR), support vector machines (SVM) and random forests (RF)) for temporal sequences. Yoon et al. [7] proposed a multi-directional recurrent neural network for missing data imputation that exploits correlation within and across data streams and with conclusions similar to [6], achieving 35-50% improvement in error metric over existing approaches. An alternative approach for handling sensing uncertainties in multi-modal temporal streams involves intelligently "transferring knowledge" from one sensor to the missing sensor modality via temporal/spatial correlation across sensors or mapping to shared latent space [8][9][10][11]. Recent endeavors include the use of generative adversarial models (GAN) to impute the missing samples synthetically, followed by training [12][13][14]. However, all of the existing methods fail to handle sampling rate invariability and timestamp uncertainty in fast temporal streams with missing data [15].

2.2 Model-Based Approaches for Autonomous Inertial Odometry

Under ideal geomagnetic conditions void of magnetic disturbances, non-uniform magnetic field or sensor noise and for slow movements, the heading H can be estimated directly from magnetometer, typically deployed for marine animal dead-reckoning (DR) [71][72]:

$$H = \arctan\left(\frac{m_{y,t}}{-m_{x,t}}\right) \cdot \frac{180}{\pi} \tag{2.1}$$

The altitude is typically obtained from barometric pressure sensors, while the latitude (ϕ) and longitude (λ) can be hypothetically obtained via double integration of accelerometer readings (f_x, f_y, f_z) under empirical accelerometer vector sum threshold α [72] to reject noise:

$$s_t = \begin{cases} \beta\sqrt{f_{x,t}^2 + f_{y,t}^2 + f_{z,t}^2} + \gamma, & \sqrt{f_{x,t}^2 + f_{y,t}^2 + f_{z,t}^2} > \alpha \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

$$\phi_t = \arcsin\left(\sin\phi_{t-1}\cdot\cos\frac{s_t}{R_E} + \cos\phi_{t-1}\cdot\sin\frac{s_t}{R_E}\cdot\cos H\right),\quad R_E = 6.371\times10^6\text{m} \tag{2.3}$$

$$\lambda_t = \lambda_{t-1} + \arctan 2\left(\sin H\cdot\sin\frac{s_t}{R_E}\cdot cos\phi_{t-1}, \cos\frac{s_t}{R_E} - \sin\phi_{t-1}\cdot\sin\phi_t\right) \tag{2.4}$$

While magnetometers are unpolluted by device motion [73], magnetic disturbances coupled with sensor placement offset can affect the seemingly simple estimation of H and lead to errors as much as 100° [74]. Furthermore, naive double integration of accelerometer readings cumulatively accumulates the effects of time-varying bias, gravity pollution and AWGN, causing errors in (ϕ) and (λ) to explode in a cubic manner [73][75][78][82], illustrated in Figure 2.1. In fact, the gyroscope within a MEMS inertial sensor is modeled as follows [76]:

$$\omega_{ib} = \underline{R^{bn}(\omega_{ie} + \omega_{en})} + \omega_{nb} + b_g + n_g \tag{2.5}$$

where, $\dot{b_g} \sim \mathcal{N}(0, Q_g)$ = bias gradient, $n_g \sim \mathcal{N}(0, \Sigma_\omega)$ = additive white Gaussian noise (AWGN), R^{bn} = rotation matrix from navigation to body frame, ω_{ie} = earth rate (7.29×10^{-5} rad/s), ω_{en} = transport rate and ω_{nb} = latent angular velocity. Assuming that the rigid body of interest is constrained on an immobile platform with respect to earth frame and is sufficiently close to the Earth's surface, the underlined term in the above equation can be considered 0. As a result, ω_{nb} is the uncorrupted angular velocity of the object. Due to bias instability (BI) and angular random walk (ARW) [77] resulting from AWGN, pink noise and thermal effects [78], the gyroscope suffers from time-varying drift in the long term, ideal for characterizing fast and agile movements [76]. In fact, for constant velocity v, the error in position estimate for one dimension x can be modelled as [78]:

$$\sigma_x(t) = v\sqrt{(2\cdot ARW\cdot\sqrt{t^3}/3)^2 + (BI\cdot t^2/2)} \tag{2.6}$$

The accelerometer model is defined as:

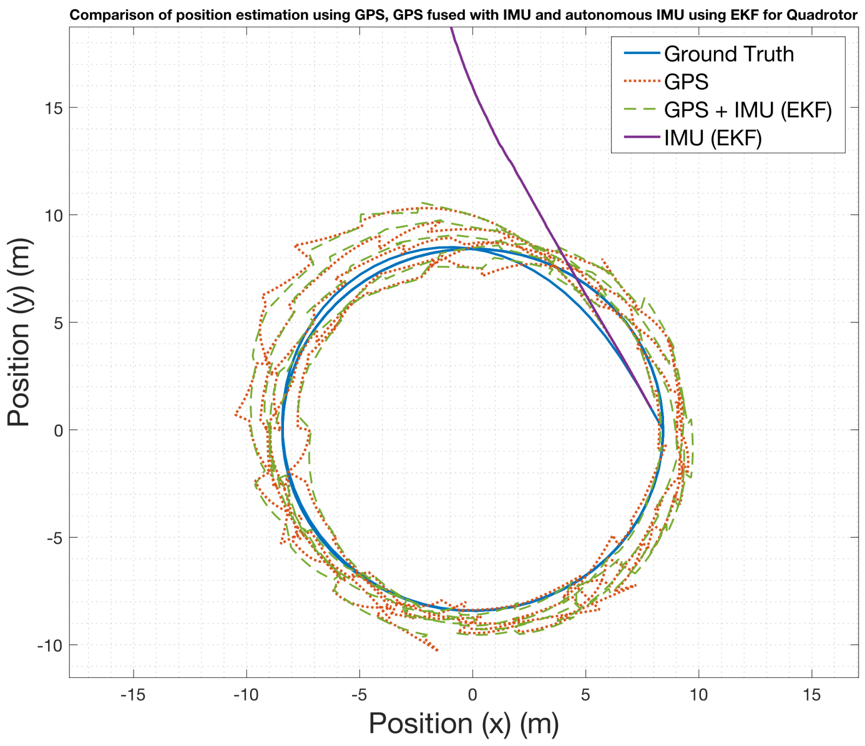


Figure 2.1: Example of localization of a quadrotor using GPS, GPS+IMU and autonomous IMU via Extended Kalman Filter (EKF). Fusion of IMU with GPS smooths the trajectory projections, while using autonomous IMU without error correction heuristics leads to drift.

$$f^b = R^{bn}(a^n_{ii} - g) + b_a + n_a, \;\; a^n_{ii} = a_{nn} + \underline{a_{CE} + a_{CO}} \tag{2.7}$$

where, a_{nn} = latent linear acceleration of body. a_{CE} = centrifugal component, a_{CO} = Coriolis component, g = gravity vector, $\dot{b_a} \sim \mathcal{N}(0, Q_a)$ = bias gradient and $n_a \sim \mathcal{N}(0, \Sigma_a)$ = AWGN. The accelerometer is noisy in the short term but does not drift significantly in the long term, ideal for slow movements and having complementary characteristics to that of the gyroscope. We assume that the centrifugal component is absorbed within g, while the Coriolis component is negligible for everyday observed velocities [76]. Mathematically, the compass can be modeled as:

$$b^n = R^{bn}m^n + n_b, \;\; m^n = [cos\delta \quad 0 \quad sin\delta] \tag{2.8}$$

where, $n_b \sim \mathcal{N}(0, \Sigma_m)$ = AWGN and δ = magnetic inclination due to Earth's magnetic dip.

The magnetometer requires frequent calibration to compensate for soft and hard iron disturbances and is usually noisy in indoor environments. Other errors that are not separately modelled in the IMU system models are axis misalignment errors in 3DoF, quantization error and scale-factor error [79], which are usually lumped into the IMU offsets and noises during calibration. Since the biases and noises of the IMU are time-varying, on-the-fly calibration is required, either opportunistically [80] or manually [81]. Given attitude estimate from sensor fusion, assuming that the IMU is rigidly strapped down to the object of interest, the heading angle θ_t is defined as the angle that vector $\vec{u_t}^{N,horz}$ makes with the horizontal axes fixed to the ground plane [83][141]:

$$\vec{\mathbf{u}_t}^{N,horz} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{C}_B^N(\mathbf{q}_t)\vec{\mathbf{u}_t}^B \tag{2.9}$$

$$\mathbf{q}_t = \mathbf{q}_{t-1} + \frac{1-\alpha}{f_{s,w}}\mathbf{R}_{ib}\mathbf{w}_t + \begin{bmatrix} \arctan\left(\frac{f_{y,t}}{\sqrt{f_{x,t}^2+f_{z,t}^2}}\right) \cdot \frac{180\alpha}{\pi} \\ \arctan\left(\frac{f_{x,t}}{\sqrt{f_{y,t}^2+f_{z,t}^2}}\right) \cdot \frac{180\alpha}{\pi} \\ 0 \end{bmatrix} \tag{2.10}$$

where $\mathbf{C}_b^N$ and $\mathbf{R}_{ib}$ are the direction cosine and rotation matrices respectively, $\mathbf{q}_t$ = orientation estimate, $\mathbf{u}_t{}^B$ = unit vector that is not parallel to the perpendicular axis of the ground plane, $\vec{\mathbf{u}_t}^{N,horz}$ = horizontal projection of $\mathbf{u}_t{}^B$, $\alpha \in (0,1)$ and $f_{s,w}$ is the gyroscope sampling rate.

Depending on target application and topography, heuristic drift reduction, magnetic anomaly detection, opportunistic calibration, quasi-static moment detection, particle filters, magnetic map matching and inertial signature verification are used to counteract heading estimation error [73][74][80][84][85], typically fused through an error-tracking indirect Kalman filter (KF). The KF is an iterative optimal state estimation algorithm (from fusing consecutive samples of single or multiple noisy indirect modalities) under Gaussian variations. It is composed of prediction (process or transition or time update) and correction (measurement update). KF is a subset of Bayes filter with Gaussian prior, linear process and measurement model with Gaussian noise and satisfying Markov property, with the goal of maximizing posterior probability. The prediction and correction equations of the linear KF [99] are as follows:

$$\text{Prediction}: \begin{cases} \hat{x}_{t,p} = A\hat{x}_{t-1} + Bu_t \\ P_{t,p} = AP_{t-1}A^T + Q \end{cases}$$

$$\text{Measurement}: \begin{cases} k_t = \frac{P_{t,p}C^T}{CP_{t,p}C^T+R} \\ \hat{x}_t = A\hat{x}_{t,p} - k_t(y_t - C\hat{x}_{t,p}) \\ P_t = (I - k_tC)P_{t,p} \end{cases}$$

where x = state, Q = process covariance noise, k = Kalman gain, y = measurement from sensor, R = sensor noise covariance. Given a prior and in absence of measurement model, covariance of prediction will increase with each step. Given a prior and measurement model, the measurement step will yield a posterior (belief) with lower covariance than prediction and measurement. Resulting estimate is optimal (minimum variance) and unbiased.

For displacement estimation, unmanned aerial, ground and underwater vehicles (UAV, UGV and UUV) typically fuse inertial sensors with GPS, LIDAR, camera and RADAR via KF variants for localization, complemented via map information from known localization space [86][87]. Typical aid includes magnetic map matching, flow-sensors and wheel odometers, with nonholonomic constraints on the motion. Satellites and spacecrafts perform nonlinear Bayesian fusion of physics-based kinematics models with inertial sensor measurements for attitude estimation [88], coupled with position information from GPS, kinematics models (e.g. ephemeris and almanac), relative angle measurements (e.g. parallax) or ground control [89][90]. For slow and predictable motion profiles such as in wildlife tracking, ecologists formulate animal-specific belief-based constraints on velocity, transportation modes and Boolean decision-trees to mitigate drifts and sensor errors, occasionally fusing GPS through KF variants when available [72][91][92].

PDR systems typically decompose position estimation problem into direction estimation and stride length estimation [74], the latter of which is further partitioned into transportation mode classification, gait cycle segmentation and step length estimation (SLE) subject to environmental constraints and iterative updates [84][85][93]. For DR via foot-mounted IMU, information about the linear and angular velocities of the foot during swing and stance phases for various motion primitives can be exploited to constrain explosion of errors in step counting [84][85][94]. This is referred to as zero velocity update (ZUPT) or zero angular rate update (ZARU) pseudo-measurements [95], segregating the gait cycle into identifiable chunks either coarsely or granularly [84], as shown in Figure 2.2. Temporal and frequency-domain analysis (e.g., wavelet and Fourier transforms, level crossing detection, extrema detection, autocorrelation, Hidden Markov Models (HMM) and local variance detection) are used to extract recurrent and temporal contextual dynamics in the gait cycle to improve robustness of gait phase identification against AWGN for various activity modes [74][84][85][96]. Typical SLE approaches include physiological knowledge injection (e.g., Weinberg SLE), linear regression upon step frequency with aiding covariance heuristics, global acceleration extrema difference, knowledge of virtual landmarks and floor plans or use of KF variants [74][84][85][93][97][98].

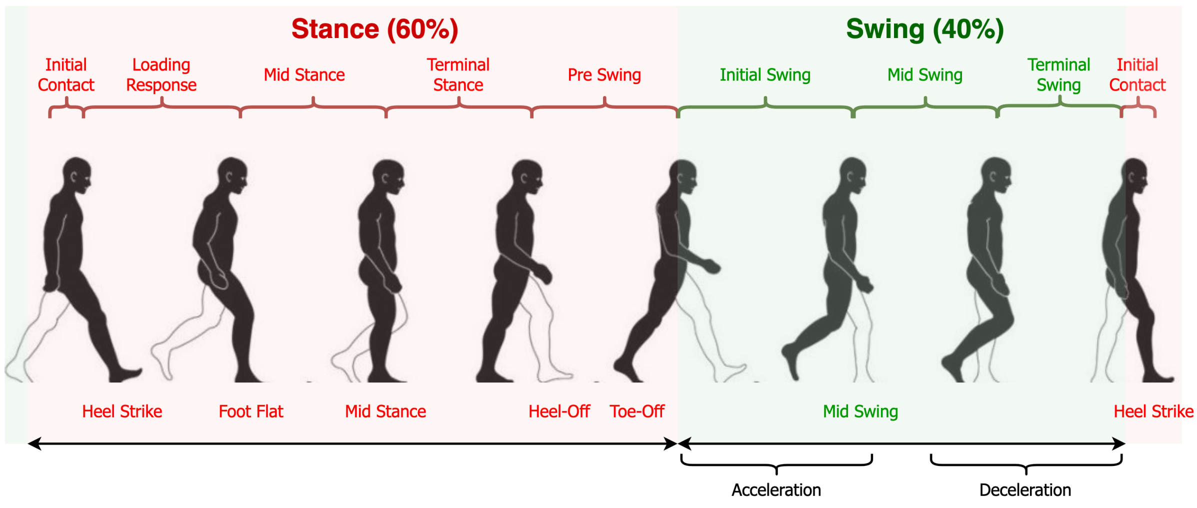


Figure 2.2: Gait cycle showing different phases of human walking [148].

2.3 Machine-Learning Approaches for Autonomous Inertial Odometry

While model-based inertial localization systems are computationally efficient and have been shown to run in real-time, designing generalizable heuristics poses a significant hurdle in the deployment of real-time inertial dead-reckoning systems, with no “one size fits all” analytical solution to the problem. The system models are linear approximations of the state evolution in the real-world, which do not translate accurately in the long run for eclectic scenarios due to non-optimal parametrization. As a result, several machine-learning approaches have emerged which are able to capture high-dimensional contextual dynamics in the non-linear domain void of human knowledge to either heuristically complement physics-based state-estimators or derive the location end-to-end, providing superior translational error characteristics over classical approaches for longer time periods while competing with other egomotion techniques.

2.3.1 Aiding Physics-Based Models

Since AI models are adept at filtering out noise and irrelevant information while bringing out useful features from sensors in the wild [15], the early papers for AI-enabled inertial odometry focused on using ML as an aid for modeling covariance parameters of Bayesian filters on-the-fly or denoise inertial sensor readings, provide pseudo-measurement updates to KF or estimate heuristic parameters in the presence of eclectic deployment scenarios.

2.3.1.1 Modeling Sensor Covariances and Denoising

A rational solution to handle the provenance of the curse of drift involves eliminating the root of sensing uncertainty on-the-fly before state estimation step. A few papers attempt to model the sensor covariance characteristics and perform indirect denoising using ML techniques to aid state estimation using Kalman or non-Bayesian filters [100]-[109].

Tekinalp et al. [104] and Fan et al. [106] proposed the use of ensemble multilayer perceptrons (MLP) for transfer alignment and one-time calibration of integrated strapdown inertial-celestial navigation system (SINS-CNS) for ballistic missile guidance to mitigate axis misalignment, position and velocity errors, accelerometer bias, gyroscope drift and scale factor errors. The MLP aids an indirect KF compensate initial state estimation errors when the star tracker (master) is absent during boost phase ($\sim$ 40 seconds) and only the SINS (slave) is being used for DR, which cannot be accounted for by the CNS and may cause dispersion of target point [106]. To make the classifier invariant to eclectic scales, spans and domains of motion and enforce location-independent precise spatial connections among adjacent noisy inertial samples [15], deep CNN can further enhance the sensing uncertainty profile estimation without explosion of weight count and having minimal chance of overfitting [101]. Nevertheless, the techniques in [101][104] [106] are one-shot and non real-time, with the AI coming into play within a limited spatial and temporal domain. Affixing a neural network block dedicated to denoising in the state estimation filter yields real-time and on-the-fly noise reduction. Proposed neural blocks include:

- *Deep Feedfoward Neural Network* (D-FFNN) - D-FFNN have been shown to be effective in extracting noise-free state-estimates from noisy inertial Kalman predictions coupled with quadrotor dynamics to enhance attitude estimation in presence of multirotor actuator microvibrations [100].

- *Dilated CNN* (D-CNN) - Apart from being invariant to random gyrations and exhibiting generalizability under unseen trajectory projections, D-CNN with Huber loss are robust to training data anomalies, maintains sample order and have a larger receptive field for extracting differential features over a larger local window of inertial samples without losing resolution compared to vanilla CNN or D-FFNN, all within a computationally inexpensive architecture [103]. Brossard et al. [103] showed that an open-loop attitude estimator with D-CNN block outperforms state-of-the-art (SOTA) autonomous inertial orientation estimation techniques for multirotors and hand-held inertial sensors and even competes with egomotion without using any camera.

- *GyroNet and FoGNet* (Bi-LSTM) - Long Short-Term Memory (LSTM) networks are capable of learning long-term temporal dynamics and invariant to irregular time delays and vanishing gradients [15], whereas bidirectional-LSTM can exploit recurrence from both past and previous context [111]. Gyronet and FoGNet [107] feature 2-layer Bi-LSTM, each designed for autonomous vehicle localization with different loss functions for MEMS IMU and fibre-optic gyroscopes (FoG) respectively. Gyronet provides

6× mean improvement in attitude estimation over one-time manual calibration, Butterworth filtering and complementary filter using MEMS gyroscopes, while FoGNet coupled with wheel odometry was shown to outperform Lwoi [108] and RINS-W [110] for autonomous vehicle localization.

- *Lwoi* (Gaussian Process) - Kernel tricks allows linear algorithms (e.g., linear SVM) to operate in a high-dimensional non-linear feature space within reasonable computational bounds in a scalable and accurate fashion. Lwoi trains a Gaussian Process (GP) to minimize the difference between ground truth position and predicted position by the dynamical system model represented as GP, while using a DNN-based approximate kernel learning method, with the GP parameters optimized with stochastic variational inference. The corrections generated by Lwoi can be fused with an extended KF (EKF) for location estimation [108].

The state estimators in aforementioned cases are agnostic to the corrections being made by the neural network, which may lead to pose predictions that are statistically non-optimal under non-ideal gradient descent convergence. An alternative approach proposed in [102][105][109] (inspired from adaptive KF training [112] but independent of state estimates or measurement errors) involves using AI blocks to dynamically update KF noise covariance parameters instead of sensor errors. In [102], an invariant EKF supported by wheeled vehicle dynamics is tightly coupled to a 2-layer temporal-CNN (TCNN), which dynamically adapts measurement noise covariance from raw IMU measurements (with process noise covariance optimized during training) for lateral and vertical vehicle movements under loose constraints. The technique was shown to compete with LIDAR and egomotion while adding only a 15% computational overhead over vanilla EKF. The work was extended to include process noise covariance in [105], using TCNN complemented with a denoising autoencoder (DAE) to enhance noise removal through multi-task learning. The DAE rejects abnormal noise profiles beyond its latent space distribution to nonholonomically constrain the noise covariance within finite bounds. Gao et al. [109] introduced a deep deterministic policy gradient (DDPG) algorithm (derivative of model-free actor-critic algorithm in deep reinforcement learning (RL)) with

MLP target networks to dynamically estimate process noise covariance of an EKF. However, the algorithm requires cloud support and is not suitable for on-board deployment.

Figure 2.3 illustrates the concept of denoising and covariance control using DL in inertial localization.

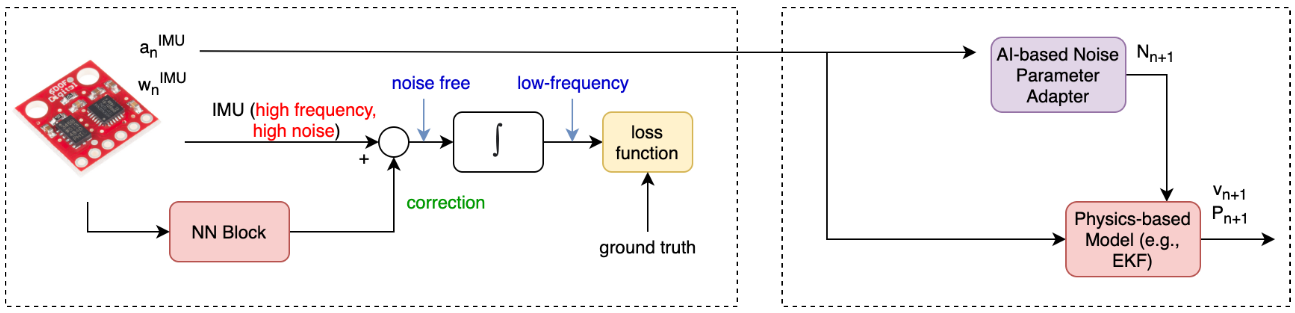


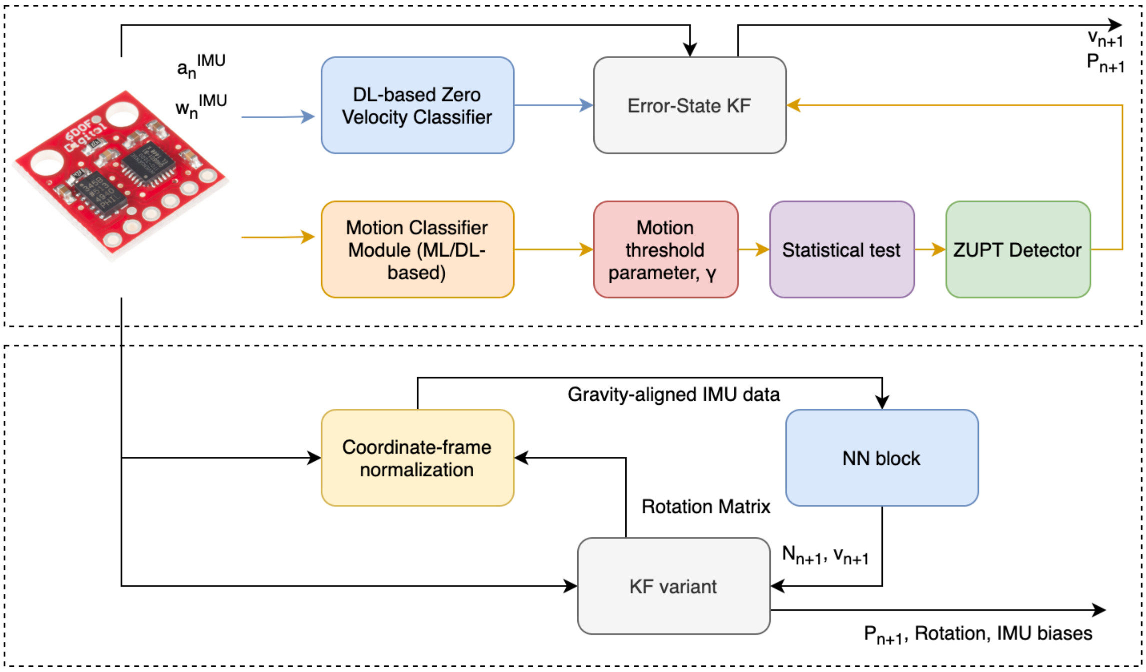
Figure 2.3: (**Left**): Denoising IMU using neural block [103] (**Right**): Adaptively updating filter noise statistics using neural block [102].

2.3.1.2 Velocity Profile Heuristics

While threshold-based step detectors and SLE with beliefs on accelerometer statistics are widely deployed for indoor PDR systems, they are not robust to varying gait patterns or sensor positions beyond the limited classes of common resolved activities. AI based velocity profile detectors and SLE are able to extract coveted velocity intervals for step detection and adapt stride length based upon discerned motion patterns. The earliest AI-based SLE technique [115] used a MLP trained with handcrafted features (extrema and variance of acceleration vector sum) for adaptive SLE based upon perceived velocity from helmet-mounted IMU readings, coupled with zero-crossing step detector. Recent proposals in this domain exploit deep neural architectures void of manual feature extraction. Wagstaff et al. [113] used a 6-layer LSTM network for ZUPT to constrain errors of an EKF based SLE robust to various activity primitives. An extension of [113] includes a SVM motion classifier to adaptively update the ZUPT threshold for cascade detector along with the LSTM network to aid an EKF based SLE to yield a PDR system robust to gait and subject variations [114]. They also proposed the addition of Gaussian noise and downsampling to training data collected from a particular IMU as a data augmentation technique, transforming the data across other IMU without the need to recollect data for various IMU models. Velocity profiles can also aid DR in wheeled robots and vehicles; ZUPT and ZARU allows attitude and bias estimation corrections, while priors on lateral and vertical velocity improve long-term pose estimation accuracy [110]. RINS-W [110] used 4 binary LSTM networks to detect the aforementioned profiles using commodity IMU and an invariant EKF for final pose estimation, competing with FoG-wheel odometry fusion techniques.

2.3.1.3 State Advisors

State advisors provide indirect information about pose and orientation (such as velocity and displacement) from inertial readings to physics-based filters, aiding in the decision making process. The HDR algorithm [77] indirectly reveals that average speeds over short time intervals of humans are reasonably constrained even under varying gait, terrain and subjects. Instead of explicitly specifying pose bounds under eclectic conditions, a CNN can be trained to regress momentary velocities directly from a finite window of raw IMU readings covering a range of motion primitives and topography as well as various test subjects [116]. The velocity information is then fed to an EKF as pseudo-measurements update along with IMU readings (as system model) for reporting the final position, velocity and orientation [116]. In [117], the authors directly regress 3D momentary displacements and measurement noise covariance using a 1D Resnet18 CNN without requiring initialization, which is fed to a stochastic cloning EKF (SC-EKF) as measurement update. The SC-EKF uses a strapdown inertial kinematic framework as system model and estimates 3D position, velocity, orientation and sensor biases, with the ability to to constrain neighboring poses by keeping track of past states and treating the measurement updates in heading agnostic coordinate frames (HACF) to address varying sensor orientations. Under ever-changing environments such as an AUV navigating in choppy waters, conventional physics-based filters can have difficulty estimating the AUV heading due to sculling velocity error (coupling of linear and angular motion) and rapid changes in pitch angle, the latter of which can cause noisy gyroscope readings and errors in coordinate frame translation. Xie et al. [118][119] showed that a DNN can aid in mapping EKF-denoised accelerometer readings to usable pitch angles, with the training data computed via non-linear coupling of filtered accelerations, depths from pressure sensors and locations tracked through acoustic localization.



a_n^{IMU}
w_n^{IMU}
DL-based Zero Velocity Classifier
Error-State KF
v_{n+1}
P_{n+1}
Motion Classifier Module (ML/DL-based)
Motion threshold parameter, γ
Statistical test
ZUPT Detector
Gravity-aligned IMU data
Coordinate-frame normalization
NN block
Rotation Matrix
N_{n+1}, v_{n+1}
KF variant
P_{n+1}, Rotation, IMU biases

Figure 2.4: (**Top**): AI-based velocity profile heuristics [113] (**Bottom**): Using AI to provide displacement and covariance pseudo-measurement advice to physics-based filter [117].

2.3.2 End-to-End Models

Bayesian filters require an accurate model of the evolution of the position, velocity and orientation (as well as IMU errors) in terms of the incoming measurements. Such system models are only linear approximations of the state evolution in the real-world, which do not translate accurately in the long run for eclectic scenarios due to non-optimal parametrization. In other words, model-based approaches are generally computationally inexpensive and accurate under the system model that they have been designed for, however, they are not robust to deviations from assumed system model such as for non-linear complex motions or deployment under different domains (domain variance) [82][120]. Recent breakthroughs in ML and DL have overcome the requirements of formulating an accurate system model or heuristics/priors for inertial odometry, resulting in several end-to-end learning-enabled inertial odometry models able to handle non-linearities associated with inertial localization which are otherwise hard to model mathematically.

2.3.2.1 Classical ML and Continuous Double Integration

Diamant et al. [121] first showed the application of an end-to-end learning pipeline for pose estimation of AUV and ships in choppy waters. They used a constraint expectation–maximization (EM) algorithm to classify accelerometer data into bins of similar pitch angles and then projected these onto the local north–east horizontal plane to account for erroneous heading angle estimation (under the assumption that the vessel motion perfectly correlated with ocean waves and slowly time-varying) in that same frame due to sculling velocity error. The EM algorithm thus corrects accelerometer sequences for stochastic changes in pitch angle, which are double integrated to yield position without the need for a gyroscope or a complex system model incorporating wave mechanics. In [122], Bi-LSTM was applied for step detection and linear regression on variance of accelerometer readings for 3 kinds of movements (walk, run, walk backwards) was applied for SLE, with the goal of creating a sensor location and transportation mode independent PDR system. The network was trained for four phone positions (pocket, foot, hand and bag) with random orientations and was shown to be invariant to transformation and distortions of the input patterns caused by phone placements and rotations, outperforming widely applied model-based classical PDR algorithms, including peak detectors, frequency domain SLE, map matching, MLP and inverted pendulum. RIDI [75] removed bounds on motion primitives and SLE, using a SVM based sensor placement detector (pocket, hand, bag, body) and SVR-based (2 each for magnitude and direction, 8 total) velocity regressor to estimate velocities from accelerometer and gyroscope readings for various kinds of motions. The resulting velocities are used to correct accelerometer readings, which are then double integrated to obtain position. To make the velocity invariant to phone transformations at each position of the body, rather than hoping for the classifiers to discard domain-specific irrelevant information, the authors leveraged the gravity direction to define a stabilized-IMU frame using orientation estimates from the phone, where the device pitch and roll are eliminated from by aligning its y-axis with the gravity vector, while the yaw ambiguity is handled during training by regressors. RIDI was shown to be robust across unseen subjects for unrestricted motion primitives and step-lengths, as well as for different devices.

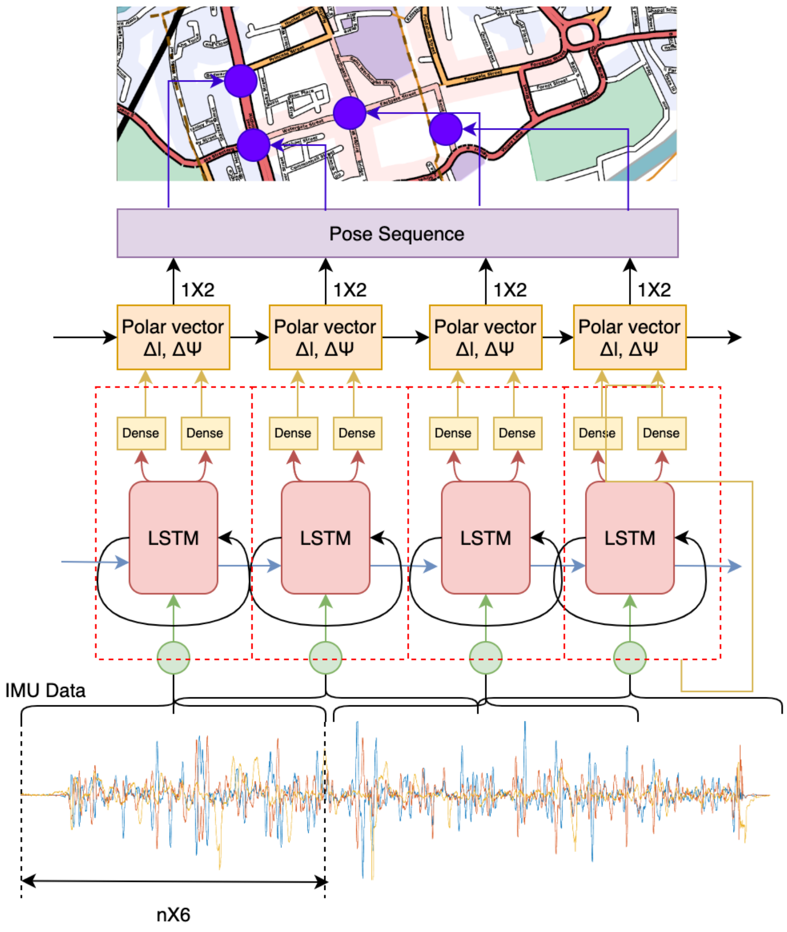


Figure 2.5: Concept of neural inertial sequential learning [82][120][131].

2.3.2.2 Sequential-Learning

A rudimentary problem with the aforementioned approaches is the growth of double integration error due to continuous integration. IONet [120] was the first end-to-end DL paper that broke the cycle of continuous integration by treating inertial DR as sequential learning problem based on Newtonian physics, illustrated in Figure 2.5. The goal is to achieve pseudo-independence across inertial windows (as windows are not truly independent) to estimate the change in navigation state over each window using a Bi-LSTM network. Assuming zero vertical velocity for PDR, the authors showed that the horizontal displacement and heading over a window can be expressed as a function of initial velocity, the gravity direction, accelerometer and gyroscope readings in the polar coordinates, with the task of estimating the initial velocity and gravity in each window (which are treated as latent states and instead the neural network outputs heading and displacement change). The algorithm did not rely on device orientation, and was shown to be robust across unseen subjects, unseen phones and various phone rotations for three positions (pocket, hand, bag), while also useable for tracking wheeled robots. The success of IONet spawned a series of successors building upon the sequential learning problem, notably RoNIN [82] for PDR, AbolDeepIO [123] for aerial vehicle DR, VeTorch [125] for autonomous vehicle localization and NavNet [124] for AUV positioning. Klein et al. [126] evaluated the performance of three neural architectures for PDR, namely CNN-based gain controller for Weinberg SLE with peak detection as step detector, CNN-based SLE with the same step detector as before and an end-to-end CNN-based (Resnet-18) for sequential DR like IONet without step detection, while using a CNN-LSTM network for phone position classification for all three models. The final architecture, which treats DR sequentially with independent time windows, outperformed both the hybrid (neural network aiding physics-based model) and the non-sequential model on the RIDI dataset. Lima et al. [127] showed that translation mean absolute error (MAE) and quaternion multiplicative error are ideal loss functions for such architectures in eclectic scenarios (e.g., for PDR and quadrotor DR).

- RoNIN [82] proposes three neural architectures for PDR, namely Resnet with strided velocity loss, LSTM and TCN latent velocity loss, using phone orientation and inertial streams as input. It also uses a LSTM-based body heading network to account for abnormal motions (e.g., walking sideways) and proposes a heading agnostic coordinate frame (HACF) instead of stabilized IMU frame (any frame whose z axis can be aligned with gravity) to account for completely unrestricted phone orientations and phone positions. During training, ground truth trajectories are randomly rotated to emulate random HACF in line with data augmentation. Tested on RIDI, OxIOD and RoNIN (the largest inertial PDR dataset till date) datasets, RoNIN was shown to overall outperform RIDI, IONet, naive double integration and ZUPT-based methods.

- AbolDeepIO [123] uses two separate LSTM channels to map accelerometer and gyroscope readings to different latent representations (to decouple individual errors and biases) while feeding the sampling rate of the inertial sensor through a third LSTM channel to make the displacement and heading regressor robust to sampling rate jitter. The high-level correlated features are then fused using slow fusion. The authors also use noise augmentation to inform the pipeline of presence of sudden noise variation and drift in inertial sensors during rapid maneuvers, commonly encountered in quadrotors and robots. The framework was shown to outperform IONet and compete with visual-inertial odometry techniques for quadrotor DR and outputs 3D trajectory and orientation. The authors later applied the multi-channel LSTM architecture for robust 3D orientation estimation (called OriNet [128]), where they fed the current gyroscope sample, previous gyroscope sample, sampling rate and previous quaternion estimate as inputs to the network and weigh state estimate from aggregated past states (from an MLP) with current measurements to update current state. The previous state quaternion helps transfer IMU measurements from world to body frame, acting similar to how a KF keeps track of past state and weighs new measurements using Kalman gain before updating the current state. A genetic algorithm was proposed to calibrate the initial bias of the IMU.

- NavNet [124] asynchronously combines inertial sensor and doppler velocity log (DVL) through separate LSTM networks (both current and previous readings for robustness against sensor errors), followed by attention layers to capture relevant long-term contextual information across timesteps with information fusion from both sensors through an MLP. NavNet can also act as state-advisors for classical EKF or unscented KF for high-frequency state-estimation.

2.3.2.3 Transformers and Autoregressive Models

Recurrent sequence-to-sequence architectures like IONet and RoNIN have limited capability in discarding domain-specific features in favor of relevant information and are thus unable to handle truly unrestricted phone position, phone model, phone orientation, changes in device physics and unseen subjects without the presence of sufficient domain invariance in the training set. This results in the need for large training sets with sufficient variance and models with large number of parameters to capture required complex and semantic information within the latent space, which are not suitable for real-time deployment on edge devices due to large inference time, memory and compute requirements. Transformer networks [129], on the other hand, are domain-invariant and are able to extract relevant information required for regression and discard noisy domain-specific motion sequence transformation in an unsupervised manner without the difficulties of training LSTM or gated recurrent units (GRU), adapting well in multiple deployment scenarios thanks to multi-head self attention [130]. The resulting models are also computationally efficient with fewer parameters while allowing parallelization thanks to sequential decoupling. In [129], the authors of IONet used a transformer network to generate domain invariant IMU sequences from raw IMU data coming from various phone placements, rotations or motion types in a completely unsupervised fashion, so that one does not need to collect labelled data for various domains. The proposed transformer architecture consists of:

- *Sequence (auto)encoder* - Finds hidden domain invariant representations from raw IMU sequences parametrized by domain vector, finding features that do not form separate clusters.
- *GAN* - The generator produces synthetic IMU data in target domain, while the discriminator encourages the generator to produce reasonable time-series data.
- *Reconstruction decoder* - Reconstructs synthetic IMU data in the source domain via constraints or conditions to create sequences that are better for learning the domain invariant features by other neural networks.

- *Polar vector predictor* - Generates heading and displacement labels for subsequence in the polar coordinate system, hence can be thought of as the PDR system itself.

The computational efficiency of generative feed-forward models thanks to lack of recurrent units can be exploited to produce efficient models suitable for deployment on edge-devices, while still retaining pseudo-sequential modeling. L-IONet [131] was the first neural inertial architecture targeted for real-time deployment on smartphones, derived from IONet and currently the least resource-intensive end-to-end DL-based PDR algorithm. L-IONet uses a generative autoregressive model with D-CNN layers (which was also later shown to be computationally inexpensive for aiding physics-based DR in [102][103]) suitable for processing long time-series sequences without the training difficulties or cost of vanilla recurrent architectures, deriving benefits from transformer networks while achieving similar (often better) translational error.

2.4 Machine-Learning for Resource-Constrained Devices

Internet-of-Things (IoT) can be thought of as a large-scale multimodal heterogenous sensing system consisting of devices at the edge and the cloud making rich inferences from large amounts of data using AI [15]. Given the depth and scale of penetration of such ”frugal” smart objects in our daily lives for eclectic applications (e.g., mHealth, smart homes, autonomous vehicles, decision support systems and agriculture), considerable endeavors have been undertaken to bring AI to the edge given energy efficiency, low cost, zero latency, data security/privacy and reliability of processing temporal streams at the edge [32]. However, there are several challenges in implementing AI for resource-constrained devices:

- The compute and specification spectrum of TinyML devices are heterogenous and non-standardized, with hundreds of thousands of IoT device varieties available in the market. It is not computationally feasible to systematically optimize ML/DL models for each device one at a time. Furthermore, running graphics processing unit (GPU) workstations for long periods increases carbon footprint [23][24].

- The memory of TinyML devices are significantly smaller than cloud or even mobile devices. For example, an Arduino BLE33 has only 320 kB of SRAM and 1 MB of flash, compared to 4 GB of RAM and 64 GB of storage on a smartphone. A GPU can have 16 GB of memory on a workstation with secondary storage in the order of terabytes. As a result, optimizing larger models for smaller devices directly using techniques such as dimension reduction, pruning, quantization and model compression alone are insufficient to mitigate loss of accuracy of smaller models [23][24].
- Decision boundaries and activation maps in the wild need to be periodically adapted to seasonal variations in the input stream to account for client behavioral changes. However, on-device training consumes significantly more memory than on-device inference [151][152].

2.4.1 Fundamentals of TinyML Paradigm for Deep-Learning

Classical methods for dimension reduction commonly used in large-scale data-mining include matrix factorization and principal component analysis (PCA) techniques such as singular value decomposition (SVD) [33], flattened convolutions [33], non-negative matrix factorization (NMF) [34] and linear discriminant analysis (LDA) [35]. Techniques for processing samples with highly non-linear distribution in the higher dimensions include uniform manifold approximation and projection (UMAP) [36], t-distributed stochastic neighbor embedding (t-SNE) [37] and autoencoders [38][39]. Figure 2.6 shows an example of matrix factorization and manifold projection in the ”20 Newsgroup dataset”. For optimizing DL models in software, the following techniques are used:

- Pruning, quantization and model compression.
- Hardware-aware NAS.
- Lightweight convolutional and recurrent block design.

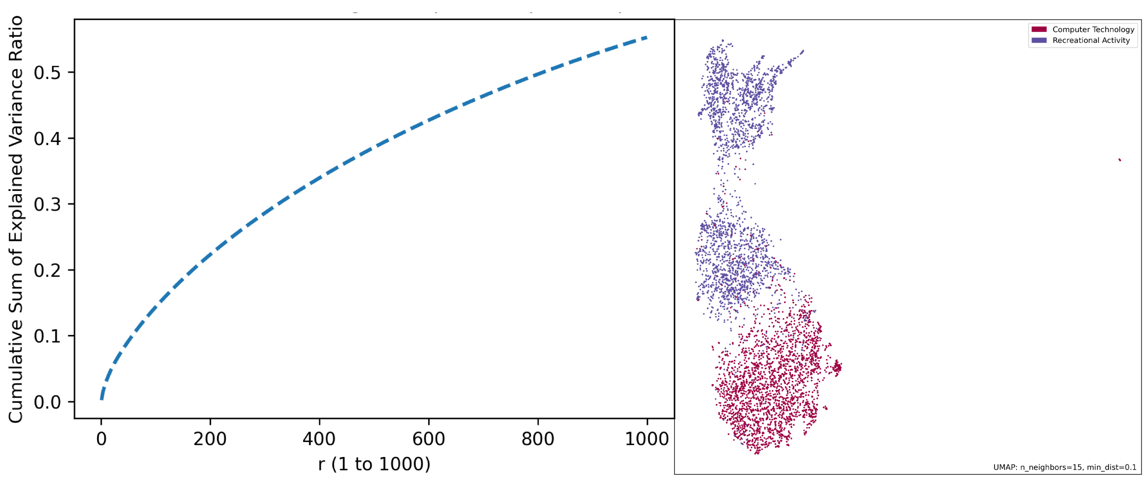


Figure 2.6: (**Left**): Variance retention capability of the top r principal components of SVD decomposition for the TF-IDF matrix in the "20 Newsgroup dataset" (**Right**): Visualization of low-dimensional embedding of two classes in the "20 Newsgroup dataset" via UMAP. The classes are linearly separable in the manifold space.

2.4.1.1 Pruning, Quantization and Model Compression

Han et al. [40] first showed the concept of pruning, quantization and model compression jointly in the context of pre-trained deep neural networks (DNN). Pruning [41] refers to masking redundant weights (weights lying within a certain activation interval) and representing them in a row form. The network is then retrained to update the weights for other connections. Quantization [42] accelerates DNN inference latency by rounding off weights to reduce bit width while clustering the similar ones for weight sharing. Compression (using Huffman coding) represents common weights with fewer bits. The authors showed that combining these techniques can drastically reduce the size of SOTA DNN such as AlexNet (35×), LeNet-5 (39×), LeNet-300-100 (40×) and VGG-16 (49×) without losing accuracy.

One of the most common fixed-precision quantization techniques include the uniform affine quantizer, where a real number is mapped to a fixed-point representation via a scale factor (step-size) and zero-point (map floating point 0 to integer 0) [43][44]. It allows 8-bit representation of weights and activations, while bias vectors are represented as 32 bits

[43][44]. The products of weights and activations (from previous layer) are collected in a 32-bit accumulator, added to the bias, scaled/casted down to 8-bits before being clamped by the activation function to produce 8-bit output. The inference framework also represents matrix multiplication as integer-only arithmetic [43]. Such post-training 8-bit quantization (per channel quantization of weights or per layer quantization of activations) has been shown to reduce model size by 4× without negligible loss in accuracy, while speeding up inference by 2-3× on generic central processing unit [44]. For smaller models (e.g., MobileNet), quantization-aware training is recommended to mitigate layer-wise quantization error due to large range of weights across channels rather than applying post-training quantization [43][44]. This is usually done by injecting simulated quantization operations during the forward pass to the neural network graph. Special layers such as batch-normalization are folded into the weights during this phase. The weights are usually clamped within a finite range, while it is recommended to not clamp the activation ranges [44]. The steps for quantization-aware training are as follows [43][44]:

- Train (or use pre-trained) a floating-point model in Tensorflow (TF).
- Insert fake quantization operations using quantization rewriter in TF at weights and activation spots.
- Train the new model until convergence.
- Convert the model to optimized C code for running on TinyML devices using TF converter.
- Execute the model (using TFLite interpreter or other COTS TinyML tools).

The aforementioned quantization paradigm assigns the same bit-width to all the layers. However, different layers have distinct compute and memory requirements requiring non-uniform quantization precision, rendering the fixed-precision bit assignment non-optimal. Wang et al. [45] proposed using deep deterministic policy gradient (DDPG) to assign mixed-precision bit-width for weights and activation for each layer, using validation accuracy as reward, while receiving information from a hardware simulator on energy, latency and memory consumption to tailor hardware-aware quantized network graph. The automated process drastically reduces human effort and time in optimizing the layer-wise quantization search space. A COTS tool that performs mixed-precision quantization is CMix-NN [46][47], which converts a MobileNet model to a mixed integer precision model of 2, 4 or 8 bits (asymmetric mixed low-bit-width) for weights and activation values of the convolution kernel. It represents a network subgraph as a quantized convolutional layer (QCL, made up of unpack block to load operands, vectorized MAC unit to aggregate output of convolution and a per-channel/per-layer compressor block), with the non-convolutional parameters absorbed into the activation function via integer channel normalization (rule-based iterative method) and driven by memory constraints of target hardware. Figure 2.7 illustrates the process of pruning, quantization and model compression, taking LeNet, AlexNet and VGG-16 as original networks [40].

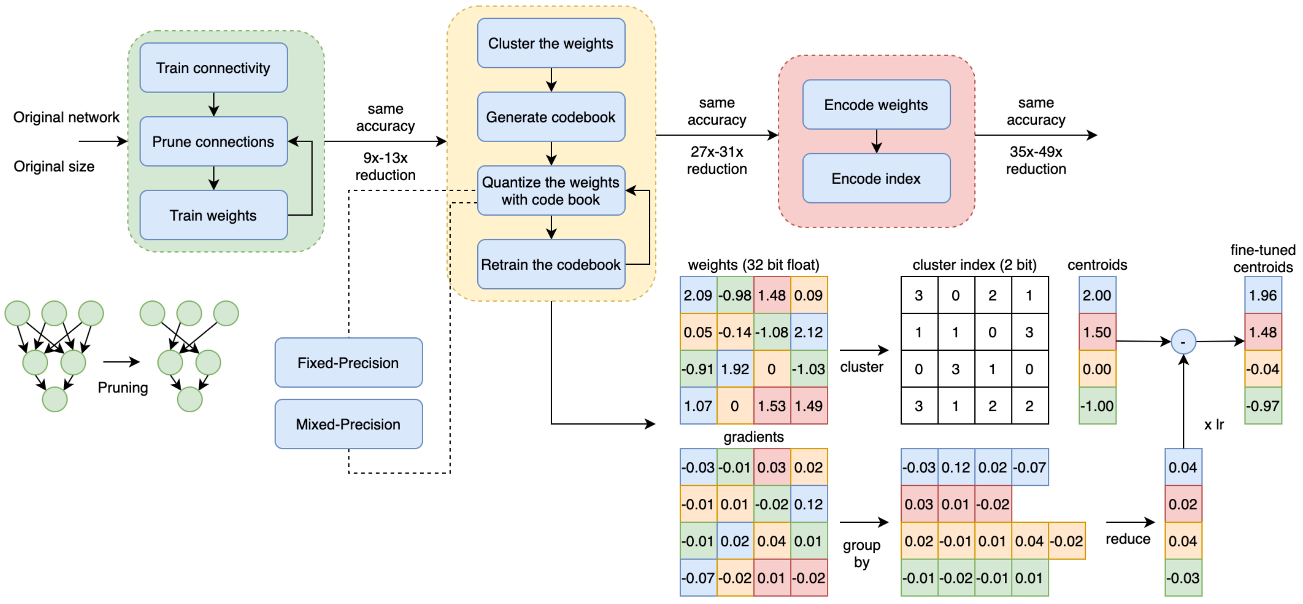


Figure 2.7: Summary of pruning, quantization and model compression [40].

2.4.1.2 Lightweight Convolutional Neural Blocks

Han et al. [48] introduced several micro-architectural enhancements to further improve DNN compression in context of AlexNet beyond deep-compression. These include replacing 3×3 convolution filters with point-wise filters (reduces parameter count), decrease input channel count to 3×3 filters using point-wise filters as a linear bottleneck (reduces parameter count) and late downsampling to enhance feature maps (improves accuracy under budget constraints). The resulting CNN (called SqueezeNet) consists of stacked ”fire modules”, with each module containing a bottleneck / squeeze layer (layer with point-wise filters) and an expand / excitation layer (mix of point-wise and 3×3 filters). Optional bypass (simple and complex) can also be added to emulate ResNet-like behavior. SqueezeNet reduced size of AlexNet by 50×, and combining pruning, quantization and model compression reduced the size by 510× without loss of accuracy. The architecture was later generalized in [49]. Other key enhancements, introduced in MobileNetsV1 [50], include the use of depth-wise separable convolution, which consists of channel-wise convolution followed by a bottleneck layer, the use of width multiplier to control the input width of a layer and resolution multiplier to control the resolution of inputs to the network. Depth-wise separable convolution is 9× cheaper and induces 7-9× memory savings over vanilla convolution in AlexNet for 3×3 filters. ShuffleNetV1 (which achieved 13× compute improvement over AlexNet) [51] introduced the concept of channel shuffle for group convolutions (group convolution refers to the use of various convolutional kernel sets on the same image), which improves the semantic relation between input and output channels across all the groups, as well as using point-wise group convolution with channel shuffle before the expand layer instead of bottleneck layer. This allows ShufflenetV1 to use wide activation maps; a channel multiplier was used to scale the number of channels. The number of channels can be further reduced (thinner network) by providing each layer activation maps of all preceding layers (channel-wise feature concatenation, introduced in DenseNet [52]), which encourages reuse and stronger propagation of low-complexity diversified feature maps and gradients while drastically reducing network parameter count. MobileNetV2 [53] introduced the concept of inverted residuals (gradient highway exists between narrow parts of the network rather than wide parts to reduce parameter count) and linear bottleneck (to counteract loss of performance at the output of inverted residuals by enforcing a linear output at the last convolution of a residual block). Figure 2.8 summarizes the aforementioned lightweight convolutional blocks.

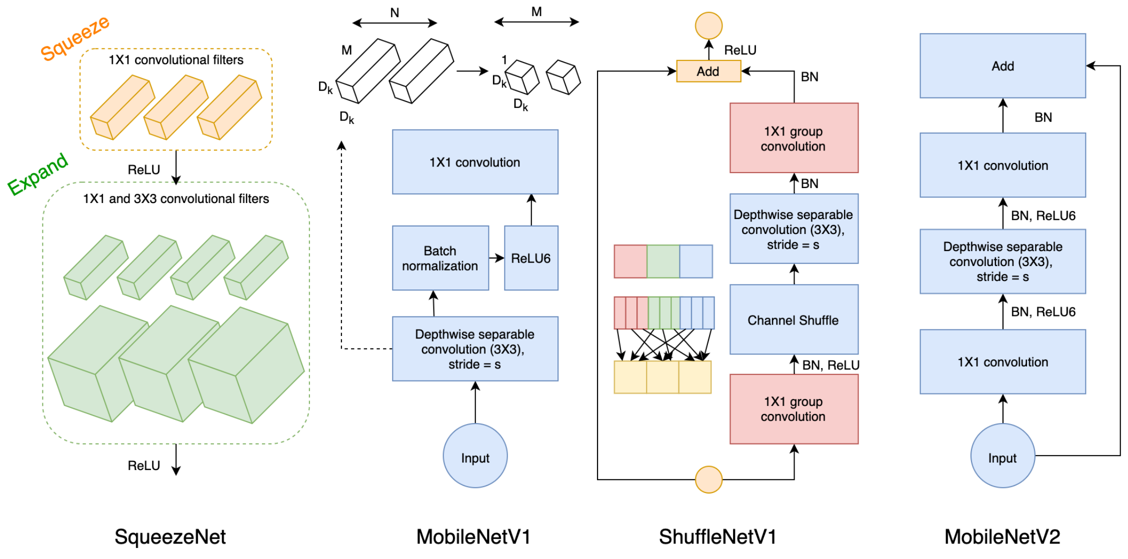


Figure 2.8: Advances in lightweight convolutional neural blocks: (a) Fire modules containing pointwise and 3×3 convolutional kernels [48][49] (b) Depthwise separable convolution [50] (c) Channel shuffle [51] (d) Inverted residuals and linear bottleneck [53].

For scalable and efficient real-time object detection, EfficientDet [54] introduced a bidirectional feature pyramid network (Bi-FPN) as the feature fusion network (neck) to aggregate features at different resolutions with two-way information flow. It is formulated by optimizing feature network topology found through NAS via addition and removal of nodes and edges. It also includes compound scaling of weight, depth and resolution (using heuristics) under target architecture memory and compute constraints for the backbone network (EfficientNet pre-trained on ImageNet), neck and bounding box prediction network (head). EfficientDet was tested on COCO and PASCAL VOC dataset and was shown to outperform YOLOv3, RetinaNet, AmoebaNet (with NAS-FPN), Resnet (with NAS-FPN) and DeepLabV3 across 7 scaling configuration in terms of mean average precision (mAP) and latency, while being 4–9× smaller and using 13× - 42× less FLOPs. Scaled-YOLOv4 [55] converts portions of FPN of YOLOv4 to cross-stage partial networks (CSP) [56], which saves up-to 50% computational budget over vanilla CNN backbones (such as DarkNet used in YOLO[57]). It also uses CSP-OSANet as backbone architecture, which has smaller computational complexity than DenseNet architectures for scaled-YOLOv4 aimed for TinyML devices, while scaling up depth, width, resolution in a compound manner like EfficientDet for larger YOLOv4 implementations. CSP-OSANet requires less number of FLOPS and minimizes convolutional Input/output.

2.4.1.3 Lightweight Recurrent Neural Blocks

FastRNN [58] adds a weighted residual connection with two scalars between RNN layers to stabilize gradients during training without adding significant compute overhead. FastGRNN [58] reuses the RNN matrices and converts the residual connection to gates while encouraging the matrices to be low-rank, sparse and quantized, resulting in < 2 kB recurrent models with accuracies similar to vanilla models for time-series classification. EMI-RNN [59] exploits the fact that only a small, tight portion of a time-series plot for a certain class contributes to the final classification (critical signature) while other portions are common among all classes (common prefixes). RNN training is thus formulated as a multiple-instance learning problem where the instance labels are iteratively updated to discover prefixes and critical signatures with provable guarantees on correctly finding the most optimal signatures. The resulting RNN is lightweight, fast ($72\times$ over vanilla RNN) and suitable for deployment on resource-constrained devices for time-series classification. ShallowRNN [60] is a hierarchical RNN architecture that divides the time-series signal into various blocks (bricks) and feeds them in parallel to several RNNs with shared weights and activation maps and the resulting feature maps are fed into a RNN in the next layer to provide output class. This improves the inference accuracy while reducing compute for time-series classification. ShallowRNN can be combined with EMI-RNN to enjoy the benefits of multiple-instance learning. Thakker et al. [61] showed the application of compressing RNN using Kronecker product, achieving 16-38$\times$ layer-wise compression.

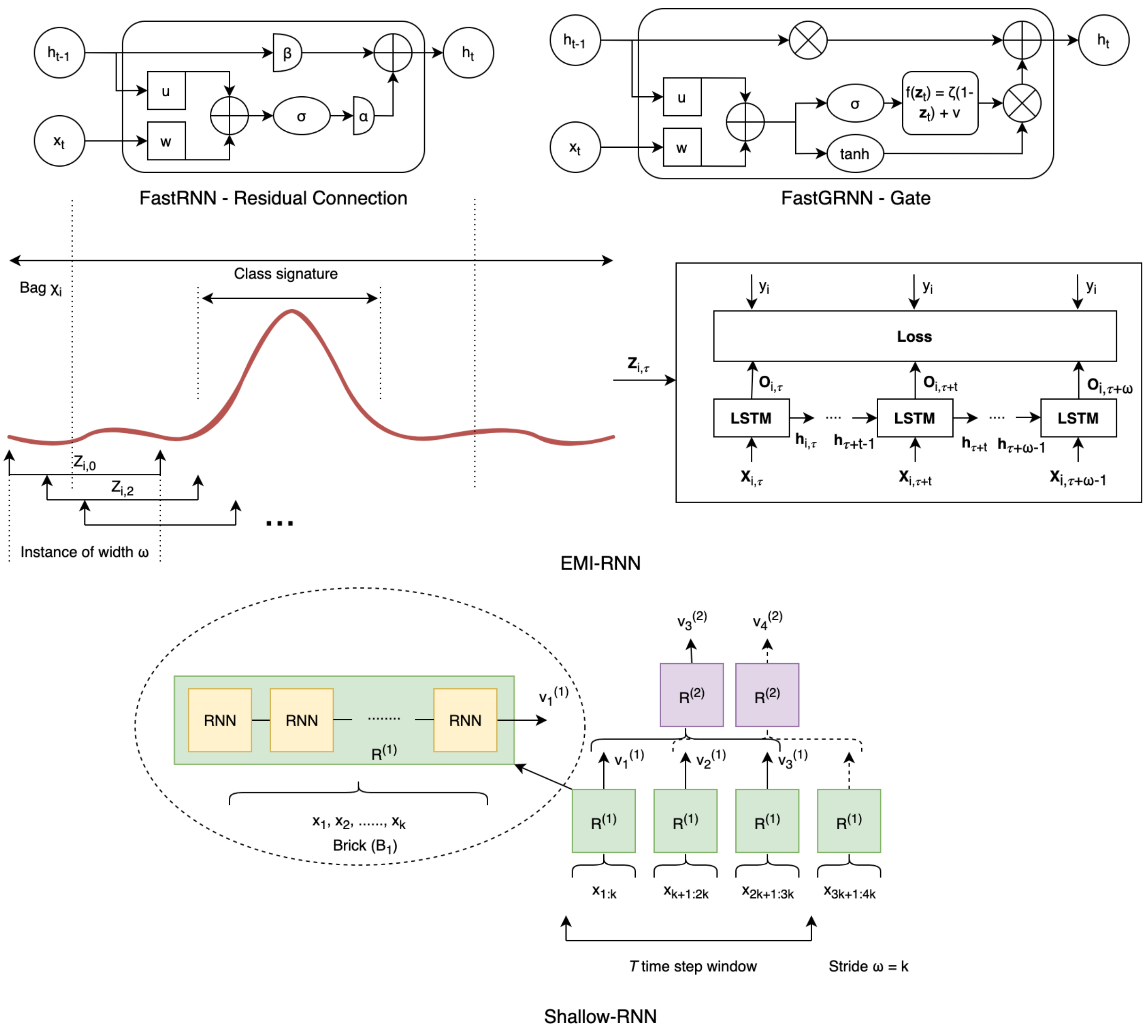


Figure 2.9: Advances in lightweight recurrent neural blocks: (a) FastRNN and FastGRNN cells [58] (b) EMI-RNN built using FastRNN cells [59] (c) Shallow-RNN built using FastRNN cells, acting as boosted decision trees [60].

2.4.2 Hardware-Aware Neural Architecture Search

NAS is the automated process of finding the most optimal neural network within a neural network search space given target architecture and network architecture constraints, achieving balance between accuracy and latency/energy [16][19][20]. The search can be performed using evolutionary algorithms or RL techniques, or the weights and architectural encodings can be jointly or separately optimized as a super-network using gradient-driven optimization or one-shot NAS [17][18]. For hardware-profiling, the device-specific measurements can be obtained in real-time by running sampled models on target device, or analytically estimated using lookup tables, architectural heuristics or prediction models. The former is slower but most accurate, while the latter is faster but least accurate [17][18]. The search space can be represented as follows [18]:

- *Layer-wise*: The entire model is generated from a collection of neural operators. The macro-architecture (e.g., number of layers and dimensions of each layer), initial and terminal layers of the network are fixed, while the remaining layers are optimized.
- *Cell-wise*: The network is constructed by stacking repeating fixed blocks called cells. A cell is a directed acyclic graph (DAG) constructed from a collection of neural operators, representing some feature transformation. Cell-based search spaces are more time-efficent compared to layer-wise approaches but are less flexible for hardware specialization.
- *Hierarchical*: In tree-based search spaces, bigger blocks encompassing specific cells are created and optimized after cell-wise optimization. Factorized hierarchical search spaces allow each layer to have different blocks without increasing the search cost, while allowing for hardware specialization.

Common hardware-aware NAS strategies include RL, gradient-based methods, evolutionary search, Bayesian optimization and hybrid techniques [17]. Figure 2.10 illustrates a generalized pipeline for hardware-aware NAS.

2.4.2.1 Reinforcement Learning

RL techniques model NAS as a Markov Decision Process (MDP). RL controllers such as proximal policy optimization (PPO), DDPG or Q-Learning are used to find the optimal combination of neural network cells from a pre-defined set recursively. Example RL NAS include NASNet and MNASNet [21][22]. The key idea is to look for optimal neural building blocks on a small dataset (e.g., CIFAR-10, called proxy task) and transfer the blocks on the target dataset (e.g., ImageNet), as directly performing NAS on large datasets is time-consuming. The network graph consists of a series of blocks (e.g., convolutional cells) whose structures are found via the controller, with the number of repetitions of each block and initial convolutional filters treated as free parameters intended for scaling based on device constraints. For similar classification tasks, the blocks are transferable. NASNet also introduced a new regularizer dubbed scheduled drop path, where a path in a cell is dropped with increasing probability as training progresses. MNASNet also involves device latency in the reward function from real mobile phones to formulate a multi-objective optimization problem to achieve balance between accuracy and inference speed. In addition, instead of limiting to two blocks and stacking them repeatedly, MNASNet uses the concept of factorized hierarchical search space. The networks found by MNASNet models are faster (2.4×), smaller, have slightly lower search cost (1.2×) and have lower number of MAC (1.7×) compared to NASNet models.

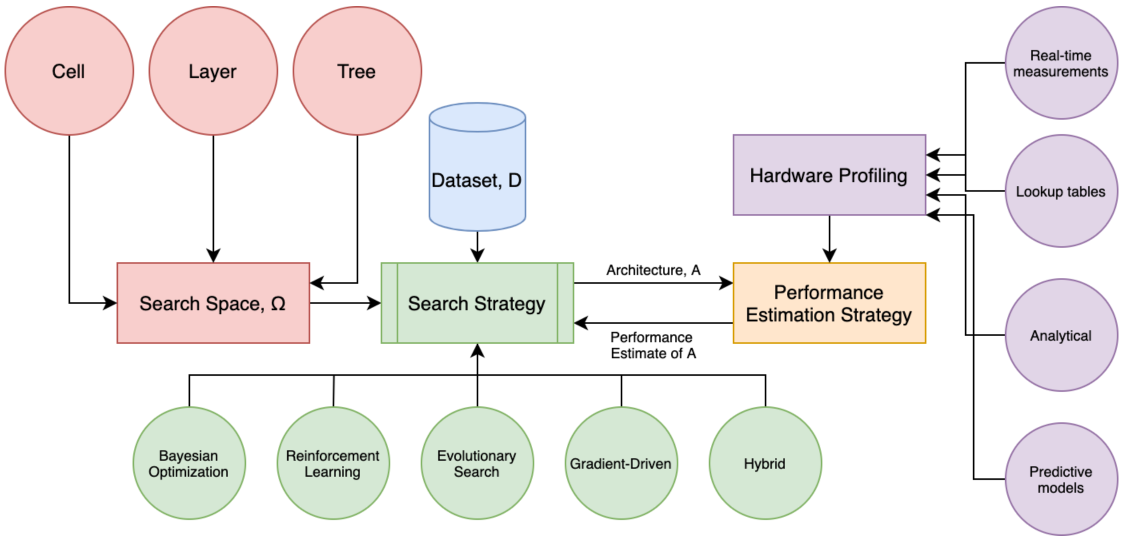


Figure 2.10: Generalized pipeline for hardware-aware NAS [17][18].

2.4.2.2 Gradient-Based Methods

Differentiable network architecture search spaces can be optimized using continuous gradient descent relaxation, which dramatically decreases the search and training cost. Gradient-based techniques include DARTS, FBNet, ProxylessNAS and MicroNets [25][26][27][28]:

- *DARTS*: In DARTS, each block in the search space is represented as a DAG, where the nodes represented activation maps and the edges represent common ML operations on the nodes. The cells can be either convolutional or recurrent and the categorical choice of operations are represented using the gradient-friendly softmax function. The goal is to learn the weights and architectural encoding as a nested bilevel optimization problem (as the optimal weights are obtained via optimization over weights and optimal architecture), with the gradients obtained approximately. The design-space of gradient-based training usually consists of the directed graph, network weights, edge operations and hyperparameters [25].

- *FBNet*: The search-space of FBNet is a layer-wise stochastic super-network (compared to static blocks in DARTS or NASNet) similar to MNASNet. It also reuses the gradient-driven architecture search from DARTS but with additional operator latency parameters from mobile phones in the loss function, thus fusing the benefits of DARTS and MNASNet. The first and last three layers of the target network are fixed while the middle layers are constructed from two point-wise convolution blocks at the end points and a depth-wise convolution (depth of 3 or 5) block in the middle, with support for skip connections and channel-shuffle for group convolutions. After training the over-parametrized network, the optimal subnetwork is obtained by sampling the architecture distribution. FBNet also ensures that the latency loss is differentiable, using Gumbel softmax to represent latency parameters [26].

- *ProxylessNAS*: Models generated by proxy-driven NAS such as NASNet may not always provide optimal performance when transferred to large datasets. On the other hand, gradient-driven NAS such as DARTS and FBNet suffer from high GPU memory consumption when the search space is too large. To solve the memory issue of differentiable NAS when training on actual datasets, ProxylessNAS uses path binarization, where the architectural parameters (real-valued path weights) of the over-parametrized network are converted to binary gates such that a single path for activation is available during the training phase. The weight parameters (sample binarized gates) and the architectural parameters (reset binarized gates, two paths used) are alternatively frozen when performing joint gradient-based training, and the final subnetwork is obtained via path-level pruning (choosing paths with the highest weights). It is also possible to use RL (policy-gradient) techniques to train the binarized weights. In addition, ProxylessNAS uses a differentiable latency model crafted from mobile phone latency data rather than using real mobile phones. ProxylessNAS yields models of eclectic configurations depending on the target architecture.

- *MicroNets*: MicroNets attempt to bring NAS to existing COTS TinyML deployment tools (e.g., TFLite Micro and CMSIS-NN) rather than using custom inference engines. The key insight behind MicroNets is the fact that both layer and model latency (as well as energy consumption) are linearly proportional to the number of FLOPS for majority of common ML operators, thus FLOPS can be a proxy for latency and energy usage for NAS. MicroNets then uses gradient-based NAS using different backbones for different tasks (e.g., MobileNetV2 for visual wake words and variants of decoder-side scalable convolutional architectures for keyword spotting and anomaly detection, with constraints on width, depth, input resolution and channel count). MicroNets then performs multi-objective optimization on accuracy, working memory and latency. The training is quantization-aware, and includes optimized 4-bit operators (sub-byte quantization) [28].

2.4.2.3 Evolutionary Search

Tailoring NAS for a diverse set of target architectures (e.g., cloud, smartphones and microcontrollers) require separate training and fining-tuning for each deployment constraints, which does not scale well due to computational expense. Cai et al. [23] suggested decoupling training from search (one-shot NAS); instead of training and fine-tuning networks for each scenario, they suggested training a single "once-for-all" (OFA) super-network consisting of several sub-networks which fits the constraints of eclectic target platforms. The training phase thus concerns improving the accuracy of all sub-networks while the search phase consists of finding a specialized sub-network using evolutionary search given target deployment constraints, reducing the computational complexity from $\mathcal{O}(N)$ to $\mathcal{O}(1)$. In evolutionary search, a fitness score is assigned to a population of network candidates contained in a single super-network. The best performing networks are selected, crossed and mutated to produce the next generation of candidate sub-networks. For efficiently training the super-network under elastic resolution, kernel size, depth and width, OFA proposes progressive shrinking, in which all of the subnetworks are jointly fine-tuned after training the full model such that the small subnetworks are gradually introduced without interfering with large subnetworks using knowledge distillation. Progressive shrinking is illustrated in Figure 2.11. To get the specialized sub-network, OFA randomly samples subnetworks from the super-network, measures their accuracy and latency on the validation set and target devices respectively and trains two MLP to predict accuracy and latency given subnetwork topography. OFA was shown to outperform RL and gradient-driven NAS strategies, achieving lower MAC, lower latency and higher accuracy for comparable subnetworks on diverse target devices, while fixing the training and search cost and achieved SOTA 80% accuracy on ImageNet for mobile devices.

MCUNet [24] tailors OFA for Class 0 devices, using a two-stage NAS (TinyNAS) to train OFA network in an optimized search space. TinyNAS optimizes the search space under elastic resolution, kernel size, expansion ratio, depth and width, with the hypothesis that models with larger FLOPS tend to have higher accuracies, selecting the search space with highest number of FLOPS. In addition, MCUNet introduces a new inference engine called TinyEngine to provide lower execution latency and memory usage compared to existing interpreter-based COTS TinyML tools such as TFLite Micro, uTensor or CMSIS-NN (more in "off-the-shelf tools" subsections). TinyEngine exploits code-generator based compilation, which reduces the runtime overhead of interpreters and compiles only the operations and variables required for deployment. It also departs from the concept of arena and instead adapts memory scheduling according to model-level statistics, which encourages feature map reuse, reduces memory fragmentation and data movement. Other features include the fusion of different operators to reduce memory footprint and improve compute, loop unrolling, loop tiling and use of in-place depth-wise convolution to reduce the amount of memory required to store output activations.

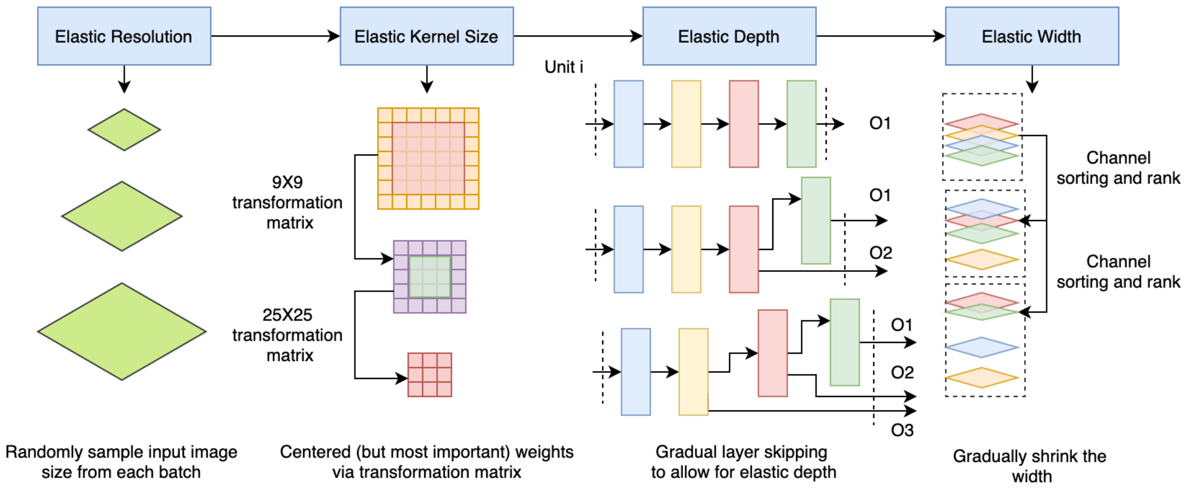

Elastic Resolution
Elastic Kernel Size
Elastic Depth
Elastic Width
Unit i
9X9 transformation matrix
25X25 transformation matrix
O1
O1
O2
O1
O2
O3
Channel sorting and rank
Channel sorting and rank
Randomly sample input image size from each batch
Centered (but most important) weights via transformation matrix
Gradual layer skipping to allow for elastic depth
Gradually shrink the width

Figure 2.11: Training process of "Once-for-All" supernetwork via progressive shrinking through knowledge distillation [23].

2.4.2.4 Bayesian Optimization

Bayesian techniques such as SpArSe [29] model the loss function as a single objective (two stage or constrained) or multi-objective (scalarization or evolutionary) optimization problem, with validation accuracy, model size and working memory optimized jointly. The search-space consists of CNNs of varying width, depth and convolution type (regular, depthwise separable, and downsampled). SpArSe also combines pruning with NAS, using both structured and unstructured pruning techniques. To restrict Pareto-optimality of candidate architectures under such diverse parameter set, constraints are put on the possible configurations as training progresses (network morphism). Treating the objective function as a Gaussian process provides SpArSe uncertainty metrics while looking for Pareto-optimal frontier. SpArSe was successfully deployed on microcontrollers with $<$ 2 kB of RAM, showing superior accuracy with fewer parameters (7.4× smaller) over competing TinyML models (Bonsai [30] and ProtoNN [31]), pruned models (LeNet + unstructured pruning and pruned decision tree

(DT)) and classical models (RBF-SVM and kNN).

2.4.3 Off-the-Shelf Tools

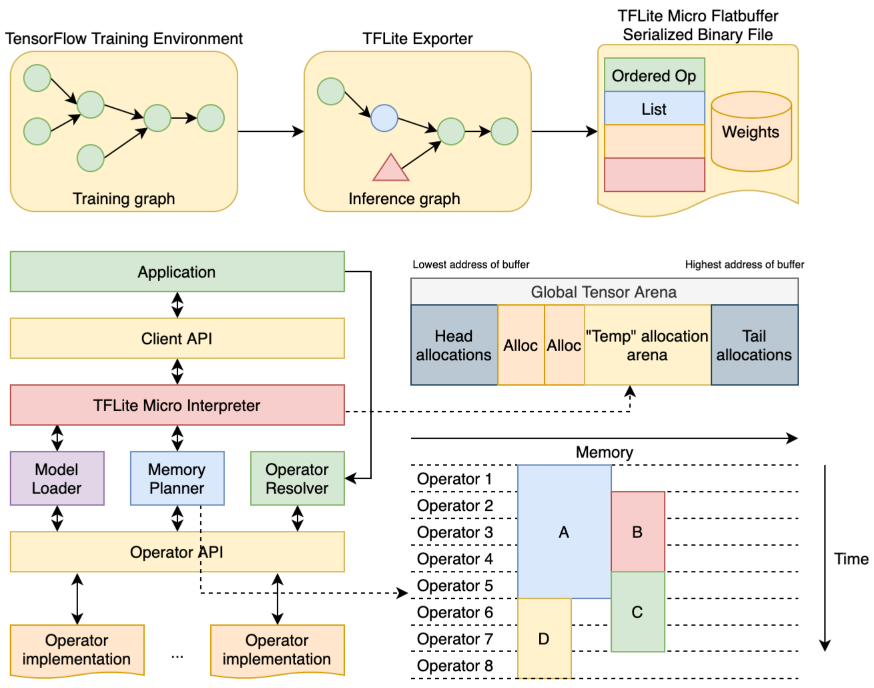


Figure 2.12: (**Top**): Converting TF model to TFLite Micro flatbuffer serialized file, suitable for importing into embedded environments (**Bottom**): The TFLite Micro runtime API and its memory allocation strategies, namely use of arena and bin packing. [62][63].

Several COTS TinyML frameworks allow conversion of ML/DL models generated by well-known libraries (e.g., PyTorch, TF, Keras, Scikit-Learn) to TinyML models via pruning and quantization, as well as providing comprehensive sets of optimized ML operators, algorithms and tools. Notable publicly available COTS TinyML packages include:

- *TFLite Micro*: TFLite Micro [62][63] is a specialized version of TFLite aimed towards optimizing TF models for Cortex-M and ESP32 MCU. TFLite Micro embraces several embedded runtime design philosophies, including dropping uncommon features, data types (e.g., float) and operations for portability, avoiding specialized library, operating system or build-system dependencies (bag of files) for heterogeneous hardware support and memory efficiency and avoiding dynamic memory allocation to mitigate memory fragmentation. TFLite Micro interprets the neural network graph at runtime rather than generating C++ code to support easy pathway for upgradability, multi-tenancy, multi-threading and model replacement, while sacrificing finite savings in memory. The implementation module is organized as follows:

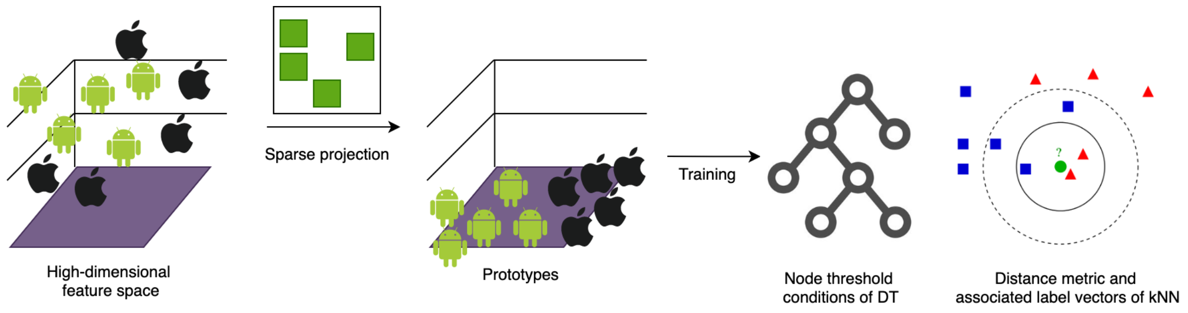
- Train a TF model and perform post-training quantization using TFLite converter.
- Use an operator resolver to link only essential operations to the model binary file.
- Pre-allocate contiguous memory (called arena) stack for initialization and storing runtime variables generated by the interpreter. TFLite Micro uses a two-stack allocation strategy to discard initialization variables after their lifetime and minimize memory consumption. The space between the two stacks is used for temporary allocations during memory planning (supports both online and offline planning), where TFLite Micro uses bin packing to encourage memory reuse and yield optimal compacted memory layouts during runtime.
- Load the model data structure. TFLite Micro uses FlatBuffer serialization format to store the models, which are represented as a schema of topologically sorted data and values for simple looping.
- Create an interpreter object with the arena, model data structure and operator resolver maps. The interpreter resolves the network graph at runtime, allocates the arena and performs runtime calculations.
- Execute the model, using a client API to handle communication between interpreter and operator schema.

Figure 2.12 shows the conversion process from TF model to TFLite Micro model, as well as the runtime API of TFLite Micro.

- *uTensor*: uTensor [64] generates C++ files from TF models for Mbed-enabled boards, aiming to generate models of < 2 kB in size. It is subdivided into two parts, namely the uTensor core, which provides a set of optimized runtime data structures and interfaces under compute constraints, and the uTensor library, which provides default error handlers, allocators, contexts, ML operations and tensors built on the core. Basic data types include integral type (8/16/32 bit integer and 32 bit float), uTensor strings (which are optimized to perform string comparisons via hashing), tensorshape (deals with shape of tensors and associated functions to transform the tensors) and quantization primitives borrowed from TFLite. Interfaces include the memory allocator interface (uses handles to allocate fixed memory like arena in TFLite micro), tensor interface (for life-cycle management and performing operations on tensors), tensor maps (ordered map of uTensor string to tensor object, a way to identify tensors) and operator interface (binds tensors to input and output names and perform evaluation). For memory allocation, uTensor uses the concept of arena borrowed from TFLite Micro, in this case, uses a circular arena that can be either bound or unbound to a handle depending on whether the user wants to avoid memory fragmentation. In addition, uTensor boasts a series of optimized (built to run CMSIS-NN under the hood), legacy (asymmetric quantized operators) and quantized (operators with symmetric quantization) ML operators consisting of activation functions, convolution operators, fully-connected layers and pooling.

- *CMSIS-NN*: Cortex Microcontroller Software Interface Standard-NN [65] was designed to transform TF, PyTorch and Caffe models for Cortex-M series MCU. It generates C++ files from the model, which can be included in the main program file and compiled. It consists of a collection of optimized neural network functions with fixed-point quantization, including fully connected layers (matrix multiplication), depth-wise separable convolution, partial image-to-column convolution (along width and height), insitu split x-y pooling and activation functions (ReLU, sigmoid and tanh, with the latter two implemented via lookup tables). It also features a collection of support functions including data type conversion and activation function tables (for sigmoid and tanh).

- *Microsoft EdgeML*: EdgeML provides a collection of lightweight ML algorithms, operators and tools aimed towards deployment on Class 0 devices, written in PyTorch and



Sparse projection
Training
High-dimensional feature space
Prototypes
Node threshold conditions of DT
Distance metric and associated label vectors of kNN

Figure 2.13: Sparse projection of high-dimensional data into prototype space, used in Bonsai and ProtoNN [30][31].

TF. Included algorithms include FastRNN / FastGRNN [58], EMI-RNN [59], ShallowRNN [60], Bonsai [30], ProtoNN [31], RNNPool [66] and DROCC [67]. Bonsai is a shallow and sparse DT ($<$ 2 kB) with non-linear activations making inferences on data projected in low-dimensional space (called prototypes), with the DT and projection variables learnt jointly under budget constraints using mini-batch stochastic gradient descent (SGD) with hard-thresholding. Similarly, ProtoNN is a lightweight KNN ($<$ 2 kB) classifier designed to operate on prototypes. The prototypes, associated label vectors and distance metric in the sparse space are jointly optimized under sparsity constraints using step-by-step minimization of the three parameters and mini-batch SGD with hard-thresholding. Figure 2.13 illustrates the idea of sparse projection, used in Bonsai and ProtoNN. RNNPool is a non-linear pooling operator that can perform "pooling" on intermediate layers of a CNN by a downsampling factor much larger than 2 (4-8$\times$) without losing accuracy, while reducing memory usage by 10x and decreasing compute by 2-5$\times$. RNNPool provides a 1$\times$1 summary for a patch of an image by sweeping the rows and columns of the patch via 4 RNN runs. This effectively downsamples the feature maps much faster, reducing the depth of the CNN. Finally, Deep robust (to mode collapse) one class classifier (DROCC) is a OCC under limited negatives and anomaly detector without requiring domain heuristics or handcrafted features. DROOC is based on the hypothesis that normal points lie on a low-dimensional linear manifold while points surrounding the normal points outside a threshold radius are outliers, which can be augmented in a generative adversarial manner into the training set.

- *STM32Cube.AI*: X-Cube-AI [68] generates STM32 compatible C code from a wide variety of DL frameworks (e.g., PyTorch, Tensorflow, Keras, Caffe, MATLAB, Microsoft Cognitive Toolkit, Lasagne and ConvnetJS). It allows quantization (minmax) of NN models, folds some of the layers into one and supports use of external flash or RAM to store activation maps or weights. The tool also features functions to measure system performance and deployment accuracy and suggests list of compatible STM32 MCU based on model complexity.

- *Eloquent MicroML and TinyML*: MicroMLgen ports DT, SVM (linear, polynomial, radial kernels or one-class), RF, XGboost, Naive Bayes, relevant vector machines (RVM), principal component analysis (PCA) transformer and SEFR (variant of SVM) from SciKit-Learn to C code, with the model entities stored on flash. TinyMLgen ports TFLite models to optimized C code using TFLite's code generator [32].

- *EmbML*: Embedded ML [69] converts LR, DT, MLP and SVM (linear, polynomial or radial kernels) models generated by Weka or Scikit-Learn to C++ code native to embedded hardware. It generates initialization variables, structures and functions for classification, storing the classifier data on flash to avoid high memory usage and supports quantization of floating-point entities.

- *FANN-on-MCU*: FANN-on-MCU [70] ports MLP models generated by fast artificial neural networks (FANN) library to Cortex-M series processors. It allows model quantization and produces an independent callable C function based on the specific instruction set of the MCU. It takes the memory of the target architecture into account and stores network parameters in either RAM or flash depending upon whichever does not overflow and is nearest to the processing unit.

CHAPTER 3

Mitigating Runtime Uncertainties in Time-Series Processing

Challenges of deploying learning-enabled inertial systems in the wild include dealing with data from multiple noisy sensors with variable sampling rates [133], misaligned time stamps [132], and missing data [1]. Traditional ML and DL approaches are unable to deal with abnormal data on their own, requiring handcrafted features and domain knowledge [1][4]. Furthermore, in context of complex events, coarse labels may be composed of a succession of simple events, whose spatial context might alter with time [3]. In this chapter, we describe intelligent data augmentation and informative missingness injection techniques to make neural inertial systems robust to runtime sensing uncertainties. We also describe a multi-stage hierarchical neural architecture tailored for complex event processing (e.g., complex activity recognition) with conditionally activated paths [15][1].

3.0.1 Handling Missing Data

Consider a multimodal sensor stream s with d channels (inputs) feeding data to a neural network $f(\cdot)$ in chunks of fixed size (called windows). Adverserial elements such as communication outages, sensor malfunction, power outages, limited memory, timing errors and sampling rate jitter can cause samples to be dropped from the channels, leading to missing data in the input stream [1][2][134]. It has been shown that missing data can hurt vanilla DL-based time-series processors, degrading medical data imputation accuracy by 18-65%, medical data classification by 2-5% and complex activity recognition by 11-24% [6][7][15].

[1]Code for this chapter is available at: https://github.com/nesl/Robust-Deep-Learning-Pipeline

To handle missing data, we use a combination of independent mask metadata channels to characterize missing samples and window alignment with contained samples popped ahead at the start of the windows during training [6][15]:

$$\mathbf{m}_t^d = \begin{cases} 1, \mathbf{s}_t^d \in \emptyset \\ 0, \text{otherwise} \end{cases} \tag{3.1}$$

$$\begin{cases} \mathbf{s}_{t:t+n}^d \to \mathbf{s}_{t:t+n-\gamma}^d \quad \forall |\mathbf{s}_{t:t+n}^d| = n - \gamma, \ \ \gamma > 0 \\ \mathbf{s}_{t+n-\gamma+1:t+n}^d = \alpha, \ \ \alpha \notin \mathbb{E}(\mathbf{s}^d) \end{cases} \tag{3.2}$$

The mask vector $\mathbf{m}_t^d$ consists of zeros at timestamps where data in channel d is missing and one at all other points. However, the masking approach may fail when the timestamps across the channels are misaligned or the actual window size is jittery. Thus, instead of aligning the sample points, the individual overlapping windows are aligned based on initial and terminal timestamps of each window, appending α for missing samples while choosing an average sampling rate for each window. The drawback in this approach is that α might be interpreted as part of the actual sensor stream. As a result, α must be chosen as a constant outside the expected value of input data. Both approaches are visualized in Figure 3.1.

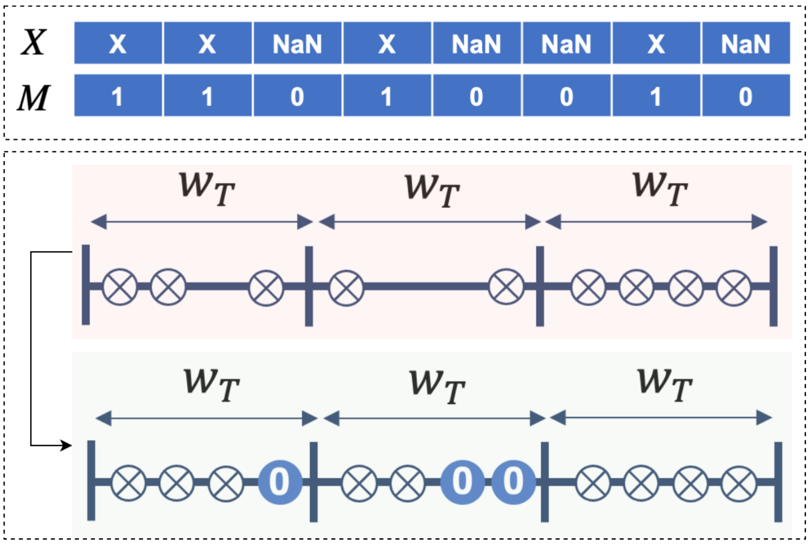


Figure 3.1: **(Top)**: Mask metadata channel M characterizing existence of missing samples in channel X. **(Bottom)**: Pushing ahead contained samples in each window of length w_T at the front and replacing the missing samples with α.

3.0.2 Window Jitter and Time-Shift Augmentation

We apply intelligent data augmentation techniques when creating sensor windows during the training phase. Our augmentations exploit the observation that timing characteristics are variable and unstable for the devices capturing data during training and deployment settings. For example, the sampling rate of smartphone peripherals can vary wildly, with the inertial sensor sampling rate varying between 40 and 100 Hz for a 100 Hz accelerometer and the microphone sampling rate oscillating between 189 KHz and 195 KHz for 192 KHz/24 bit recording [133][135]. Several factors, including delay in the operating system time stamping and variable instantaneous I/O load, are responsible for this variation. For fixed sampling rates, the general approach is to use a fixed duration for window creation. Aware of the sampling rate instability, we introduce a controlled timing jitter δ in the window length when creating windows from the training files. We hypothesize that the introduced jitter in the window length, sampled from a uniform distribution, explicitly exposes $f(\cdot)$ to the variable sampling rate, allowing them to generalize on the test data [135]:

$$|\mathbf{s}^d_{t:t+n}| \rightarrow |\mathbf{s}^d_{t:t+n}| \pm \delta, \quad \delta \sim \mathbb{U}\left(0.5(a_1+b_1), \sqrt{\frac{(b_1-a_1)^2}{12}}\right) \tag{3.3}$$

In multimodal sensing, Sandha et al. [132] have shown that data timestamps across sensors can have significant timing errors that can misalign the modalities. For example, the timing stack on Android smartphones can have as much 50000 mS of error. A 1000 mS timestamp misalignment can drop multimodal fusion classification accuracy by 6% [132]. Causes of misaligned modalities include poor management of timing stack by the operating system and the choice of time synchronization techniques used by edge devices. This can result in overfitting of $f(\cdot)$ on the timing characteristics of data, with poor generalization in the deployment scenarios [133]. To avoid this situation, we use the proposed time-shift data augmentation approach [132] of adding artificial misalignments across the device's data timestamps when creating windows. This artificial misalignment helps $f(\cdot)$ maintain accuracy in the presence of variable timing characteristics in the wild:

$$\mathbf{s}^1_{t:t+n}, \mathbf{s}^2_{t:t+n} \rightarrow \mathbf{s}^1_{t\pm\epsilon:t\pm\epsilon+n}, \mathbf{s}^2_{t\mp\epsilon:t\mp\epsilon+n}, \quad \epsilon \sim \mathbb{U}\left(0.5(a_2+b_2), \sqrt{\frac{(b_2-a_2)^2}{12}}\right) \tag{3.4}$$

Figure 3.2 illustrates how sampling rate jitter and timestamp misalignments are introduced in the training data.

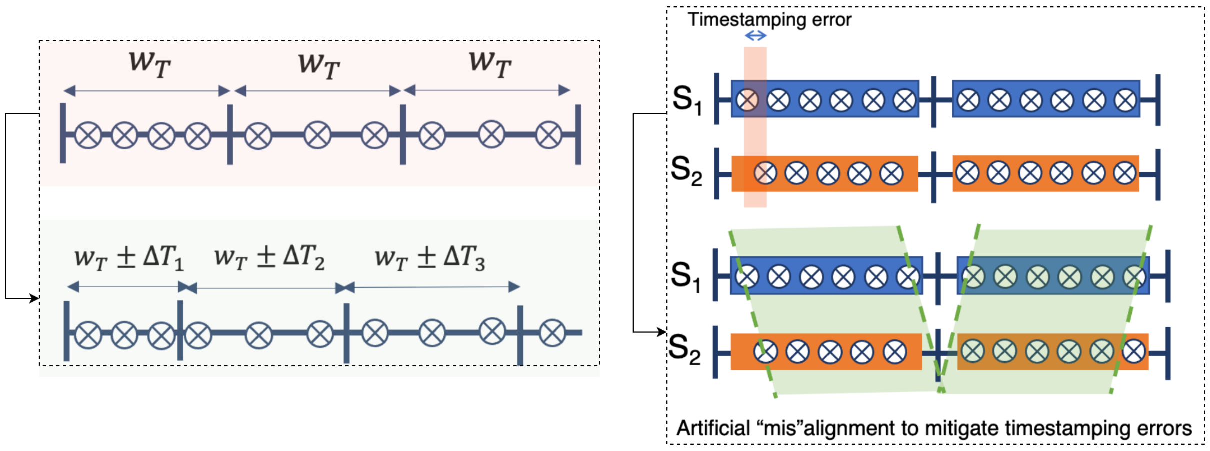


Figure 3.2: **(Left)**: Introducing artificial jitter in sampling rate **(Right)**: Adding controlled artificial timestamp misalignments across channels.

3.0.3 Hierarchical Deep Neural Architectures

Complex events can be decomposed into simpler yet granular primitives whose temporal and contextual dynamics evolve differently from the overall time-series subtleties [3]. In addition, the existence of a granular event might be dependent on the class of the coarse event, requiring logic injection in the decision making pipeline. Thus, we propose a multi-stage recurrent-convolutional neural architecture, where the latter stages are activated hierarchically based on the output of initial stages using conditional logic. The first stage consists of an ensemble of multi-label bidirectional CNN-LSTM networks with majority decision fusion for coarse event recognition, while the latter stages are composed of binary CNN-LSTM networks for granular event processing.

3.0.3.1 Convolutional Layers

CNNs are able to extract differential patterns in activity windows regardless of precise location by enforcing a degree of local connectivity among adjacent and enabling scale invariance (account for noisy feature discrepancy among same event class), making the architecture suitable as a high-dimensional feature extractor and classifier for complex time-series processing. Batch normalization layers smoothen the objective functions to account for noisy and unpredictable multimodal data. Dropout layers improve generalization performance such that the network ignores fine-grained domain-dependent event signatures, preventing overfitting [4][15][136][137][138][139][140].

3.0.3.2 Recurrent Layers

In order to infer temporal dependencies of a sequence of common integrant micro-events across several macro-events (e.g., event A may arrive after event B in a certain macro-event but event C may arrive after event B in another macro-event) and translate them differently, it is necessary to store perceived windows, which can be achieved via LSTM and RNN. LSTMs can handle variable delays between timestamps and events and learn long-term contextual dynamics of the time-series data for each sensor channel [4][15][136][137].

3.0.3.3 Deep Convolutional Bidirectional LSTM

A special class of LSTM is bidirectional LSTM, which can “look into” both the past and the future [111], increasing the amount of contextual information available to the network. Deep Convolutional Bidirectional LSTM (DCBL) is an amalgam of CNN and bidirectional LSTM, capable of extracting patterns while learning temporal dynamics of feature activations [137]. The initial layers resemble the CNN architecture except at the end, where a bidirectional LSTM layer maps the highly correlated spatial features extracted by the convolutional layers to the activity classes using recurrent information from both the past and future states [4][15][136]. The bidirectional LSTM layer is added at the end instead of the beginning to prevent long time-windows from causing difficulties in learning for the recurrent layer [111], while at the same time, ensuring that the feature map activations are highly correlated across the samples from a large event duration.

Figure 3.3 illustrates the concept of hierarchical neural architecture in context of complex activity recognition. It consists of two stages, with the first stage multilabel classifiers handling coarse macro-activities, and the second stage binary classifiers handling more fine-grained micro-activities. The classifiers in the second stage are conditionally activated based on class label from the first stage. The structure of each neural cell is illustrated in Figure 3.4, which is made up of convolutional and bidirectional LSTM layers.

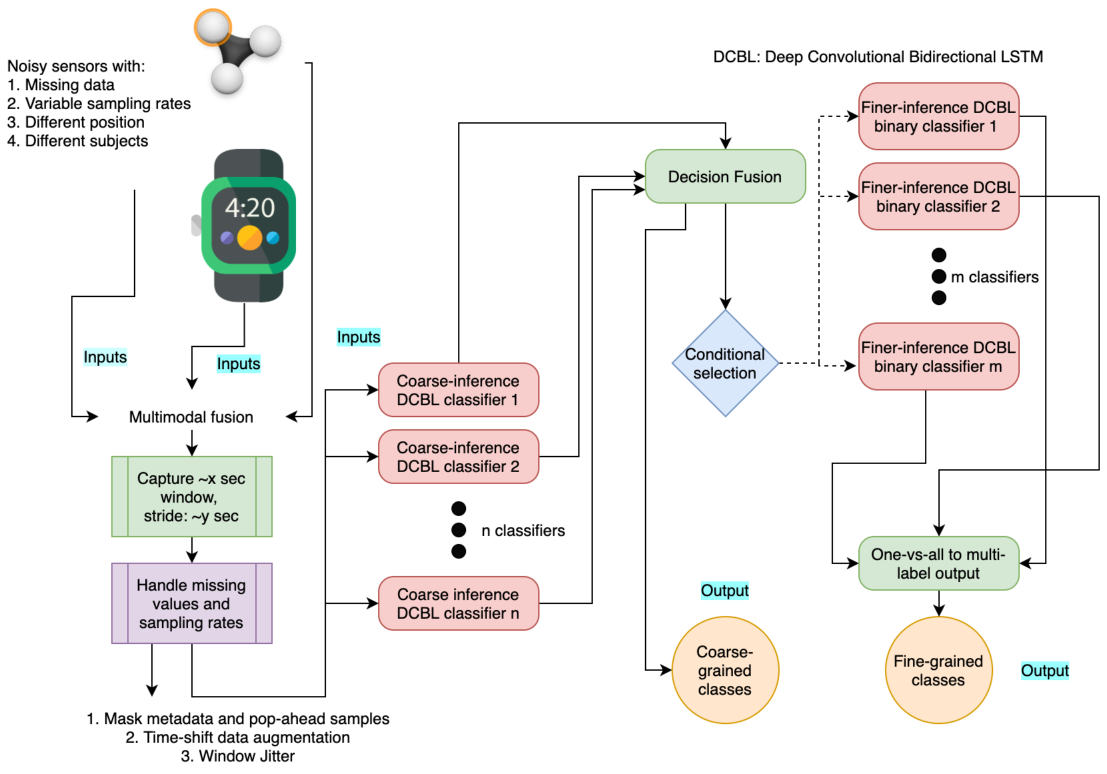

Noisy sensors with:
1. Missing data
2. Variable sampling rates
3. Different position
4. Different subjects
4:20
Inputs
Inputs
Inputs
Multimodal fusion
Capture ~x sec window, stride: ~y sec
Handle missing values and sampling rates
1. Mask metadata and pop-ahead samples
2. Time-shift data augmentation
3. Window Jitter
Coarse-inference DCBL classifier 1
Coarse-inference DCBL classifier 2
n classifiers
Coarse inference DCBL classifier n
DCBL: Deep Convolutional Bidirectional LSTM
Decision Fusion
Conditional selection
Output
Coarse-grained classes
Finer-inference DCBL binary classifier 1
Finer-inference DCBL binary classifier 2
m classifiers
Finer-inference DCBL binary classifier m
One-vs-all to multi-label output
Fine-grained classes
Output

Figure 3.3: Two-stage hierarchical, uncertainty aware, DL-based complex-event processing pipeline [15].

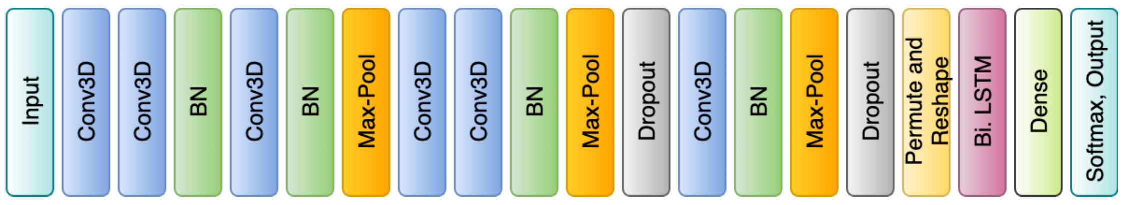


Figure 3.4: Architecture of each neural cell in the hierarchical pipeline [15].

CHAPTER 4

Efficient Neural Inertial Localization

In Chapter 2, we have seen that DL-based inertial odometry techniques provide superior translational error characteristics over classical approaches for longer time periods in GPS-denied environments, while competing with computationally intensive egomotion techniques (e.g., visual odometry, LIDAR and RADAR) [82][102][117][120][123][131]. Unfortunately, bulk of these models are unsuitable for real-time deployment on resource-constrained edge devices due to memory, power and compute constraints, binding their application in the offline world or on smartphones, cloud and workstations contrary to classical approaches, many of which have been shown to work in real-time on TinyML devices. In this chapter, we provide a systematic approach for developing neural inertial localization models for resource-constrained devices.

4.1 Inertial Sequential Learning

To break the cycle of continuous integration and error propagation, we treat dead-reckoning as a sequential-learning problem based on Newtonian physics, similar to IONet [120] and its successors. The goal is to achieve pseudo-independence across inertial windows (as windows are not truly independent) to estimate the change in navigation state in the body frame over each window rather than absolute coordinates. Under loose nonholonomic constraints, the horizontal displacement Δl and heading rate $\Delta\psi$ over a window can be expressed as a function of initial velocity $\mathbf{v}(0)$, the initial gravity direction $\mathbf{g}_0$, latent linear accelerometer $\mathbf{a}$ and gyroscope readings $\mathbf{w}$ in the polar coordinates. The task involves estimating the initial velocity and gravity in each window of size n, which are treated as latent states and instead the neural network outputs heading and displacement change, thereby constraining the ball of outputs required for the function g to model:

$$(\Delta l_t, \Delta\psi_t) = g_\theta(\mathbf{v}^b(0), \mathbf{g}_0^b, \mathbf{a}_{t:t-n}, \mathbf{w}_{t:t-n}) \tag{4.1}$$

The ground truth displacements Δl_g and heading rates $\Delta\psi_g$ for each inertial window of size n are obtained from ground truth positions L_g as follows:

$$\Delta l_{t,g} = \sqrt{(L_{x,t,g} - L_{x,t-n,g})^2 + (L_{y,t,g} - L_{y,t-n,g})^2} \tag{4.2}$$

$$\Delta\psi_{t,g} = \psi_{t,g} - \psi_{t-n,g} \tag{4.3}$$

For calculating $\Delta\psi_{t,g}$, the following conditions must hold:

$$\begin{cases} \psi_{t,g} = \arctan 2((L_{x,t,g} - L_{x,t-n,g}), (L_{y,t,g} - L_{y,t-n,g})) \\ [(\psi_{t,g} < \pi) \rightarrow (\psi_{t,g} + \pi)] \\ \phi = (\psi_{t,g} < \psi_{t-n,g}) \\ [(\Delta\psi_{t,g} > \pi/2) \vee (\Delta\psi_{t,g} < -\pi/2)] \rightarrow [(\phi \rightarrow \psi_{t,g} = \psi_{t,g} + 2\pi) \wedge (\neg\phi \rightarrow \psi_{t-n,g} = \psi_{t-n,g} + 2\pi)] \end{cases} \tag{4.4}$$

The dynamic model is given by:

$$\begin{cases} L_{x,t} = L_{x,t-1} + \Delta l_t \sin(\Theta_t) \\ L_{y,t} = L_{y,t-1} + \Delta l_t \cos(\Theta_t) \end{cases} \tag{4.5}$$

where:

$$\Theta_t = \Theta_{t-1} + \Delta\psi_t \tag{4.6}$$

The functional parameters θ of g are shared between Δl and ψ, with the ideal forms for strapdown planar motion given as [72][83][141]:

$$\psi_{\text{ideal},t} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{C}_b^N(\mathbf{q}_t)\vec{\mathbf{u}}_t^{\,b} \tag{4.7}$$

$$\mathbf{q}_t = \mathbf{q}_{t-1} + \frac{1-\alpha}{f_{s,w}}\mathbf{R}_{ib}\mathbf{w}_t + \begin{bmatrix} \arctan\left(\frac{a_{y,t}}{\sqrt{a_{x,t}^2+a_{z,t}^2}}\right)\cdot\frac{180\alpha}{\pi} \\ \arctan\left(\frac{a_{x,t}}{\sqrt{a_{y,t}^2+a_{z,t}^2}}\right)\cdot\frac{180\alpha}{\pi} \\ 0 \end{bmatrix} \tag{4.8}$$

$$\Delta l_{\text{ideal},t} = \beta\sqrt{(\mathbf{a}_t - \mathbf{g}_t)(\mathbf{a}_t - \mathbf{g}_t)^T} + \gamma \tag{4.9}$$

α, β and γ are constants, $\alpha \in (0,1)$, $\mathbf{C}_b^N$ and $\mathbf{R}_{ib}$ is the direction cosine and rotation matrices, $f_{s,w}$ is the gyroscope sampling rate, $\vec{\mathbf{u}}_t^{\,b}$ is the unit vector that is not parallel to the perpendicular axis of the ground plane and $\mathbf{q}_t$ is the attitude of the object. Hypothetically, the latent spatial and temporal connections between (Δl_{ideal}, $\Delta\psi_{\text{ideal}}$) and $(\Delta l, \Delta\psi)$ maybe be recovered from raw $\mathbf{a}$ and $\mathbf{w}$ in the presence of domain abnormalities, drift and noise using a single recurrent-convolutional neural architecture with strided velocity and heading loss [82]:

$$\mathcal{L}_f = \mathbb{E}[(\overline{\Delta l_{\text{g,t:t-n}}} - \Delta l_{\text{t:t-n}})^2] + \kappa\mathbb{E}[(\overline{\Delta\psi_{\text{g,t:t-n}}} - \Delta\psi_{\text{t:t-n}})^2] \tag{4.10}$$

In order to handle cross-modal and multi-modal timestamp misalignment and sampling rate variability, we expose g_θ to controlled artificial time shifts and window jitter augmentation in inertial windows during training phase [15]. To handle missing data, we use a combination of independent mask metadata channels to characterize missing samples and window alignment with contained samples popped ahead at the start of the inertial windows [15]. In addition, we apply controlled rotation $\mathbf{R}$ (coordinate frame normalization) and multivariate Gaussian noise to each 3D inertial channel during training phase to inform g_θ of unrestricted sensor orientation and varying sensor physics [82][114]:

$$\mathbf{s}_{t:t+n}^{x,y,z} \rightarrow \mathbf{R}\mathbf{s}_{t:t+n}^{x,y,z} + \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}) \tag{4.11}$$

4.2 Temporal Convolutional Network

We use a TCN [142][143] to model g [102][103][125][131], which can jointly handle spatial and temporal features hierarchically. Without explosion of parameter, memory footprint, layer count or overfitting, TCN kernels allow the network discover global context in long inertial sequences while maintaining input resolution and coverage. In TCN, the convolution operation has three desirable properties:

- *Causality*: The output of the operator at current timestep t depends only on the current and past inputs but not future inputs. This ensures temporal ordering of the input sequences without requiring recurrent connections. Contrary to the usual method of processing the input sequences as affine transformations of past values on a per-step feed-forward basis, causal convolutions allow parallel training on long sequences as the ordering is maintained via weight sharing among the input chunks. Causal convolutions are also known as Conv1D. convolving only on current and past elements from earlier layers, as illustrated in Figure 4.1 (Right).

- *Dilation*: Conv1D requires a large receptive field to integrate global context in the sequence, which can be achieved efficiently with fewer layers and parameter count using dilated convolutions. The receptive field F_i of each unit in the ith layer in a TCN dilated causal kernel of size $k \times k$ with dilation factor l is given by:

$$F_{i,\text{TCN}} = F_{i-1} + (k_l - 1) \times l, F_0 = 1 \tag{4.12}$$

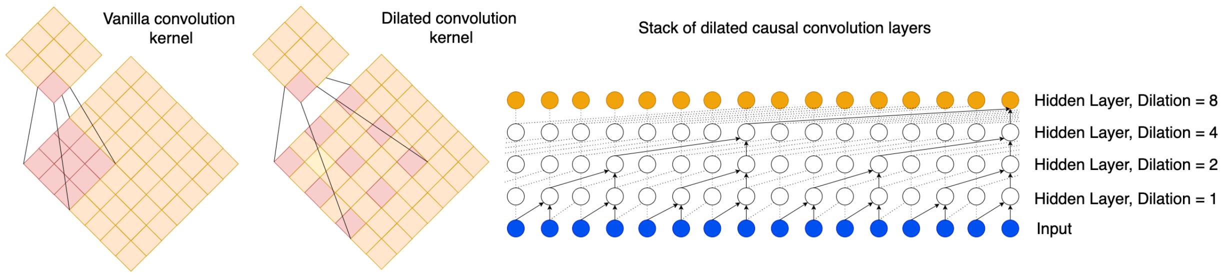
  $F_{i,\text{TCN}}$ is larger than $F_{i,\text{CNN}}$, which is $i \times (k-1) + k$. When dilated CNN are stacked on top of each other, the dilation factor increases exponentially, increasing model capacity and receptive field size. Each residual block usually has two dilated convolution operations. Assuming fixed kernel size, the receptive field of a TCN with n residual blocks and an expansion factor of 2 is given by:

$$F_{\text{TCN}_n} = 1 + 2 \times (k-1) \times (2^n - 1) \tag{4.13}$$

  Figure 4.1 (Left) shows the difference in the receptive field between a vanila 3×3 kernel and a dilated 3×3 kernel.

- *Residual Blocks:* Two stacks of dilated causal convolution layers are fused through gated residual blocks $\mathbf{z}$ (shown in Figure 4.2) for expressive yet bounded non-linearity, complex interactions and temporal correlation modeling in the input sequence:

$$\mathbf{z} = \tanh(\mathbf{W}_{f,k} * \mathbf{x}) \odot \sigma(\mathbf{W}_{g,k} * \mathbf{x}) \tag{4.14}$$



Vanilla convolution kernel
Dilated convolution kernel
Stack of dilated causal convolution layers
Hidden Layer, Dilation = 8
Hidden Layer, Dilation = 4
Hidden Layer, Dilation = 2
Hidden Layer, Dilation = 1
Input

Figure 4.1: (**Left**): Vanilla convolution filters vs. dilated convolution kernels (dilation factor = 2) (**Right**): Stack of dilated causal convolution layers [143].

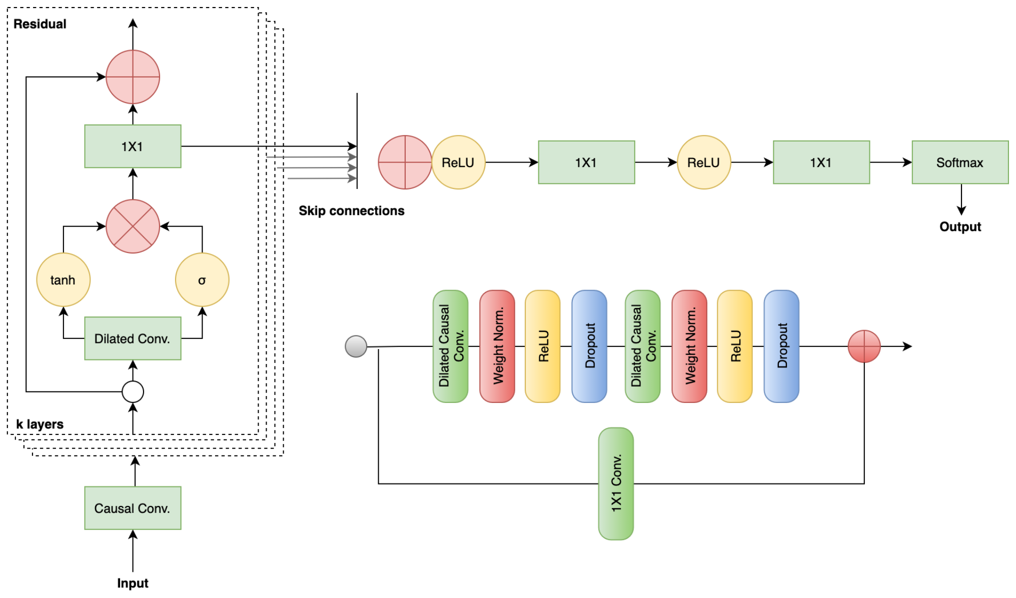


Figure 4.2: Residual block in the TCN architecture [143][153].

TCN have the following advantages over vanilla neural architectures in inertial DR:

- Small number of parameters (10,000× compared to 1,000,000×), which is less prone to overfitting while keeping a low-memory footprint.
- Fewer complex non-linear operations compared to recurrent architectures, keeping a low runtime latency.
- Maintains temporal ordering without requiring recurrent units, which are computationally expensive and difficult to train and optimize for TinyML devices due to vanishing

gradient (leads to unstable training) for long time-series sequences.

- Supports parallelization due to out-of-order training, while recurrent units need sequential information to be fed in-order.
- Dilated convolutions allow larger receptive fields (exponential expansion) without explosion of parameter or layer count compared to LSTM or CNN, while maintaining input resolution and coverage, allowing the network to discover semantic connections in long sequences of inertial samples.
- Gated activation units replace ReLU in vanilla CNN for expressive non-linearity and temporal correlation modeling, while competing with larger candidate neural architectures in terms of absolute translation error.
- Jointly handles spatial and temporal features in a hierarchical fashion without requiring two-stage convolutional and recurrent pipeline using causal convolution (Conv1D).

4.3 Orientation and Altitude Estimation

For tracking depth or height using barometric sensors, we designed a model-free $\alpha - \beta$ filter [144] with thermocline, salinity, noise and timestamp jitter mitigation [145][1]. The $\alpha - \beta$ filter is a simplified version of a KF. The altitude measurements $L_{z,m}$ at timestep t are given by:

$$L_{z,t,m} = \underbrace{\left| -\frac{R(T_{c,t}+273.15)}{Mg} \ln\left(\frac{P_{t,\mathrm{m}}}{P_0}\right)\right|}_{\text{air}} \vee \underbrace{\left|\frac{P_{t,\mathrm{m}}}{\rho_0 g}\left(1-\frac{P_{t,\mathrm{m}}}{K}\right)\right|}_{\text{fluid}} \tag{4.15}$$

where, $P_{t,\mathrm{m}}$ = pressure measurement, M = air molar mass, g = gravitational acceleration, R = gas constant, $T_{c,t}$ = temperature in Celsius and P_0 = average sea level pressure (kPa). Furthermore:

$$\rho_0 = D(T_{c,t}) + sA(T_{c,t}) + s^{1.5}B(T_{c,t}) + cs^2 \tag{4.16}$$

[1]Code for the filter is available at: https://github.com/nesl/robust_depth_filter

$$K = E(T_{c,t}, s) + F(T_{c,t}, s)P_{t,\mathrm{m}} + G(T_{c,t}, s)P_{t,\mathrm{m}}^2 \tag{4.17}$$

where, $N(T_{c,t}) \leftarrow \gamma 10^j T_{c,t}^k$, $M(T_{c,t}, s) \leftarrow \mu 10^l T_{c,t}^n s^q + N(T_{c,t})$, s = salinity of fluid and μ, γ, j, k, l, n and q are constants. The filter prediction steps are given by:

$$L_{z,t,p} = L_{z,t-1,p} + \Delta T \dot{L}_{z,t-1,p}, \quad \dot{L}_{z,t,p} = \dot{L}_{z,t-1,p} \tag{4.18}$$

The update steps are given by:

$$\dot{L}_{z,t,p} = \dot{L}_{z,t,p} + \frac{\beta}{\Delta T}(L_{z,t,m} - L_{z,t,p}) \tag{4.19}$$

$$\underbrace{L_{z,t,p}}_{\text{altitude}} = L_{z,t,p} + \alpha(L_{z,t,m} - L_{z,t,p}) \tag{4.20}$$

$\dot{L}_{z,t,p}$ refers to the velocity of the object, ΔT is the difference between current and previous timestamps and (α, β) are filter coefficients. Large values of α favor measurements over prediction, while large values of β increases the transient sensitivity of the filter.

For real-time orientation estimation (only under strapdown scenario) in the world frame, ${}_W\mathbf{q}_{est}$ (in quaternion domain) we use the 9-DoF Madgwick filter [146]:

$$\underbrace{{}^I_W\mathbf{q}_{est,t+1}}_{\text{attitude}} = {}^I_W\hat{\mathbf{q}}_{est,t} + \Delta T(\underbrace{{}^I_W\dot{\mathbf{q}}_{\mathbf{w},t+1}}_{\text{gyro. inc.}} + \underbrace{{}^I_W\dot{\mathbf{q}}_{\nabla,t+1}}_{\text{acc. inc.}}) \tag{4.21}$$

where,

$${}^I_W\dot{\mathbf{q}}_{\mathbf{w},t+1} = 0.5\, {}^I_W\hat{\mathbf{q}}_{est,t} \otimes [0, {}^I\mathbf{w}_{t+1}]^T \tag{4.22}$$

$${}^I_W\dot{\mathbf{q}}_{\nabla,t+1} = -\beta \frac{\nabla f\left({}^I_W\hat{\mathbf{q}}_{est,t}, {}^W\hat{\mathbf{g}}, {}^I\hat{\mathbf{a}}_t, {}^W\hat{\mathbf{b}}_{t+1}, {}^I\hat{\mathbf{m}}_t\right)}{\left\|f\left({}^I_W\hat{\mathbf{q}}_{est,t}, {}^W\hat{\mathbf{g}}, {}^I\hat{\mathbf{a}}_t, {}^W\hat{\mathbf{b}}_{t+1}, {}^I\hat{\mathbf{m}}_t\right)\right\|} \tag{4.23}$$

${}^W\hat{\mathbf{g}}$ denotes the normalized gravity vector in the world frame, ${}^W\hat{\mathbf{b}}$ refers to the local magnetic declination and ${}^I\hat{\mathbf{m}}$ denotes the magnetic field strength in the sensor frame. The attitude consists of gyroscope increment (numerical integration of angular velocity measurements) and the acceleration increment found via gradient descent on f, which is given by:

$$f\left({}^I_W\hat{\mathbf{q}}, {}^W\hat{\mathbf{d}}, {}^I\hat{\mathbf{s}}\right) = {}^I_W\hat{\mathbf{q}}^* \otimes {}^W\hat{\mathbf{d}} \otimes {}^I_W\hat{\mathbf{q}} - {}^I\hat{\mathbf{s}} \tag{4.24}$$

$$\nabla f\left({}^{I}_{W}\hat{\mathbf{q}}, {}^{W}\hat{\mathbf{d}}, {}^{I}\hat{\mathbf{s}}\right) = \mathbf{J}^{T}({}^{I}_{W}\hat{\mathbf{q}}, {}^{W}\hat{\mathbf{d}}) f\left({}^{I}_{W}\hat{\mathbf{q}}, {}^{W}\hat{\mathbf{d}}, {}^{I}\hat{\mathbf{s}}\right) \tag{4.25}$$

where, ${}^{W}\hat{\mathbf{d}}$ denotes the field reference direction in the world frame (e.g., gravity for accelerometer), ${}^{I}\hat{\mathbf{s}}$ denotes the sensor measurements and $\mathbf{J}$ is the Jacobian of function f. For ${}^{W}\hat{\mathbf{d}} = {}^{W}\hat{\mathbf{g}}$ and ${}^{I}\hat{\mathbf{s}} = {}^{I}\hat{\boldsymbol{a}}$:

$$f_g\left({}^{I}_{W}\hat{\boldsymbol{q}}, {}^{I}\hat{\boldsymbol{a}}\right) = \begin{bmatrix} 2\left(q_2q_4 - q_1q_3\right) - a_x \\ 2\left(q_1q_2 + q_3q_4\right) - a_y \\ 2\left(\frac{1}{2} - q_2^2 - q_3^2\right) - a_z \end{bmatrix}, \mathbf{J}_g\left({}^{I}_{W}\hat{\boldsymbol{q}}\right) = \begin{bmatrix} -2q_3 & 2q_4 & -2q_1 & 2q_2 \\ 2q_2 & 2q_1 & 2q_4 & 2q_3 \\ 0 & -4q_2 & -4q_3 & 0 \end{bmatrix} \tag{4.26}$$

For the magnetometer components:

$$f_b\left({}^{I}_{W}\hat{\boldsymbol{q}}, {}^{W}\hat{\boldsymbol{b}}, {}^{I}\hat{\boldsymbol{m}}\right) = \begin{bmatrix} 2b_x\left(0.5 - q_3^2 - q_4^2\right) + 2b_z\left(q_2q_4 - q_1q_3\right) - m_x \\ 2b_x\left(q_2q_3 - q_1q_4\right) + 2b_z\left(q_1q_2 + q_3q_4\right) - m_y \\ 2b_x\left(q_1q_3 + q_2q_4\right) + 2b_z\left(0.5 - q_2^2 - q_3^2\right) - m_z \end{bmatrix} \tag{4.27}$$

$$\boldsymbol{J}_b\left({}_{W}S\hat{\boldsymbol{q}}, {}^{W}\hat{\boldsymbol{b}}\right) = \begin{bmatrix} -2b_zq_3 & 2b_zq_4 & -4b_xq_3 - 2b_zq_1 & -4b_xq_4 + 2b_zq_2 \\ -2b_xq_4 + 2b_zq_2 & 2b_xq_3 + 2b_zq_1 & 2b_xq_2 + 2b_zq_4 & -2b_xq_1 + 2b_zq_3 \\ 2b_xq_3 & 2b_xq_4 - 4b_zq_2 & 2b_xq_1 - 4b_zq_3 & 2b_xq_2 \end{bmatrix} \tag{4.28}$$

Thus, $f\left({}^{I}_{W}\hat{\mathbf{q}}, {}^{W}\hat{\mathbf{g}}, {}^{I}\hat{\mathbf{a}}, {}^{W}\hat{\mathbf{b}}, {}^{I}\hat{\mathbf{m}}\right)$ and its Jacobian are given by:

$$f\left({}^{I}_{W}\hat{\mathbf{q}}, {}^{W}\hat{\mathbf{g}}, {}^{I}\hat{\mathbf{a}}, {}^{W}\hat{\mathbf{b}}, {}^{I}\hat{\mathbf{m}}\right) = \begin{bmatrix} f_g\left({}^{I}_{W}\hat{\mathbf{q}}, {}^{I}\hat{\boldsymbol{a}}\right) \\ f_b\left({}^{I}_{W}\hat{\mathbf{q}}, {}^{W}\hat{\mathbf{b}}, {}^{I}\hat{\boldsymbol{m}}\right) \end{bmatrix} \tag{4.29}$$

$$\boldsymbol{J}_{g,b}\left({}^{I}_{W}\hat{\boldsymbol{q}}, {}^{I}\hat{\boldsymbol{b}}\right) = \begin{bmatrix} \boldsymbol{J}_g^T({}^{I}_{W}\hat{\boldsymbol{q}}) \\ \boldsymbol{J}_b^T\left({}^{I}_{W}\hat{\boldsymbol{q}}, {}^{W}\hat{\boldsymbol{b}}\right) \end{bmatrix} \tag{4.30}$$

The Madgwick filter provides several advantages for real-time orientation estimation over counterpart [146]:

- Formulates attitude estimation in the quaternion domain, devoid of the singularities and gimbal lock in Euler domain.

- Computationally inexpensive; 109 and 248 scalar operations per update in C code.
- Effective for low-sampling rate sensors.
- Converges in one gradient descent step; the optimization formulation ensures counteraction of the effects of translational movement (e.g., linear acceleration) and obtains the most likely orientation candidate.
- Includes magnetic distortion compensation and IMU calibration steps.
- Single tunable parameter β for simple and intuitive deployment.

4.4 Hardware and Quantization-Aware Bayesian NAS

Most SOTA NAS techniques use evolutionary search or reinforcement-learning to find optimal neural architectures, with the weights and architectural encodings optimized separately via gradient-driven or one-shot NAS as a "once-for-all" supernetwork, resulting in networks configurations that provide the best performance during deployment under diverse target constraints but take thousands of GPU hours to evaluate [23]. To find the ideal TinyML neural inertial candidate from the backbone TCN for limited flash, RAM and latency requirements, we model the search as a parallelizable black-box Bayesian optimization problem (BO) with fewer elastic parameters or hard constraints. The search space Ω consists of neural network weights w, hyperparameters θ, network structure denoted as a directed acyclic graph (DAG) g with edges E and vertices V representing activation maps and common ML operations v (e.g., convolution, batch normalization, pooling etc.) respectively, which act on V.

$$\begin{cases} f_{\text{opt}} = \lambda_1 f_{\text{accuracy}}(\Omega) + \lambda_2 f_{\text{flash}}(\Omega) + \lambda_3 f_{\text{SRAM}}(\Omega) + \lambda_4 f_{\text{latency}}(\Omega) \\ f_{\text{accuracy}}(\Omega) = \mathcal{L}_{\text{val}}(\Omega), \Omega = \{\{V, E\}, w, \theta, v\} \\ f_{\text{flash}}(\Omega) = -\max(0, (||w||_0 + \alpha) - \text{flash}_{\max}) \\ f_{\text{SRAM}}(\Omega) = -\max(0, \max_{l\in[1,L]}\{||x_l||_0 + ||a_l||_0) + \beta\} - \text{SRAM}_{\max}) \\ f_{\text{latency}}(\Omega) = \text{OPS} \vee \text{FLOPS} \end{cases} \tag{4.31}$$

where,

$$a = w \vee y, \quad y = \sum_{k=1}^{K} v_k g_k(x, w_k), \;\; |z|_0 = 1 \tag{4.32}$$

The objective function f_{opt} can be thought of as seeking a Pareto-optimal configuration of parameters Ω^* under competing objectives [29] such that:

$$f_k(\Omega^*) <= f_k(\Omega) \;\; \forall k, \Omega \;\; \wedge \exists j : f_j(\Omega^*) < f_j(\Omega) \;\; \forall \Omega \neq \Omega^* \tag{4.33}$$

Validation loss serves as a proxy for the accuracy of the model [29][147], while the total number of weights serves as a proxy for the total flash required for storing the neural network weights (unstructured pruning) [28][29]. COTS tools such as TFLite Micro stores network weights, quantization parameters and network graph on flash, while using the arena on SRAM to store intermediate activation maps and tensors, persistent buffer and TFLite Micro runtime interpreter parameters [29]. As a result, we use either the standard RAM usage model or assume that the outputs are stored on flash while the layerwise inputs and weights reside in the arena for optimizing peak working memory. Since, model latency is linearly proportional to the OPS count for a variety of convolutional models, we use FLOPS or OPS as a proxy for runtime latency [28]. In addition, we inject layerwise quantization simulation encodings based on weight and activation distribution in the training phase using "Enhanced Tensorflow" scheme for quantization-aware training (QAT) [43]:

$$w_{i,l} = S w_{f,l} + \left(\frac{\delta_{\text{min}}}{S}\right), \quad S = \frac{\delta_{\text{min}} - \delta_{\text{max}}}{2^{\text{BW}} - 1} \tag{4.34}$$

where, δ represents clamping thresholds and BW represents the target bitwidth. QAT mitigates layer-wise quantization error across channels due to variance of weights for small models.[2]

We use Gaussian process $\mathcal{GP}$ as the surrogate model to approximate f_{opt}, which allows priors on the distribution of moments to propagate forward as the search progresses. In addition, the domain of random scalarizations λ can be specified by the user to guide the parallel search acquisition functions (hallucination or K-means clustering) into the promising

[2] https://github.com/quic/aimet

Pareto-optimal regions of the gradient plane. The acquisition function decides the next set of Ω_n to sample from the design space using Monte Carlo sampling with Bayesian Upper-Confidence Bounds (UCB), also known Thompson sampling, which balances exploration and exploitation [147]. Apart from speeding up the NAS, parallel search ensures that NAS is not being performed on network morphs early on (exploitation) and information gain is maximized in the search process (exploration), yielding a stage-wise "coarse-to-fine" search space:

$$\hat{f}(\Omega) \sim \mathcal{GP}(\mu(\Omega), k(\Omega, \Omega')) \tag{4.35}$$

$$\Omega_t = \arg\max_{\Omega}(\mu_{t-1}(\Omega) + \beta^{0.5}\sigma_{t-1}(\Omega)) \tag{4.36}$$

CHAPTER 5

Datasets, Hardware and Implementation

In this chapter, we outline implementation specifics of robust and efficient neural inertial localization and complex activity recognition. Specifically, we benchmark our uncertainty aware complex event pipeline (Chapter 3) on the *Cooking Activity Dataset with Macro and Micro Activities* as part of the 2020 Cooking Activity Recognition Challenge [154][155]. We perform preliminary evaluation of our efficient neural inertial localization framework (Chapter 4) on the *Oxford Inertial Odometry Dataset* (OxIOD) [131], and provide a compact, ultra-low-power, environmentally resilient and modular inertial odometry hardware configuration that pushes the state-of-the-art in inertial odometry hardware. Lastly, we test the performance of Madgwick filter on the *NESL Earable Mobility Corpus*[1], a large-scale head-pose and simple activity detection dataset developed by ourselves using earable sensors.

5.1 The Cooking Activity Recognition Challenge

The 2020 Cooking Activity Recognition Challenge [154] embraces the hurdles of uncertainty and abnormality in sensor-based complex activity detection in the wild, with the goal of building a classifier to classify 3 distinct macro and 10 distinct micro-activities as follows:

- Making a sandwich - cut, wash, take, put, other
- Preparing fruitsalad - cut, take, peel, add, mix, put, other
- Preparing cereal - cut, take, pour, peel, put, open, other

[1]Dataset available at: https://github.com/swapnilsayansaha/Wearable_Acoustic_AR

The training dataset consists of 30-second windows from 3 subjects in 288 data frames/files. Sensors include 4 triaxial accelerometers placed at left wrist, right wrist, right arm, and left hip and 29 triaxial motion capture (mo-cap) markers placed at random locations of the body, totaling 99 sensor channels. Each frame corresponds to a single macro-activity, with one-to-multiple possible micro-activities. Mo-cap samples are captured at approximately 100 Hz, while the accelerometers' sampling rates vary between 50Hz - 100Hz. The starting and terminal timestamps for each micro-activity are absent. The test dataset contains unlabeled data from a 4th subject. The count for each class label is shown in Figure 5.1.

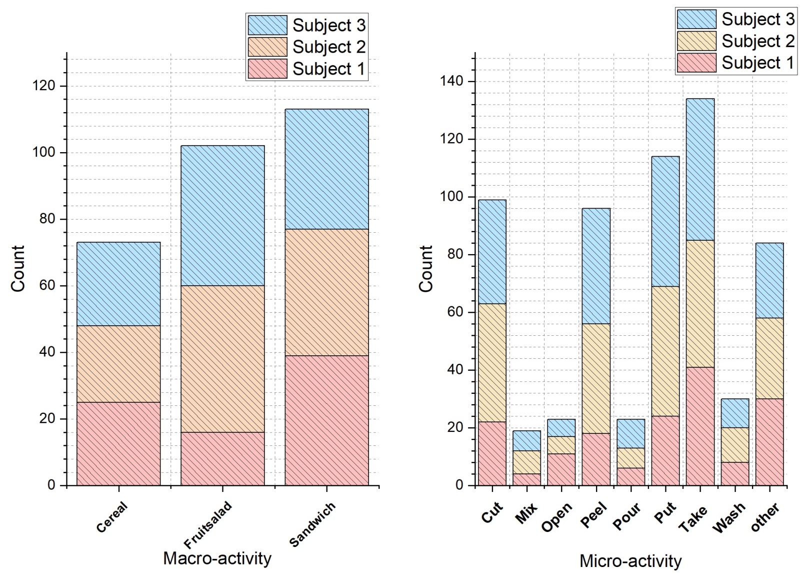


Figure 5.1: Distribution of subject-wise class labels in the Cooking Activity Dataset.

Fig. 5.2 shows the plots for 12 accelerometer channels for one of the 288 training files, indicating missing samples and sampling rate jitter. For our experiments, we chose to omit the 3 left-wrist channels due to high frequency of wrongly annotated timestamps and missing data in all files. Furthermore, mo-cap supplies absolute position rather than motion signatures unique to each action primitive. In our experiments, the motion capture data did not improve the validation accuracy. Thus, we trained final models using 9 accelerometer channels (right wrist, right arm, and left hip).

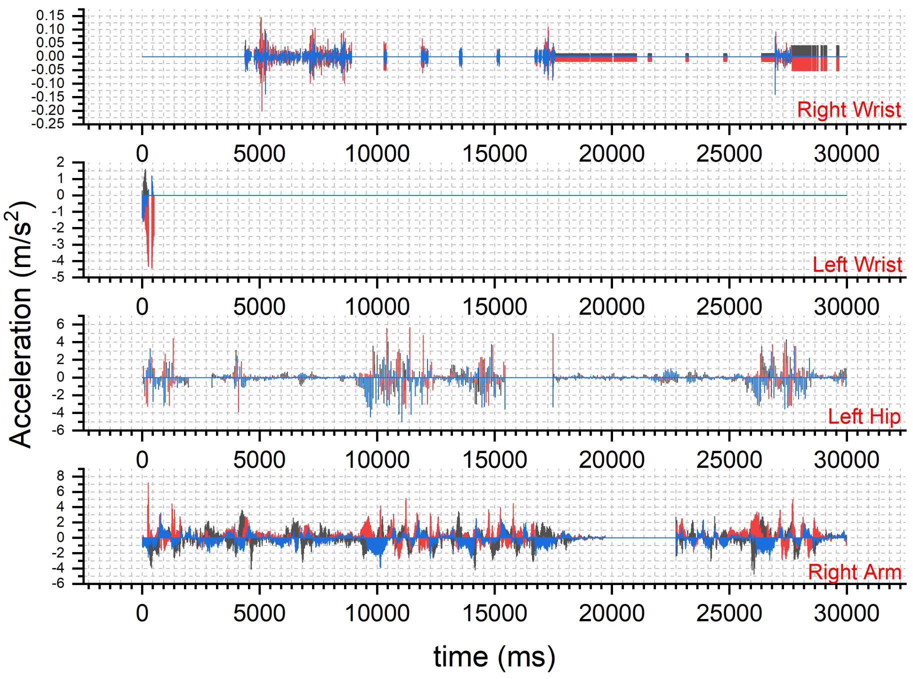


Figure 5.2: Accelerometer plots for one of the 288 frames, indicating the presence of missing data and variable sampling rates.

For classification, the files in the training set were split into 60:20:20 (train:validation:test) ratio randomly for both macro and micro-activity classification. The dataset was split by files (sessions/trials).

5.1.1 Model Implementation Specifics

We use an ensemble of 10 DCBL classifiers with majority voting decision fusion to classify macro-activities and conditionally select some of 20 micro-activity DCBL binary classifiers (one-vs-all, ensemble of 2) based on the macro-activity label. In addition, for tuning the window size, we tested the performance of single CNN, LSTM and DCBL for macro and micro-activity classification. Table 5.1, 5.2 and 5.3 illustrates the architecture of each model. The right wrist, right arm, and left hip accelerometers were used in training the final models. Sliding overlapping window of length 10 second (other candidates included 3 and 6 seconds) was used, with a stride of 1 second. No feature extraction or post-processing techniques were applied to the final models.

The models were implemented in Jupyter notebook (Python), using Keras and Sklearn via a Tensorflow backend, with MATLAB being used for minor errands such as generating labels and splitting training set. All models were trained on a GPU-machine with 128 GB RAM, 2x 12 GB Nvidia GeForce GTX 1080 Ti, and 3.4GHz AMD Ryzen Threadripper 1950X 16-core CPU. The total training time for the final 10 macroactivity classifiers was approximately 5 hours (including data preprocessing and loading). 40 epochs were used per classifier. The training time for the 20 microactivity binary classifiers was approximately 1 hour and 30 mins, with 10 epochs being used per classifier. The total time required to operate on the test set was approximately 8 minutes.

Table 5.1: Sample architecture of implemented CNN.*

Layer (type)	**Param #**	**Size**	**Stride**	**Activation**	**Output Shape**
Conv 1	1408	(1, 10)	(1, 1)	ReLu	(12, 500, 128)
Conv 2	163968	(1, 10)	(1, 1)	ReLu	(12, 500, 128)
B. Norm 1	512				(12,500,128)
Conv 3	163968	(1, 10)	(1, 1)	ReLu	(12, 500, 128)
B. Norm 2	512				(12, 500, 128)
Max. pool 1		(1,3)	(1,1)		(12, 166, 128)
Conv 4	163968	(1, 10)	(1, 1)	ReLu	(12, 166, 128)
Conv 5	163968	(1, 10)	(1, 1)	ReLu	(12, 166, 128)
B. Norm 3	512				(12, 166, 128)
Max. pool 2		(1,2)	(1,1)		(12, 83, 128)
Dropout 1					(12, 83, 128)
Conv 6	163968	(1, 10)	(1, 1)	ReLu	(12, 83, 128)
B. Norm 4	512				(12, 83, 128)
Max. pool 3		(1,2)	(1,1)		(12, 41, 128)
Dropout 2					(12, 41, 128)
Flatten 1					(62976)
Dense 1	3022912			ReLu	(64)
B. Norm 4	256				(64)
Dropout 3					(64)
Dense 2	195			Softmax	(3)

* Output shape and parameter numbers are different for micro-activity classifiers

Table 5.2: Sample architecture of implemented LSTM.*

Layer (type)	**Param #**	**Size**	**Stride**	**Activation**	**Output Shape**
LSTM 1	70656			tanh	(500, 128)
Dropout 1					(500,128)
LSTM 2	131584			tanh	(128)
Dropout 2					(128)
Dense 2	387			Softmax	(3)

* Output shape and parameter numbers are different for micro-activity classifiers

Table 5.3: Sample architecture of implemented DCBL.*

Layer (type)	**Param #**	**Size**	**Stride**	**Activation**	**Output Shape**
Conv 1	1408	(1, 10)	(1, 1)	ReLu	(12, 500, 128)
Conv 2	163968	(1, 10)	(1, 1)	ReLu	(12, 500, 128)
B. Norm 1	512				(12,500,128)
Conv 3	163968	(1, 10)	(1, 1)	ReLu	(12, 500, 128)
B. Norm 2	512				(12, 500, 128)
Max. pool 1		(1,3)	(1,1)		(12, 166, 128)
Conv 4	163968	(1, 10)	(1, 1)	ReLu	(12, 166, 128)
Conv 5	163968	(1, 10)	(1, 1)	ReLu	(12, 166, 128)
B. Norm 3	512				(12, 166, 128)
Max. pool 2		(1,2)	(1,1)		(12, 83, 128)
Dropout 1					(12, 83, 128)
Conv 6	163968	(1, 10)	(1, 1)	ReLu	(12, 83, 128)
B. Norm 4	512				(12, 83, 128)
Max. pool 3		(1,2)	(1,1)		(12, 41, 128)
Dropout 2					(12, 41, 128)
Permute 1					(41,12,128)
Reshape 1					(41,1536)
Bi. LSTM	1704960				(256)
Dense	771			Softmax	(3)

* Output shape and parameter numbers are different for micro-activity classifiers

5.1.2 Baselines and Variations

For uncertainty-aware complex activity recognition, we had the following baselines:

- *Vanilla DNN*: We tested the performance of the three multi-label classifiers shown in

Table 5.1, 5.2 and 5.3 without any uncertainty injection.

- *DNN with normalization and interpolation*: Vanilla DNN with classical uncertainty pre-processing techniques such as bandpass filtering (using 20 Hz butterworth low-pass filter), normalization (z-normalization, uni-variance, and min-max scaling) and vanilla interpolation (linear, spline, autoregression).

- *DNN with mo-cap data*: Multi-label classifiers trained on motion capture data instead of IMU signals.

- *Hierarchical DNN with time-shift augmentation and window jitter*: We add controlled artificial shifts and sampling rate jitter to the training data. To handle missing data, we follow the push ahead contained samples (no masking). In addition, we start using the hierarchical DNN architecture discussed in Chapter 3.

- *Hierarchical DNN with time-shift augmentation, window jitter and masking*: Same as previous, but with mask metadata channel included.

- *Top 10 competitors*: We also compare the performance of our approach with the top 10 (out of 78) competitors in the competition. Models include CNN, CNN-LSTM, DC-GRU, LightGBM, Naive-Bayes, Graph CNN, kNN and multi-sampling classifiers [154]. Details of each approach can be found in [156].

5.2 The OxIOD Dataset

There are several large-scale annotated inertial odometry datasets in eclectic domains, with notable ones being:

- Human localization: TUM-VI [157], RIDI [75], RoNIN [82] and OxIOD [131].

- Vehicle trajectory estimation: KITTI [158], KAIST Urban [159], nuScenes [160], Berkeley DeepDrive [161] and PandaSet [162].

- UAV flight patterns: EuRoC MAV [163].

- Underwater AUV motion: AquaLoc [164].

As a starting point for preliminary benchmarks, we chose the Oxford Inertial Odometry Dataset (OxIOD) [131], which is the second largest inertial navigation dataset aimed towards human localization right behind the RoNIN dataset [82]. Features of the dataset include:

- 14.7 hours of 9DoF indoor inertial sequences collected via smartphone IMU and annotated via Vicon motion tracker system (provides ground truth position and orientation of the phone in real-time), totaling a distance of 42.6 km from 5 users and across 2 floors (1650 and 2475 meter squared).
- 0.5 mm ground truth resolution, which is better than any other inertial navigation dataset.
- Various sensor configurations - smartphone in the hand, in the pocket, in the bag and on a trolley.
- Diverse walking patterns - slow walking (0.5 m/s), normally walking (1 m/s) and running (1.5 m/s).
- 4 common inertial sensor models found in commodity smartphones (iPhone 5, 6 and 7 plus and Nexus 5).
- UTC time-synchronized ground truth and inertial sequences available to account for time-stamp misalignment.
- Separate file-wise training and test splits.

For initial evaluation, we chose 6 inertial channels across 6 different deployment scenarios, namely 'handheld', 'handbag', 'pocket', 'trolley', 'slow-walking' and 'running'. The 6 inertial channels include tri-axial accelerometer streams (minus gravity) and tri-axial gyroscope streams recorded at 100 Hz. The initial frame length was 200 samples, with a stride of 10 samples (sliding window), which improves the output rate of the neural network. For the ground truth polar vector, we extracted the average velocity and heading over these

2-second frames from the 3D ground truth pose, assuming loose nonholonomic constraints on the vertical motion patterns of test subjects. Figure 5.3 shows sample trajectory from one of the training files in the OxIOD dataset, along with sample accelerometer and gyroscope traces. Figure 5.4 shows the sample extracted displacement, heading rate and absolute heading traces for the same file. The heading rate changes much more slowly and predictably compared to the absolute heading, and is easier to model using DNN. By extracting headings and displacements from raw positions, we constrain the possible outputs required for the DNN to learn within finite bounds.

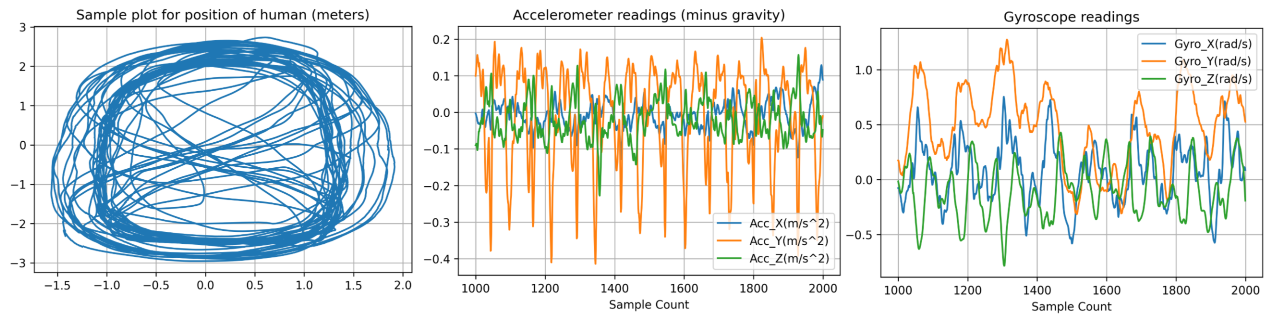


Figure 5.3: Sample trajectory, accelerometer and gyroscope traces from one of the training files in the OxIOD dataset.

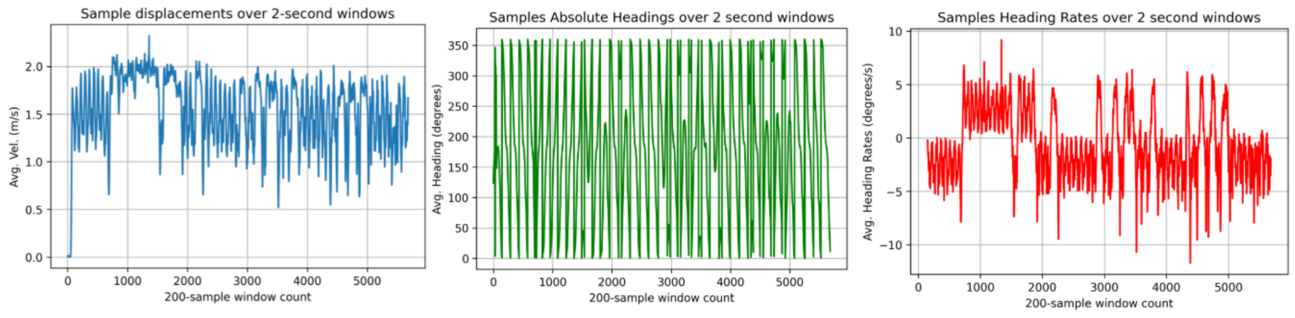


Figure 5.4: Sample displacement, absolute heading and heading rates from one of the training files in the OxIOD dataset.

We split the training set with 49 sequences into 34 training files and 15 validation files (we did not touch the test files during training), totaling 164514 2-second 6-channel inertial sequences for training and 62392 sequences for validation. For quantifying the performance of TCN, we train the TCN using displacement and heading rates. However, due to technical issues, we used the absolute heading and displacement for evaluation in the NAS algorithm.

5.2.1 Inertial Localization Model Settings

Inspired from L-IONet [131], our initial backbone network consisted of a 8-layer TCN with a dilation factor of 2, filter size of 32 and batch size of 256. We used Keras TCN library to implement the network via a Tensorflow backend. We tested separate models for heading rate and displacement, before ultimately switching to a single model for both heading and displacement regression. We used Adam optimizer with a learning rate of $1e^{-5}$, using mean-squared error loss between predicted and ground-truth elements (strided velocity and heading loss). Initially, all the models were trained on a GPU-machine with 128 GB RAM, 2×12 GB Nvidia GeForce GTX 1080 Ti, and 3.4 GHz AMD Ryzen Threadripper 1950X 16-core CPU for 1000 epochs. The neural architecture for joint optimization is shown in Figure 5.5.

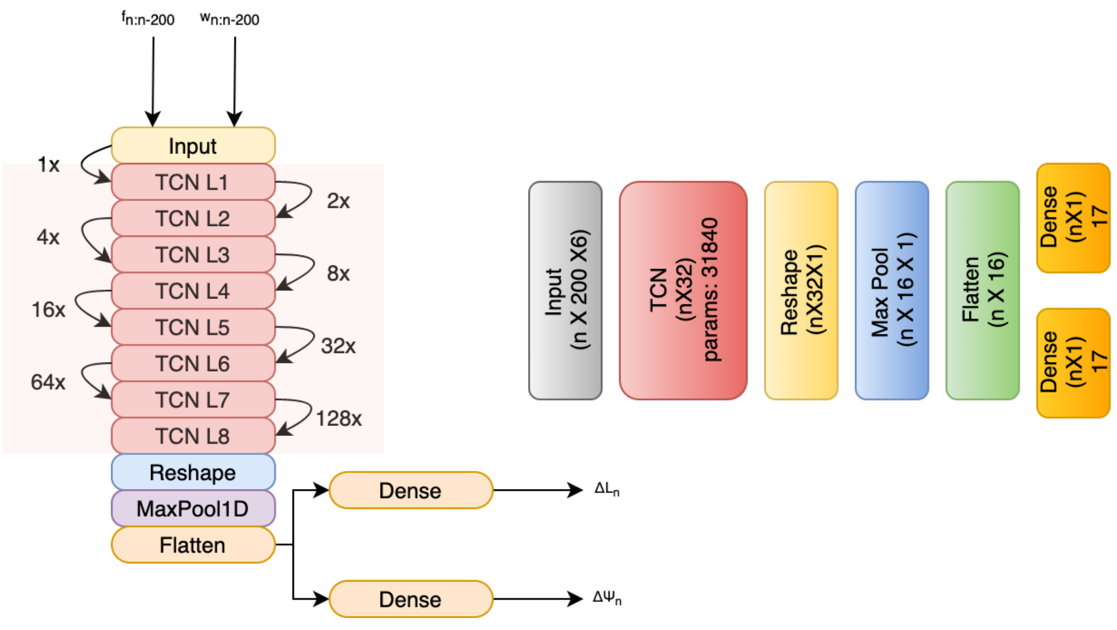


Figure 5.5: Sample backbone TCN architecture for dead-reckoning, with shapes and parameter count of each layer.

5.2.2 NAS Parameters

To implement Bayesian NAS, we use the Mango parallel hyperparameter tuning library [147]. Mango is a multi-armed bandit Bayesian optimizer, using Gaussian Process as surrogate function and Bayesian UCB to guide the search in the NAS design space. Mango also supports parallel search acquisition functions using either hallucination or K-means clustering through distributed computing frameworks such as Celery, Kubernetes or Horovod. We used 10 epochs in the Bayesian optimization (except one case where we used 100 iterations) with a batch size of 1. For each candidate model, we trained for 50 epochs. The search space for our NAS (colored red in Figure 5.5) was as follows, containing $\sim 10^{14}$ neural networks:

- Number of filters: 2 to 64 with a stride of 1
- Kernel size: 2 to 16 with a stride of 1
- Dropout rate: 0 to 0.5 with a stride of 0.1
- Skip connection usage: True or False
- Normalization: Batch normalization, Layer normalization and Weight normalization
- Dilations: Combination of 3 to 8 layer TCNs with dilation factors chosen from 1, 2, 4, 8, 16, 32, 64, 128, 256

5.2.3 Loading Models on Target Hardware

To load the TF models onto embedded hardware, we use TFLite Micro [62][63]. We convert the model to TFLite object and then convert the TFLite object to portable C code as header files. We then import the header files into Mbed Studio to include in the application. The steps to include and run the model in code are as follows:

- Initialize header files and declare TFLite Micro namespace.
- Initialize the target, set up debug log and map the model to usable data structure.

- Declare the operator resolver and build an interpreter.
- Allocate memory from arena to model's tensors and obtain pointers for model's input and output buffer.
- During runtime, pass data to the model via the input pointer.
- Invoke the interpreter and get inference from model via output pointer.

5.2.4 Baselines and Variations

For preliminary evaluation of our efficient neural inertial localization algorithm, we chose the following baselines:

- *TinyEKF* [165]: EKF without error correction heuristics shown in Figure 2.1, but geared towards embedded implementation.
- *TinyEKF with GPS*: TinyEKF, but with GPS for error correction.
- *IONet-LSTM* [120]: The first end-to-end DL-based neural inertial navigation model, where sequential learning was introduced.
- *RoNIN-TCN* [82]: One of the successors of IONet, which allows unrestricted sensor orientations and positions.
- *L-IONet* [131]: Lightest neural inertial navigation model geared towards real-time smartphone implementation. The base network that we start with is structurally similar to L-IONet.

We also qualitatively compare the performance of our NAS algorithm with MCUNet, Once-for-All and MicroNets [24][23][28], which are SOTA among existing NAS algorithms.

5.3 Hardware for Neural Inertial Localization

To support long-term, low-power, durable and inexpensive neural inertial localization at the edge, we designed one of the smallest, lightest, lowest-power and highest-resolution inertial hardware framework with active communications, capable of running inertial odometry algorithms at the edge that pushes the SOTA in odometry hardware. The fabricated circuit board is illustrated in Figure 5.6.

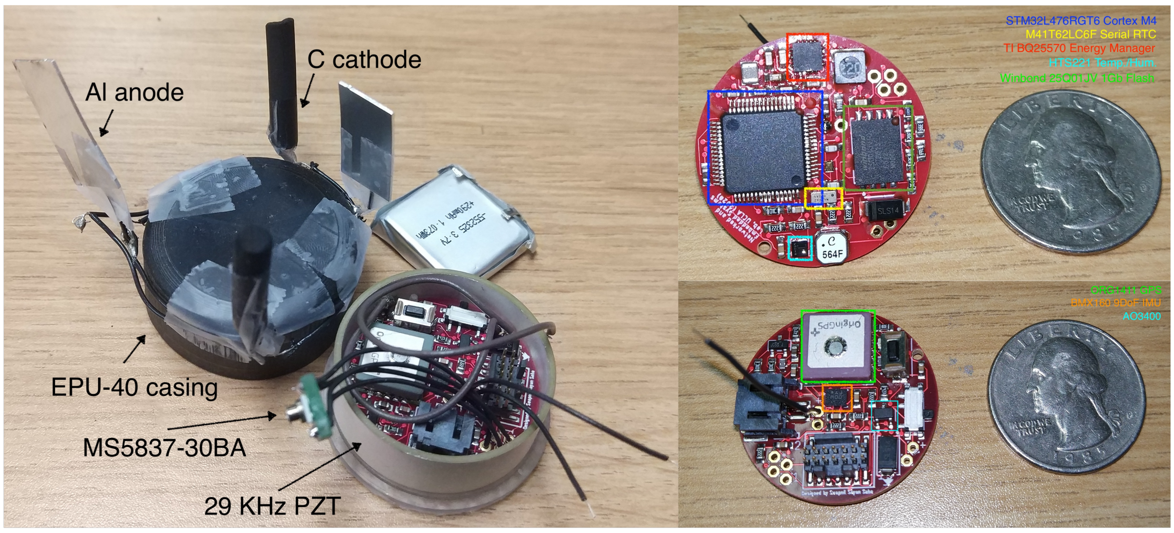


Figure 5.6: Proposed dead-reckoning hardware.

We use an ultra-low-power STM32L476RG ARM Cortex M4 microcontroller (128 kB RAM, 1 MB flash) with built-in floating point unit running Mbed RTOS. For the inertial sensor, we use a Bosch BMX-160 9DoF MEMS IMU [169] with built-in temperature compensation, power-management unit and cross-channel time synchronization. The board also features a MS5837-30BA underwater pressure sensor for altitude estimation, an ORG-1411 GPS module for occasional position correction when GPS is available and a HTS221 temperature/humidity chip. For time-stamping, we added a M41T62 (with embedded crystal) real-time clock (RTC). The board is capable of harvesting energy depending on deployment environment, e.g., for the underwater version, we added an Al-air energy-harvesting system (BQ25570 for energy management) that can generate $\sim$ 4-5 mW of power from oceanic salt water. The tag supports LoRa or BLE for over-the-air communication, and uses 29 KHz ultrasonic piezoacoustic communication in both active and backscatter [170][171] mode for underwater communication. The entire board consumes 330 mW of power maximum. For storage, we added 128 MB of serial flash. The tag measures 32 × 32 × 10 mm, and is suitable for widespread deployment without incurring noticeable size, weight or power overhead.

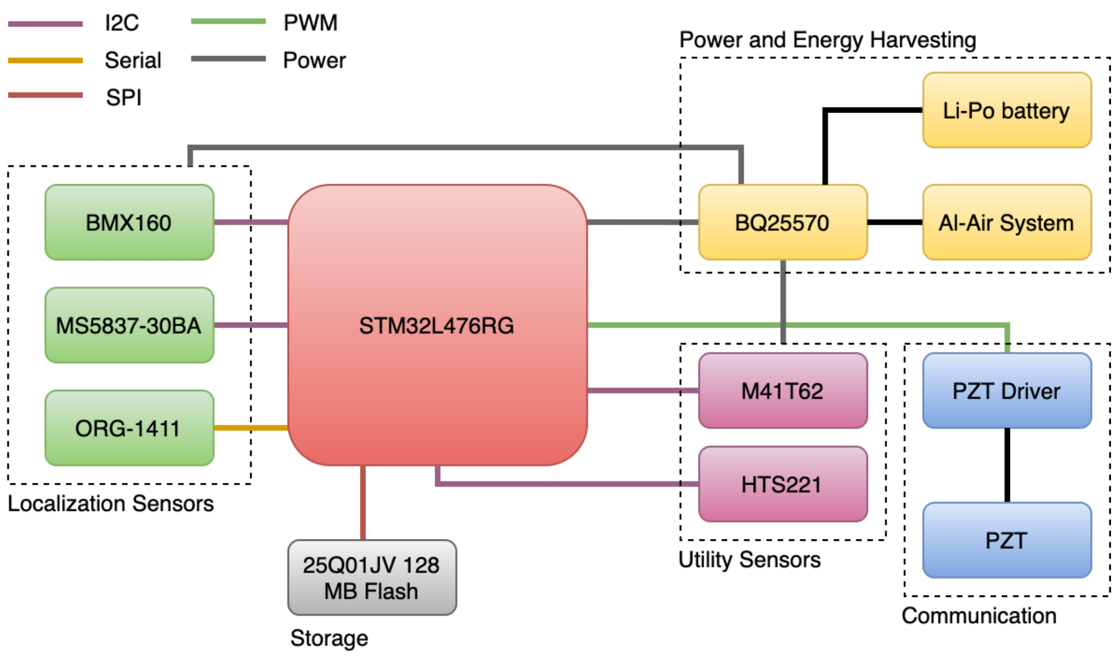


Figure 5.7: System diagram of the sensor tag configuration.

5.3.1 Mechanical Encasement and Circuit Board Specifications

We designed a waterproof 3D-printed base and cap for the sensor tag, made out of photopolymer resin, shown in Figure 5.8. The encasement around the transducer was 3D-printed using Carbon Elastomeric Polyurethane Resin (EPU-40) to allow the transducer vibrate freely during communication (high elasticity), while still allowing for a tear-resistant surface protecting the electronics of the tag against the elements. EPU-40 provides better acoustic impedance over generic polyurethane encasement [170] commonly used for underwater backscatter networking, while being easier to prototype in an inexpensive manner. In addition, the transducer sits on a rubber washer to reduce friction between the base and the vibrating transducer [170]. To seal the tag, we use waterproof removable caulk, which allows the adhesive to be peeled off for easy access to the electronics and battery during programming and maintenance.

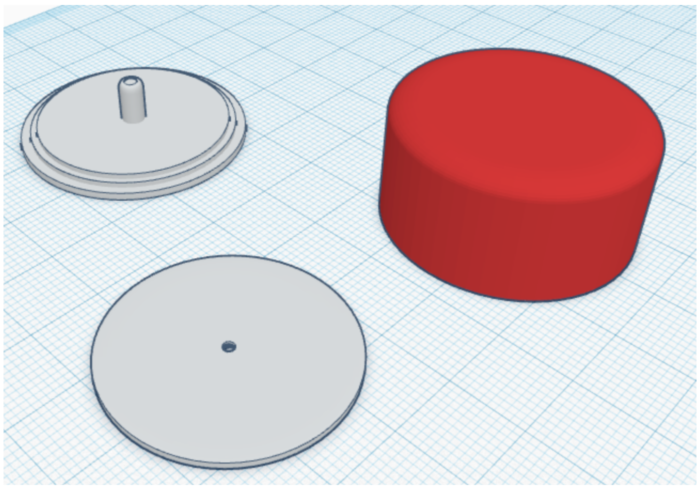


Figure 5.8: Encasement for the proposed hardware, showing the bases in gray and encasement in red.

The circular printed circuit board (PCB) is mounted inside the cylindrical piezoacoustic transducer (PZT). The PCB has a radius of 14.5 mm and is the 2nd densest inertial PCB after the ST SensorTile [172], which itself is underpowered to run odometry algorithms on-board. The PCB is a 4-layer 2-sided high-density-interconnect PCB with lead-free gold contacts for reducing environmental toxicity footprint. The cost of each PCB is $\sim$ 100 USD.

5.3.2 Energy Harvesting

For underwater energy harvesting, we use a miniature Al-air battery system [173][174]. We use Aluminum (Al) strips as anodes and activated charcoal as cathode. The following reactions occur in the presence of salt-water:

$$\text{Anode:} \quad Al \rightarrow Al^{3+} + 3e^{-} \tag{5.1}$$

$$\text{Cathode:} \quad O_2 + 2H_2O + 4e^{-} \rightarrow 4OH^{-} \tag{5.2}$$

The reaction requires flow of ocean water to occur, which contains 3.5% Sodium Chloride. We use a ring of electrodes around the encasement (shown in Figure 5.6). We managed to generate ∼4-5 mW of power with the electrode configuration shown in Figure 5.6, achieving an open-circuit voltage of 1.1V and short circuit current of 40 mA. The generated power is well over the power consumption of the tag in sleep mode and is suitable for charging the included Li-Po battery. To harvest such tiny amounts of power, we use the BQ25570 energy harvester chip [175]. The chip can harvest uW to mW of power while consuming power in the order of nW, with a minimum input voltage of just 100 mV required to start harvesting. The chip also features a boost battery charger with overcharge/undervoltage protection and a buck regulator to supply system power rail from the battery, which in our case is 3.3V. It is also possible to use other forms of energy harvesting with this chip, such as solar and thermoelectric generators.

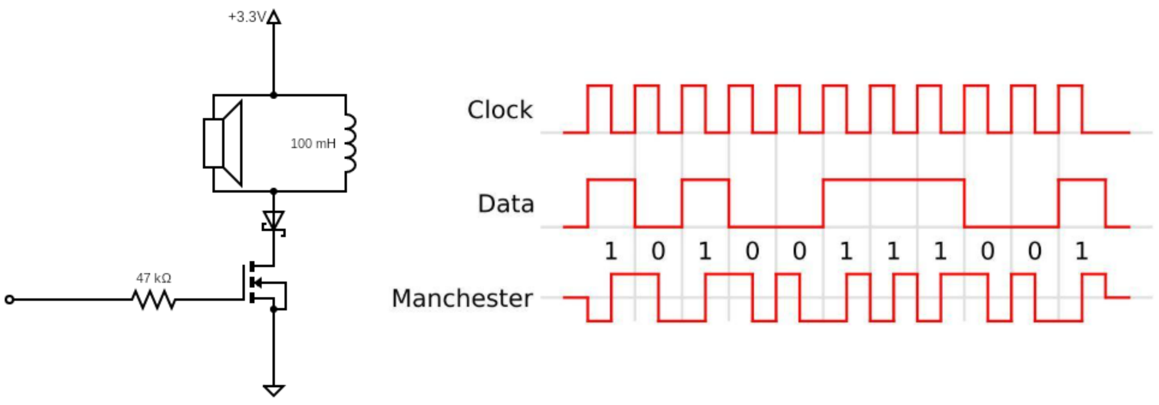


Figure 5.9: **(Left)** : PZT driver circuit in active mode. The speaker symbol denotes the PZT. The PWM signal is fed through resistor. **(Right)**: Manchester encoding.

5.3.3 Communication

For underwater communication, we use an ultrasonic PZT tuned to 29 KHz resonant frequency. In active mode, we use a MOSFET driven LC tank circuit controlled by the pulse width modulation (PWM) pins of the Cortex M4. The communication uses Binary Phase Shift Keying (BPSK) with Manchester encoding. The signals can be picked up using a hydrophone. In addition, the PZT supports batteryless backscatter communication and energy-harvesting [170][171] with range in the order of tens of meters. In active mode, the PZT supports range in the order of hundreds of meters. Figure 5.9 shows the PZT driver circuit. The extracted impedance curve for the PZT is shown in Figure 5.10. For over-the-air communication, we also provide provision to use LoRa or BLE [176].

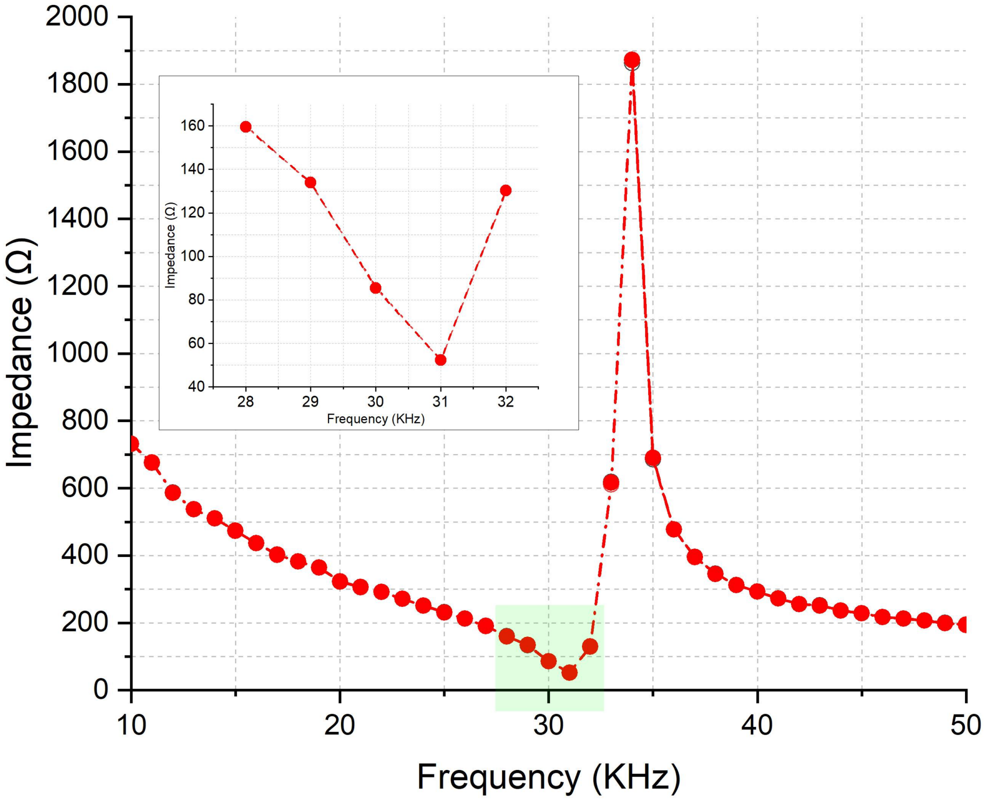


Figure 5.10: Extracted impedance plot for the 29 KHz PZT, showing a resonant frequency around 31 KHz.

5.3.4 Specifications of Localization Sensors

5.3.4.1 IMU

We used the Bosch BMX-160 IMU in our design, which is one of the smallest footprint, lowest power, lowest noise profile and highest resolution inertial sensors in the market [169]. The IMU has built-in temperature compensation, power-management unit and cross-channel time synchronization. The specifications of the sensor are as follows:

- Current consumption: Gyroscope: 850 uA, Accelerometer: 180 uA, Magnetometer: 660 uA, total: 1.69 mA @ 3.3V.
- Output data rates (maximum): Gyroscope: 3200 Hz, Accelerometer: 1600 Hz, Magnetometer: 12.5 Hz.
- Range and Sensitivity: Gyroscope: ±250-2000 deg/s (131 - 16.4 LSB/deg/s), Accelerometer: ±2-16g (16384 - 2048 LSB/g), Magnetometer: ±1150 uT (28.5 LSB/uT) and ±2500 uT (13.1 LSB/uT).
- Space footprint: 2.5 × 3.0 × 0.95 mm
- Noise density: Gyroscope: 0.008 deg/s/$\sqrt{\text{Hz}}$, Accelerometer: 180 ug/$\sqrt{\text{Hz}}$, Magnetometer: 0.5 uT.

5.3.4.2 Pressure Sensor

We included the MS5837-30BA underwater pressure sensor with built-in thermometer for temperature compensation for altitude measurements [177]. The sensor can be used up to 300 meters depth in water, with a resolution of 0.2 cm. It consumes 0.6 uA @ 3.3V of power. The space footprint is 3.3 × 3.3 × 2.75 mm.

5.3.4.3 GPS

We also included a Origin ORG-1411 GPS module [178] for occasional localization error correction when GPS signal is available. ORG-1411 is the world's smallest profile GPS module with included patch antenna, with a footprint of only 10 × 10 × 3.8 mm with a mass of 1.4 grams. The position resolution is approximately 2 meters. The GPS consumes 77 mW of power during fix with a time-to-first-fix (TTTF) of approximately 30 seconds [176], and also features a low-power mode of 15 uA @ 3.3V. It is possible to use aided start (assisted-GPS) by storing almanac and ephemeris data of GPS satellites in the flash, reducing TTTF to 6 seconds.

5.3.5 Target Hardware Platforms

For benchmarking optimized neural inertial architecture found via NAS as well as baselines in terms of latency, memory usage and flash usage, we setup several embedded boards with varying clock speeds, SRAM and flash apart from the described hardware platform:

- STM32L476RG Nucleo (128 kB SRAM. 1 MB flash)
- STM32F746 Discovery (340 kB SRAM, 1 MB flash)
- STM32 Nucleo-144 (1 MB SRAM, 2MB flash)
- Arduino BLE 33 (256 kB SRAM, 1 MB flash)
- Arduino Uno R3 ATMega328p (2 kB SRAM, 32 kB flash)

5.4 The NESL Earable Mobility Corpus

The Networked and Embedded Systems Lab (NESL) Earable Mobility Corpus was developed to support the integration of dynamic/spatial audio capabilities in a mixed reality MR) environment using sensor modalities from the physical world. Specifically, we use eSense [166], an open earable platform [167][168] to collect inertial data of the head pose and stream the data to an edge device via BLE. Using traditional or DL frameworks coupled with Bayesian and non-Bayesian filters, the user's motion/status changes are mapped to the reference frame in the virtual world via a head-pose transfer function, ultimately altering the dynamics of the monoaural sound sources in the virtual world to dynamic audio. The MR platform reacts to head mechanics by creating a digital twin of sound dynamics (e.g. attempting to reproduce the Doppler Effect and Binaural Effect) and transmitting it to the earables to complete a perception-processing-feedback loop. We test the performance of Madgwick filter on this dataset.

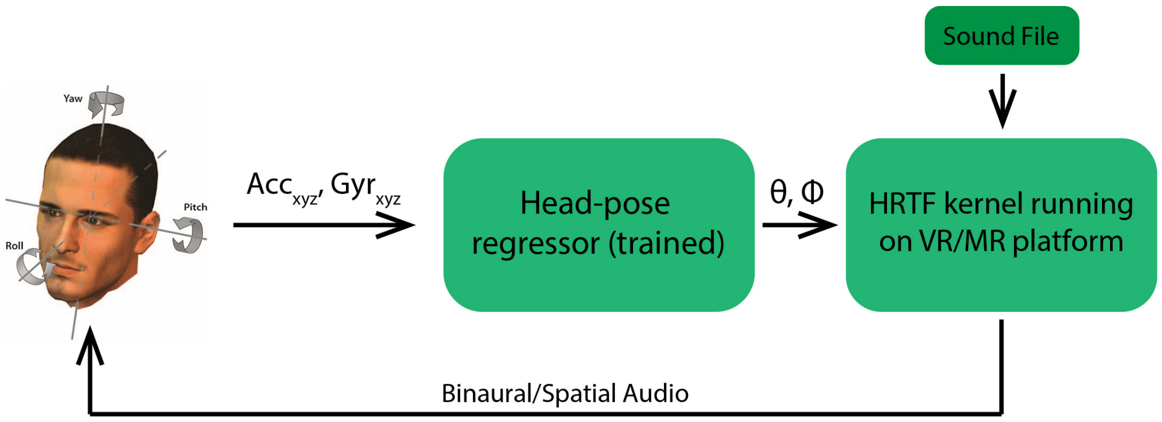


Figure 5.11: Goal of the NESL Earable Mobility Corpus

5.4.1 Data Collection Framework

To collect head-pose training data, a subject is instructed to sit at a particular position (called the origin, say (x', y', z')) in the experimental testbed with the appropriate earables worn on both ears. The left earable, which houses the 6DOF IMU, transmits inertial data to an android app developed for mining eSense IMU data. The right earable is used to record sound being emanated from a speaker with the centroid at a fixed point (x'', y'', z''). In addition, the subject is asked to wear a hat with OptiTrack IR markers mounted in a rigid body configuration. The subject is then asked to look at several marked points in the 3D space in the experimental testbed with respect to (x'', y'', z''), i.e. the user initially fixes his head towards the origin and then moves his head towards a particular point (x''', y''', z'''). Two types of transitions need to be taken into account:

- Head movements from origin to marked point and vice versa.
- Head movements from origin to multiple marked points and vice versa.

In the experimental testbed, there are 27 marked points and we collect 34 (13+21) distinct head-pose inertial frames per subject. Each user keeps his gaze fixed at the marked points for at least 1/2 seconds. The experimental setup is shown in Figure 5.12.

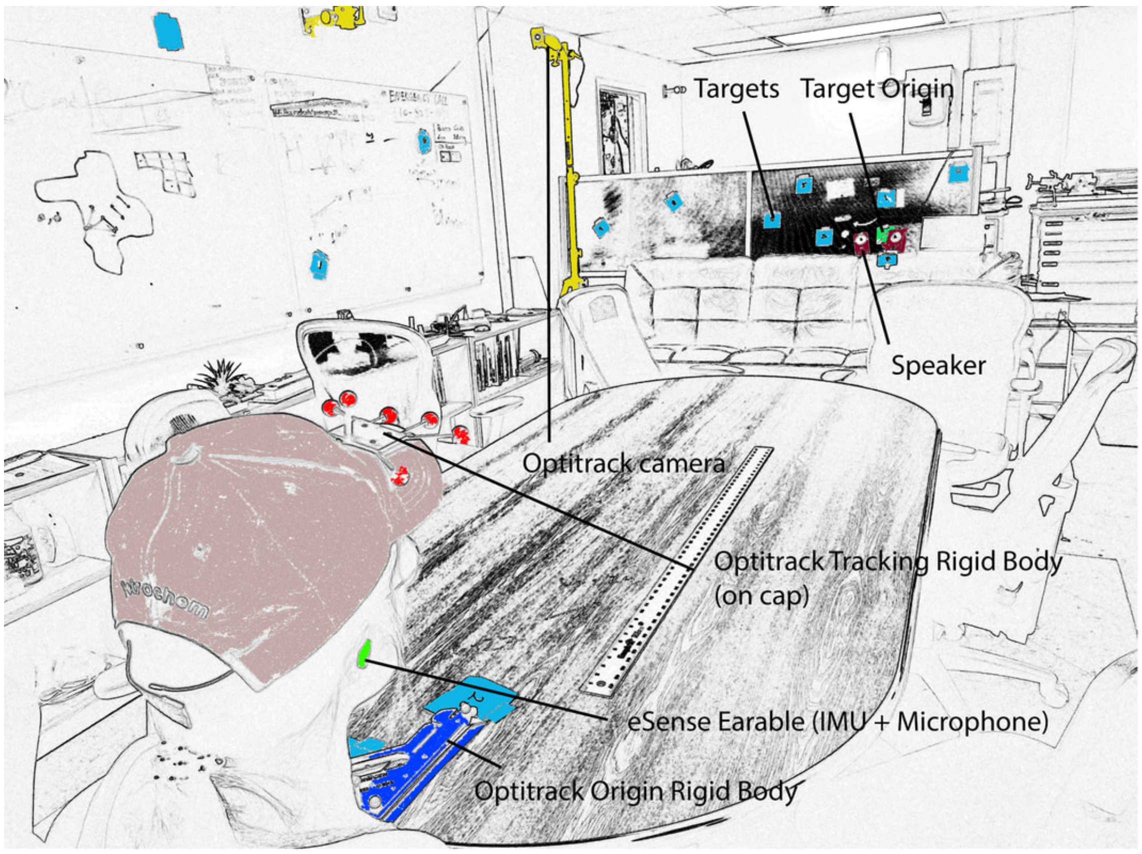

Targets
Target Origin
Speaker
Optitrack camera
Optitrack Tracking Rigid Body (on cap)
eSense Earable (IMU + Microphone)
Optitrack Origin Rigid Body

Figure 5.12: Experimental setup for earable, Optitrack and audio data collection.

The OptiTrack system consists of 6 IR cameras and serves as the ground truth for the eSense inertial data, with a screen recorder application recording visual cues and the Motive-Tracker software recording inertial data of the rigid body formed by the markers. In order to synchronize all of the data collection systems, timestamping is used such that the frameworks synchronize time with their respective local system clocks already synchronized with time from the Internet NTP servers. To help simplify things further, the user nods his head in a certain motion (we call it "calibration nod") while saying "start" before providing the head-pose readings. The subjects were asked to do the following in part 1 of the experiment:

- Wear the cap. Inform the collector to start data collection. Collector starts the e-Sense android app, starts the binaural audio playing on the speakers, starts audio recorder and finally starts the Optitrack system.

- When collector says ”START”, nod your head horizontally in “No” motion three times and vertically in “Yes” motion three times (this is to sync the OptiTrack system with the e-sense system). Collector will show how to do it efficiently. Say the word “Start” when you start nodding your head.
- Start with your eyes fixed at the origin for 2 seconds. (step 1).
- Move your head towards a target. (step 2)
- Keep your head fixed at the target for 2 seconds. (step 3)
- Move your head back to origin. (step 4)
- Repeat the above steps (steps 1 to 4) for the following transitions (origin – target – origin). If you can't see a number, make sure your point it out to the data collector before starting the experiment.
- Once you're done (make sure your head is fixed towards origin, inform the data collector without moving your head). Don't remove cap or earables. Collector will stop data collection.

Part 2 of the experiment followed similarly, except subjects moved their head from origin to first target, first target to second target and then back to origin. For the last 2/3 subjects, we used a laser pointer to help subjects identify targets. In order to characterize the position of each marked point in the 3D experimental testbed, we use a Leica Disto X3 laser rangefinder and a smartphone compass to obtain the distance, azimuth and elevation angles of the targets from the point the subject sits. It is important to note that since height of each subject varies, it is important to designate the azimuth and elevation angles of each point with respect to the origin separately for each subject. We keep record of the height of the ears of the subject from the Optitrack origin rigid body, their natural height, weight and age. We have found out that if we set the rangefinder at the average height of the ears of the subject from the Optitrack rigid body (15 inches) of all subjects, then the elevation angles is 0 degrees when the rangefinder points directly towards the origin. Figure 5.13 shows the target locations in the 3D space.

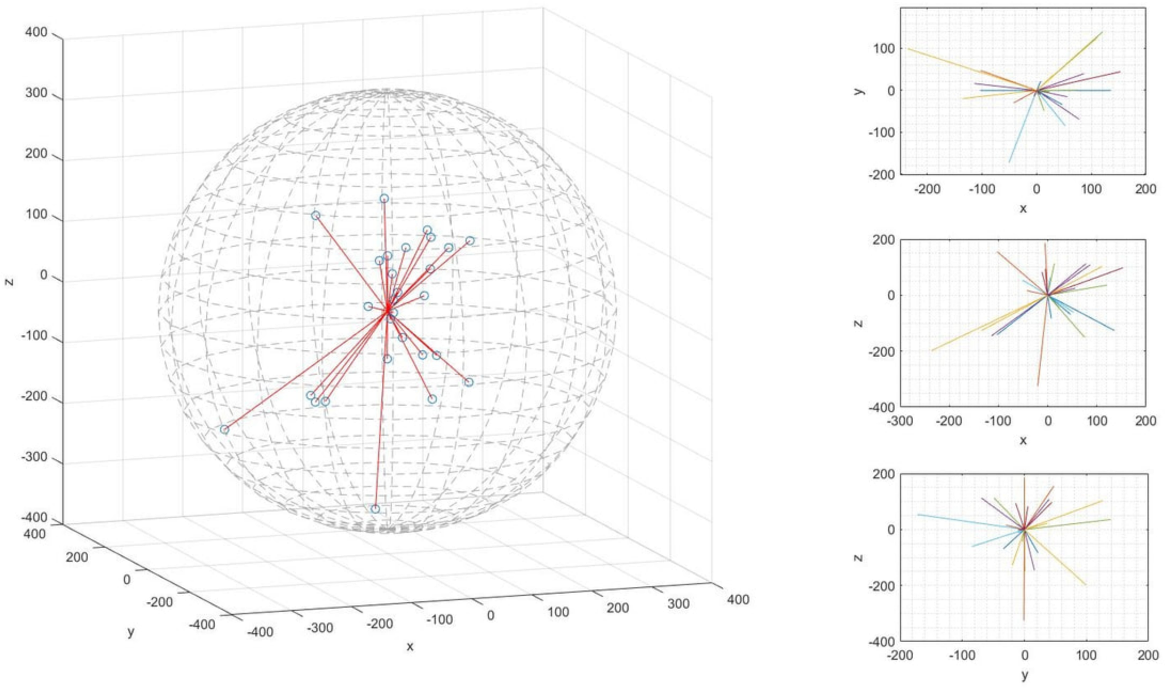


Figure 5.13: Targets characterized in Cartesian Coordinates (dimensions are given in inches).

In addition, we also aim to create a HAR dataset using the earable sensors. Using the same subjects and an optical camera for ground truth, we have created a database of several fine-grained activities such as activities relevant to body and limb motions pertinent to VR/MR context. Such a dataset maybe useful for future research where the user is able to move physically in a MR context rather than being bounded to head-movement. The following activities have ben recorded: 1. Walking 2. Jogging 3. Jumping 4. Standing 5. Turning Left 6. Turning Right 7. Sitting 8. Laying 9. Falling.

The Optitrack data is recorded as .tak files, while the earable inertial data is recorded as .txt file. The audio data is recorded in .mp3 format and the camera data were recorded as video files. The earable parameters were set as follows:

- Range of accelerometer: +- 2g, gyroscope: +- 500 deg/S
- Advertisement and connection intervals: 45-55 mS and 20-30 mS respectively

- Sampling rate: 100 Hz.
- Low pass filter: Enabled for both accelerometer and gyroscope, cutoff frequency set at 5 Hz for both.

In total, we collected inertial data from 15 subjects.

5.4.2 Data Preprocessing

Filtering, windowing and labeling of the data were done manually, correlating visual Optitrack cues with gyroscope sensor data plots as well as audio data for assistance. Manual correlation was applied for four reasons:

- Some subjects made mistakes during takes, either missing designated targets, overshooting their head movement or looking at wrong targets.
- Some subjects didn't move their head at all while looking at targets very close to the origin (targets 1-6 for example). They were not captured by Optitrack, let alone the earable.
- The sampling rate of earable data collection was not constant and jumped erratically between 1 Hz to 100 Hz due to loss in LoS between earable and mobile phone, BLE channel latency, improper advertisement and connection intervals of the earable as well as smartphone's performance bottleneck.
- Sometimes, the app didn't collect data at all (missing data).

We observed that when user moves his head left and right, gyroscope's x and y axis provides negative and positive values respectively (unless a subject decides to wear the earable in a completely erroneous manner, which happened in case of one subject. It is important to note that the earable goes in only one way). Likewise, When user moves his head up and down, gyroscope's z axis provides negative and positive values respectively (regardless of how the user wears the earable). A single head rotation on the gyroscope-time plot essentially consists of a triangular/bell curved shape peak, with the rate of change of the angular velocity as well as the angular velocity being proportional to how fast the user moves his head (faster = thinner and taller peak, slower = thicker and smaller peak). In case of head movements from a single target and back, one will observe two opposite peaks on the gyroscope plot. For dual targets, one will observe groups of three peaks. The observations are illustrated in Figure 5.14.

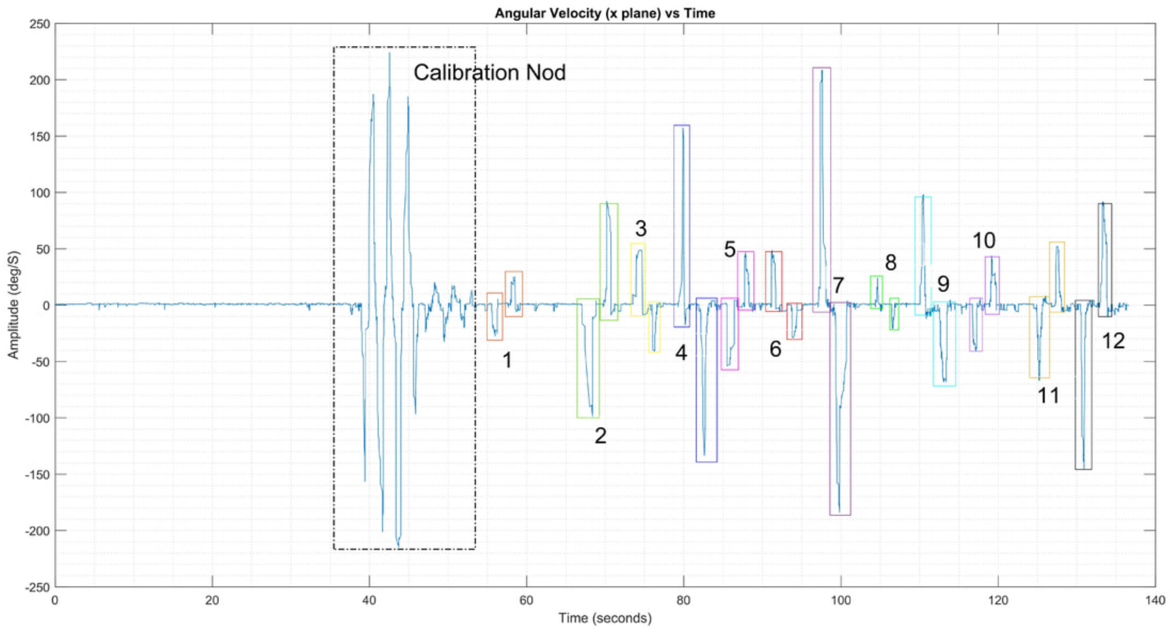


Figure 5.14: Sample plot of angular velocity (x plane) vs time for origin-target-origin take. The head movements are identified and numbered in the graph visually here without the need for aid of ground truth data from Optitrack.

After slicing, cleaning and labeling the headpose data from all 15 subjects. 1266 inertial frames were obtained. 356 of these frames were captured for origin-target-origin takes and 910 of these frames were captured for origin-target1-target2-origin takes. 607 of the 910 frames correspond to origin-target1 and target2-origin frames while the rest 303 frames correspond to target1-target2 frames. The head pose estimation is presented in the form of altitude and elevation and direction appropriate to the HRTF. The HRTF is then interpolated to the point specified by the IMU and filters are made for left and right audio. For activity classification, a sliding window of 1 seconds with stride length of 0.5 seconds was chosen.

241 spatial features were extracted from 8002 frames of 9 activities. For this thesis, we omit the activity recognition part and instead focus on the performance of Madgwick filter as a head-pose regressor.

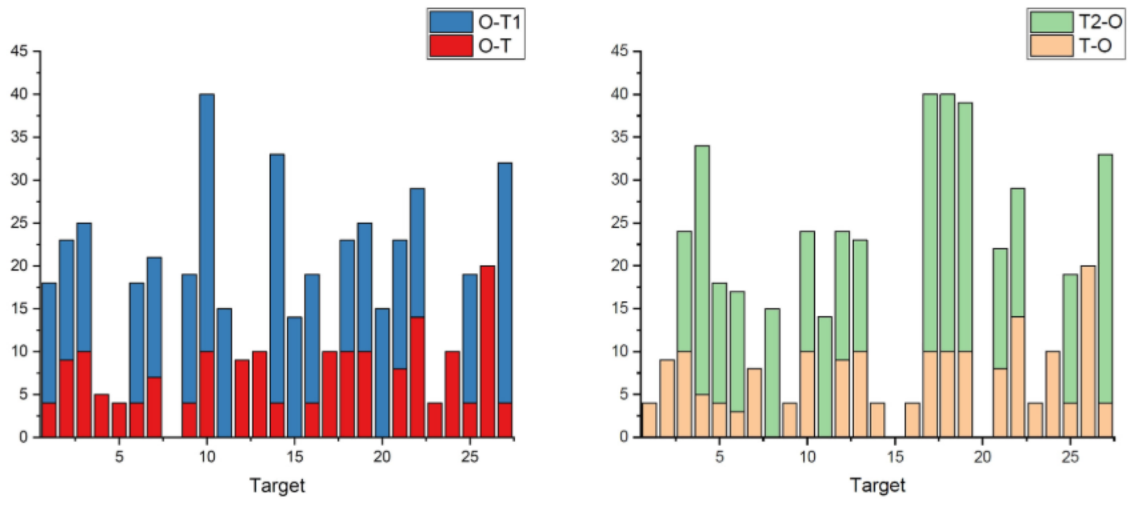


Figure 5.15: Number of frames of all 27 targets for head movement from origin to target and target to origin (apart from these, there also exists target-target head movement).

CHAPTER 6

Preliminary Evaluation and Discussion

In this chapter, we discuss the results of our preliminary benchmarks and provide inferences on the observations. We divide the preliminary evaluation into three parts:

- Evaluation of the robust uncertainty aware hierarchical complex event processing pipeline on the cooking activity dataset.
- Evaluation of efficient neural inertial localization framework, namely the TCN obtained via NAS on the OxIOD dataset and various target hardware.
- Evaluation of orientation estimation algorithm on the NESL Earable Mobility Corpus.

6.1 Complex Activity Recognition

This section is concerned with the ideas presented in Chapter 3 and 5.1.

6.1.1 Performance characteristics for vanilla models

Figure 6.1 shows the classification performance of proposed classifiers (window-by-window basis) on the raw dataset without incorporating any interpolation, augmentation, masking, or jitter techniques. For micro-activity classification, a multi-label classifier was used. From Figure 6.1, we see that vanilla LSTMs perform poorly as they are unable to extract spatially-correlated temporal features over long time-windows. However, since a macro-activity is composed of distinct constituent micro-activities, smaller time-windows may not provide sufficient discriminatory features to distinguish between each activity. This trend is evident for CNNs and DCBLs, where increasing the time-window generally increases the classification performance. DCBL provides better generalization over CNN (although macro-activity classification can largely be attributed to learning transient features over the latent space) by taking into account temporal insights of constituent micro-activities apart from spatial features, which may not be similar for each macro-activity data frame due to distinct constituent micro-activity. The best classification performance for both macro and micro-activity classification was observed for DCBL operating on 10-second windows, being 0.80 and 0.50, respectively, motivating us to choose DCBL-10 for our endeavors. The same classifier provided test accuracies of 0.73 and 0.37 on macro and micro-activity test splits respectively for window-by-window basis, and 0.77 and 0.48 on macro and micro-activity test splits respectively for file-by-file basis (taking maximum classification output of all windows in a file).

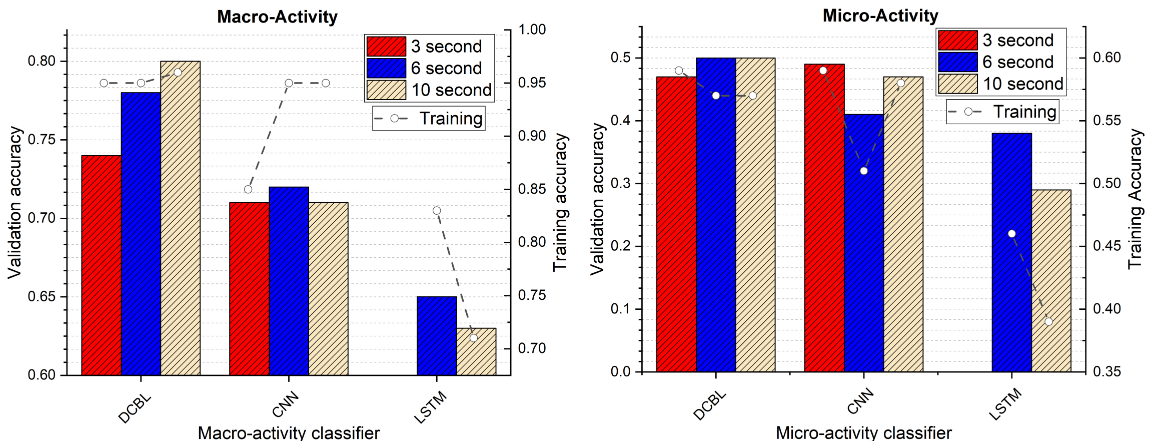


Figure 6.1: Average classification performance (window-by-window basis) for proposed architectures without pre-processing applied.

6.1.2 Effect of normalization and vanilla interpolation techniques

To manually account for missing data and smooth the output, pre-processing techniques such as bandpass filtering (using 20 Hz butterworth low-pass filter), normalization (z-normalization, uni-variance, and min-max scaling) and vanilla interpolation (linear, spline, autoregression) were applied but negligible or negative performance improvement was observed over vanilla approach. For example:

- Z-normalization on DCBL-6 provided a validation accuracy of 0.77 (compared to 0.78 for vanilla approach).
- Vanilla interpolation and smoothing techniques yielded validation accuracies as low as 30-40% for DCBL-3, DCBL-6 and DCBL-10. Our hypothesis is that since large sections of data are missing, vanilla interpolation (and smoothing) techniques do not generalize well to repair missing data.

6.1.3 Incorporating mo-cap data

Validation accuracies fluctuated within 30-50% for DCBL, CNN, and LSTM macro-classifiers using all 99 raw sensor channels (including mo-cap). Validation accuracy of 0.33 was observed for the DCBL-10 macro-classifier using only the 87 mo-cap channels. As discussed earlier, raw marker positions do not supply sufficient discriminatory features for each action class, yielding low accuracies. To extract temporal features such as velocity, acceleration, and jerk from the mo-cap channels, we utilized the RT-Mocap toolbox [179]. However, we did not observe significant performance gains, with validation accuracies remaining within 37-44% for both DCBL micro and macro-classifiers through the extracted features.

6.1.4 Performance with augmentation and jitter (without masking)

Table 6.1 and Table 6.2 illustrate the performance of DCBL-10 for macro and micro-activity classification after incorporating jitter and data augmentation. For handling missing data, we used the technique to push ahead contained samples and fill in the empty spots with a constant (in our case, we chose to fill the spots with zeros). To enhance the performance of micro-activity classification and allow usage of conditional classifier selection, we chose to train one-vs-all binary classifiers instead of multi-label classifiers.

Table 6.1: Macro-activity classification metrics for 5 random dataset splits.*

Trial	**Test (W-W)**	**Validation (W-W)**	**Train(W-W)**	**Test (F-F)**	**Validation (F-F)**	**Train (F-F)**
Split 1	0.75	0.79	1.00	**0.80**	0.83	1.00
Split 2	0.74	0.82	1.00	**0.78**	0.90	1.00
Split 3	0.84	0.80	0.99	**0.93**	0.85	1.00
Split 4	0.81	0.84	1.00	**0.88**	0.93	1.00
Split 5	0.73	0.73	0.99	**0.78**	0.83	1.00
5-way mean	**0.77**	**0.80**	1.00	**0.83**	**0.87**	1.00

* W-W stands for window-by-window basis. F-F stands for file-by-file basis (taking maximum classification output from all windows in a file).

Table 6.2: Micro-activity classification metrics (one-vs-all).

Binary Classifier	**Test (W-W)**	**Validation (W-W)**	**Train(W-W)**	**Test (F-F)**	**Validation (F-F)**	**Train (F-F)**
Cut (0)	0.81	0.78	0.86	**0.88**	0.82	0.99
Wash (1)	0.89	0.91	1.00	**0.90**	0.92	1.00
Take (2)	0.82	0.82	0.99	**0.84**	0.90	1.00
Put (3)	0.83	0.79	0.99	**0.90**	0.84	1.00
Peel (4)	0.87	0.85	0.98	**0.98**	0.96	1.00
Add (5)	0.94	0.97	0.99	**0.94**	0.98	0.99
Mix (6)	0.93	0.94	1.00	**0.92**	0.92	1.00
Pour (7)	0.91	0.92	1.00	**0.90**	0.92	1.00
Open (8)	0.96	0.93	1.00	**0.94**	0.92	1.00
Other (9)	0.84	0.88	0.99	**0.88**	0.86	0.99

For a file-by-file basis, average test accuracy of 0.72 was obtained for micro-activity classification for all 10 labels, yielding a 24% performance improvement over the vanilla approach (0.48). From Table 6.1, we can see that incorporating window jitter, time-shift augmentation and popping-ahead samples yielded 6% test performance improvement on average over vanilla approach (0.83 vs 0.77) for macro-activity classification (file-by-file basis).

6.1.5 Performance with augmentation, jitter and masking

Table 6.3 and Table 6.4 illustrate the performance of DCBL-10 for macro and micro-activity classification after incorporating jitter and data augmentation, as well as masking from [6], coupled with popping-ahead samples.

Table 6.3: Final macro-activity classification metrics (masking included)

Trial	**Test (W-W)**	**Validation (W-W)**	**Train(W-W)**	**Test (F-F)**	**Validation (F-F)**	**Train (F-F)**
Split 1	0.80	0.81	0.99	**0.88**	0.90	1.00
Split 2	0.91	0.97	1.00	**0.90**	1.00	1.00
Split 3	0.87	0.82	1.00	**0.95**	0.90	1.00
Split 4	0.82	0.86	1.00	**0.85**	0.95	1.00
Split 5	0.77	0.77	1.00	**0.80**	0.90	1.00
5-way means	**0.83**	**0.85**	1.00	**0.88**	**0.93**	1.00

Table 6.4: Final micro-activity classification metrics (masking included)

Binary Classifier	**Test (W-W)**	**Validation (W-W)**	**Train(W-W)**	**Test (F-F)**	**Validation (F-F)**	**Train (F-F)**
Cut (0)	0.80	0.79	0.98	**0.88**	0.87	1.00
Wash (1)	0.87	0.89	0.99	**0.88**	0.92	1.00
Take (2)	0.85	0.82	1.00	**0.90**	0.92	1.00
Put (3)	0.84	0.81	0.99	**0.86**	0.92	0.98
Peel (4)	0.85	0.89	1.00	**0.98**	0.92	1.00
Add (5)	0.94	0.96	0.93	**0.94**	0.96	0.93
Mix (6)	0.94	0.95	1.00	**0.94**	0.96	1.00
Pour (7)	0.91	0.92	0.99	**0.90**	0.92	0.98
Open (8)	0.94	0.93	0.99	**0.94**	0.93	1.00
Other (9)	0.86	0.81	0.99	**0.88**	0.82	0.99

From Table 6.3, we can see that incorporating masking yielded a 5% test performance improvement on average over the non-masking approach (0.88 vs. 0.83) for macro-activity classification (file-by-file basis). However, for micro-activity classification on a file-by-file basis, average test accuracy of 0.72 was obtained for the binary classifiers for all 10 labels using masking, which is same as the non-masked binary classifiers.

6.1.6 Analysis and Discussion

From our evaluation, we see that the performance of vanilla classifiers improve significantly when we incorporate our proposed training pipeline in the channel. Classical approaches such as interpolation, filtering, normalization and feature extraction (on mo-cap data) do not yield significant (in some cases lessen accuracy) performance improvement over vanilla classifiers. We hypothesize that classical approaches fail to converge due to the unnatural way the missing data was emulated in the dataset, consisting of long blocks of missing and abnormally aligned samples not normally encountered in the real world, coupled with the innate trade-off and time-evolving issues of classical methods discussed in section I. The uncommon nature of the dataset actually hurts micro-activity classification performance gains when we incorporate masking in the pipeline along with jitter and augmentation, yielding in negligible performance improvement. This occurs because masking assumes perfect sample alignment among adjacent sensors. Thus, the gains obtained from "space-aware" neural architectures is nullified by the natural misalignments of sensor samples in the dataset. However, our proposed pipeline component, namely augmentation, and jitter, clusters the non-missing samples within a window at the beginning, ultimately yielding 24% performance improvement for complex activity recognition over vanilla approaches and thus exhibiting promising performance characteristics in worst-case multimodal sensing scenarios.

From Figure 5.1, we see that the dataset is not balanced, with some classes having fewer occurrences than others. This affected the output on the test dataset, with some of the rarely occurring labels being missed by our classifiers. To minimize the penalty, we chose one-vs-all classifiers for micro-activity recognition to attempt to provide sufficient "positive and negative" examples to the classifiers. Additionally, we observed that subject-dependent training (e.g., use subjects 1 and 2's files for training and test on subject 3) yields inferior generalization performance due to the class imbalance as well as limited training examples. This also limits the direct usage of information-rich high sampling rate sensors (e.g., mo-cap), which translates to more model parameters and requiring more training examples than currently provided.

Table 6.5: Summary of the Top 10 teams in the 2020 Cooking Activity Recognition Challenge [154].

Rank	Classifier	Used Sensing Modalities	Competition Accuracy (%)	Window Size (s)	Training Time	Testing Time	RAM	CPU	GPU
1	HMM	Mo-cap	92.08	2	<1 min	<1 min	16 GB	Intel I7-4790	-
2	k-NN	Mo-cap, IMU (all)	61.05	30	15 seconds	-	16 GB	Intel I7 2.8 GHz	-
3	**DCBL (uncertainty aware)**	**IMU (RW, RA, LH)**	**59.11**	**10**	**6.5 hours**	**8 min**	**128 GB**	**3.4 GHz AMD Ryzen Threadripper 1950X (16 core)**	**2x Nvidia GeForce GTX 1080 Ti**
4	CNN and GCN	Mo-cap, IMU (RA)	56.63	NA	2 hours	12 min	30 GB	3.7 GHz Intel I7-8700K	4x Nvidia GeForce GTX 1080 Ti
5	LightGBM and Naive Bayes	Mo-cap, IMU (all)	55.00	10	530 seconds	20 seconds	32 GB	4.00 GHz	-
6	ConvLSTM	IMU (all)	52.79	0.5	50 seconds	117 seconds	64 GB	3.7 GHz Intel I7-8700K	Nvidia GeForce RTX 2080 Ti
7	Multi-sampling Classifiers	Mo-cap, IMU (all)	43.39	30	1 hour 49 mins	12 seconds	64 GB	Intel Xeon ES-1620v3	Nvidia GeForce GTX 1080 Ti
8	DC-GRU	IMU (RW, RA, LH)	42.16	3	-	-	8 GB	Intel I3	-
9	CNN	IMU (all)	32.75	10	4.2 mins	1.3 mins	16 GB	Intel I7	Nvidia GeForce RTX 2060

6.1.7 Comparison with Contestant Models

For the test dataset (without labels) provided for the competition, we used an ensemble of the 10 macro-activity classifiers (5 with masking and 5 without masking), while for macro-activity classification, we used 20 micro-activity binary classifiers, with 2 classifiers (masked and non-masked) for each of 10 micro-activities. Table 6.5 illustrates the performance of our model on the unlabelled test set versus other contestants. We ranked 3rd in the competition. We can make several observations.

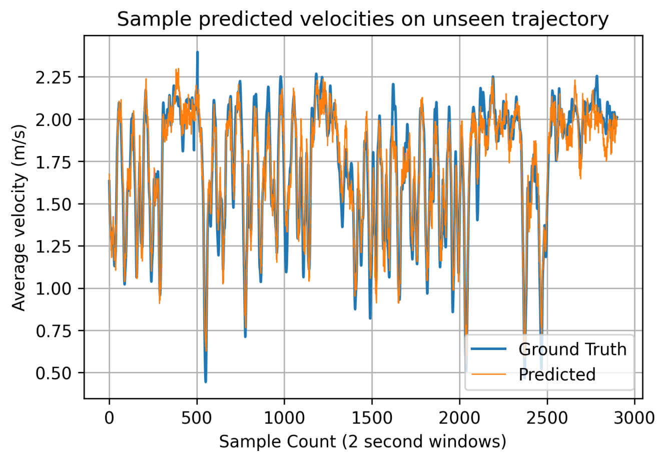
- Out of all the teams using DNN, our framework scored the highest.
- We were the only team to acknowledge the issues of missing data, sampling rate jitter and timestamp misalignment in the dataset.
- The top two teams required manual relabeling of the training set (and even the test set) using reference videos generated from the Mo-cap data. Our pipeline is completely autonomous and agnostic of the test data characteristics.
- Note that the mo-cap data did not have any missing data or jitter issues. It is easy to manually relabel the entire dataset by contructing videos from the mo-cap data. Among the best performing teams using only the inertial sensor data, Team 6 performed $\sim 6\%$ worse than our framework, while on average, our framework provides 16% improvement in classification accuracy over other teams using only the inertial sensor data.
- Even after using the error-free Mo-cap data, most of the teams still received poor classification accuracy on the test set, likely by the uncertainty-related errors introduced by the use of inertial data.

6.2 Efficient Neural Inertial Localization

This section is concerned with the ideas presented in Chapter 4, 5.2 and 5.3.5.

6.2.1 Preliminary Backbone Network Performance

Figure 6.2 shows part of the performance of the backbone network for displacement and heading rate estimation on unseen trajectories.



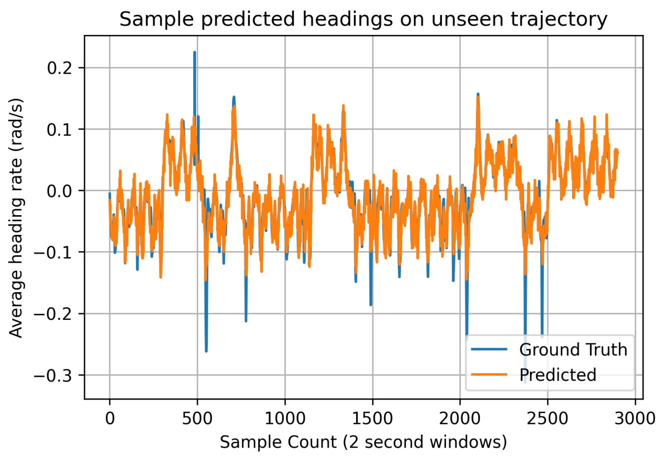


Figure 6.2: Sample velocity and heading estimation by the backbone network for 4.9 minutes of unseen trajectory.

The performance of the backbone network on unseen trajectories (9 files, 36234 2-second 6-channel inertial sequences) are reported as follows:

- Displacement RMSE (meters): 0.11 meters.
- Heading rate RMSE (rad/s): 0.09 rad/s (5 degrees).

The validation RMSE are as follows:

- Displacement RMSE (meters): 0.013 meters.
- Heading rate RMSE (rad/s): 0.0099 rad/s (0.5 degrees).

Figure 6.3 shows sample trajectory plots on unseen test data estimated by our TCN. The ground truth trajectories are shown on the same plots.

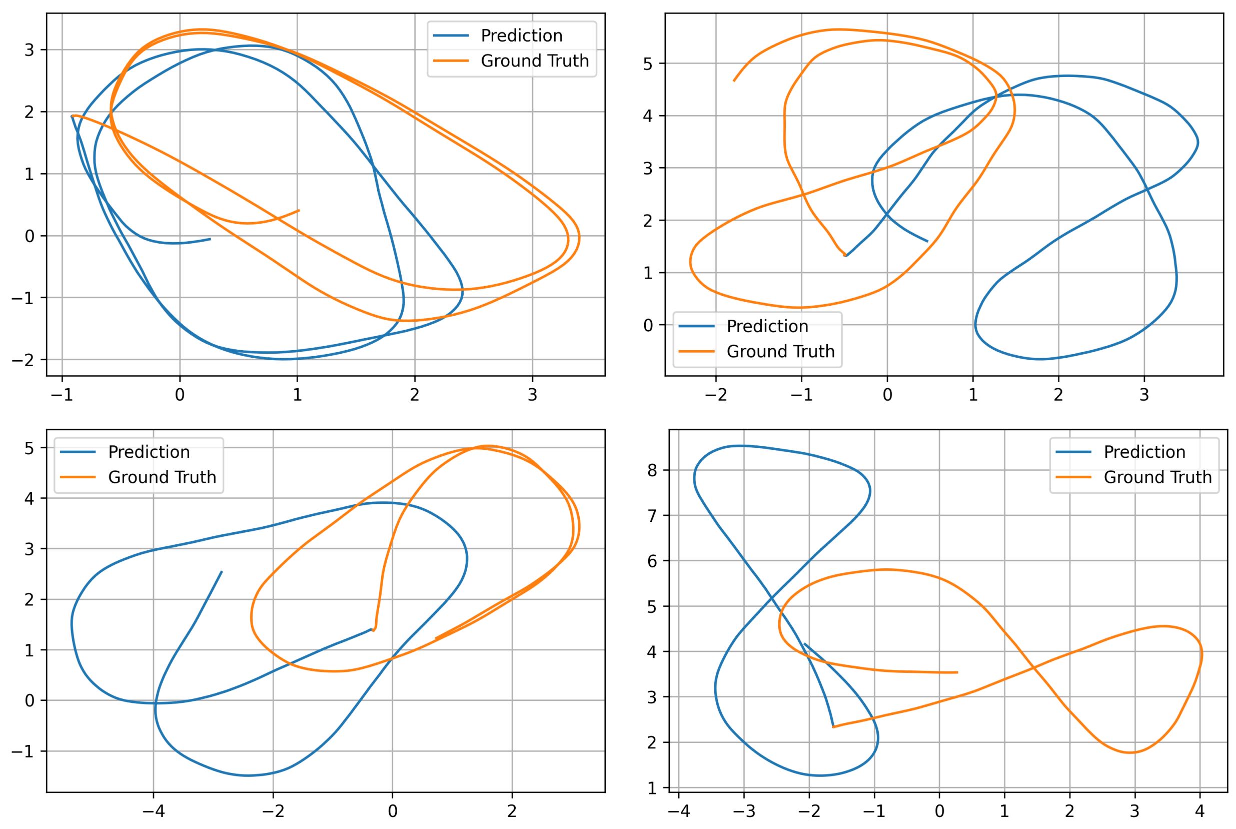


Figure 6.3: Sample trajectory projections on unseen test data by our neural inertial model (unit: meters).

We can make several observations from Figure 6.2 an 6.3:

- The backbone network is only 0.5 MB in size. However, it can still predict displacements with sub-meter resolution under eclectic sensor domains on unseen trajectories, with the heading being within 5 degrees.
- Predicted polar vectors sometimes suffer from high-frequency noise, which requires smoothing. Chen et al. [131] suggested using a low-pass filter at the output of the network to remove the effects of inertial-sensor noise in the final output. We suggest using a 3rd order Savitzky–Golay filter with a window size of 50 to smooth out the TCN predictions.

- The dataset contains sudden initial peaks in the ground truth heading rates and velocities at the start of each file. During those times, the inertial sensors do not actually record anything. We hypothesize that those initial movements correspond to Vicon calibration cues. The RMSE is expected to improve if we remove those cues.

- While the shapes and scales of predicted trajectories match the ground truth, there is a slight rotation. This is caused by mismatch in the initial headings provided by the ground truth data, which is due to the calibration cues in the data. This causes the trajectory to move in the wrong direction.

- We found out that a single TCN can predict both the displacement and heading equally well as two TCNs. This is due to the latent connection between displacement and heading. Note that, the heading and displacement needs to be normalized to the same scale for a single TCN to work well.

- We observed that the TCN took approximately only 20 epochs to stabilize. Vanilla recurrent networks take longer to converge.

- Most of the trajectories in the OxIOD dataset are circular. This may not be the most realistic dataset when compared with practical human walking. One of our future goals is to benchmark the performance of our models on more realistic datasets such as RIDI and RoNIN [75][82].

6.2.2 Sample Architectures Generated by NAS

Figure 6.4 shows sample networks generated by our proposed NAS algorithm compared to human engineered TCN network [131]. Table 6.6 illustrates the performance of NAS algorithm under various device constraints. Note that for this section, the network was trained to regress absolute headings compared to heading rates. In addition, we did not use quantization. The orange shaded model in Table 6.6 is the one used for later benchmarks, as it pertains to our designed target hardware. Note that Mbed OS, TFLite Micro and sensor libraries require ∼50 kB SRAM and ∼280 kB flash on Cortex M4 devices. TFLite Micro requires ∼32 kB SRAM during runtime.

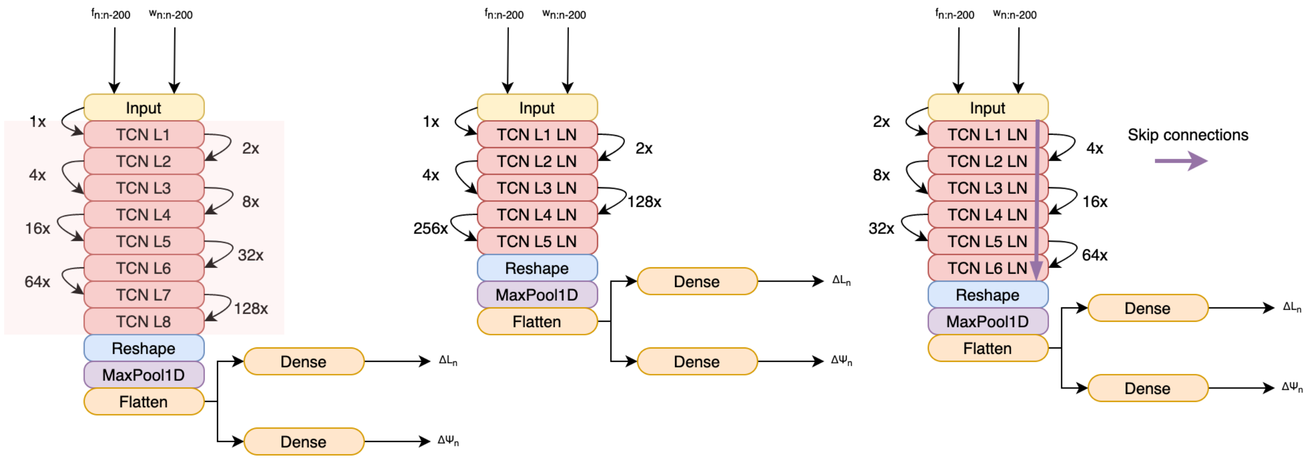


Figure 6.4: (From Left) Backbone supernetwork designed by humans, optimized TCN produced by NAS for STM32F746 Discovery and optimized TCN produced by NAS for STM32L476RG Nucleo (LN refers to layer normalization).

From Figure 6.4 and Table 6.6, we can make several observations:

- Proposed NAS yields different pareto-optimal but specialized network architecture for different target devices, regularized by SRAM, flash and latency constraints. In other words, the NAS algorithm tries to ensure that the generated network fully exploits the hardware capabilities of target device instead of providing a small model for all devices.

- Our NAS algorithm balances local and global semantics in the inertial windows to retain accuracy in the long run. In other words, Bayesian NAS uses low dilation factors in the lower layers to capture local spatial connectivities, and large dilation factors in the higher layers to capture global temporal dynamics, striking a balance between spatial and temporal feature extraction.

- Hard constraints do not regularize the Gaussian process properly, leading to poor search space and models with higher number of FLOPS (more latency) and activation size

Table 6.6: Summary of pareto-optimal architectures generated by NAS for diverse target constraints

Target Hardware (RAM, Flash)	**SRAM (kB)**	**Flash (kB)**	**FLOPS***	**No. of filters**	**Parameter Count**	**Kernel Size**	**Dilation Factors**	**Dropout**	**Skip Connections**	**Normalization**	**Heading RMSE (rad.)**	**Displacement RMSE (m)**
None (L-IONet)	162	537	13.8M	32	31874	3	[1, 2, 4, 8, 16, 32, 64, 128]	0.0	No	None	0.61	0.092
STM32L476RG (128 kB, 1 MB)**	75	102	8.15M	41	17877	2	[8, 32, 256]	0.0	No	None	0.58	0.15
STM32L476RG (128 kB, 1 MB)	56	117	5.42M	19	13092	3	[2, 4, 8, 16, 32, 64]	0.0	Yes	Layer-wise	0.57	0.17
STM32F746 Discovery (340 kB, 1 MB)	263	317	29.1M	42	66110	4	[1, 2, 4, 128, 256]	0.0	No	Layer-wise	0.58	0.14
STM32 Nucleo-144 (1 MB, 2 MB)	396	496	-	48	100042 (50% non-trainable)	3	[4, 8, 32, 128]	0.0	No	Weights	0.58	0.13
Arduino BLE 33 (256 kB, 1 MB)***	132	192	15.3M	22	32254	7	[4, 32, 64, 128, 256]	0.1	Yes	Layer-wise	0.59	0.19

* we did not optimize FLOPS due to a problem with TF 2.4 profiler when these experiments were run
** hard constraints on SRAM and flash for 100 iterations (without informing NAS), others are 10 iterations
*** early stopping used

for the same target hardware. In hard constraint NAS, the value of the optimization function (to be maximized) is directly set to a very negative value without running the training process when the chosen model parameters exceed the required flash, RAM or latency constraints. Meanwhile, in soft-constraint NAS, the device constraints are modeled as rectified linear unit (ReLU).

- Random scalarizations helps weigh the important additive terms in the optimization function, making sure the effects of all the terms (accuracy, latency, flash and RAM) are equally taken into account.
- Obtained error metrics are comparable to L-IONet but with up to 3× peak activation reduction and 5.3× memory usage reduction for the most constrained target hardware.
- Relaxed device constraints yields model with slightly better error metrics, but would also be slower due to higher FLOPS count. Very rarely, TinyOdom makes unrealistic choices for each NAS epoch, yielding in slow search times for 1 or 2 epochs (out of every 10) due to large training time for unrealistic models that will yield in large penalties. This is not the case in case of NAS with hard-constraints, which however, loses out on proper regularization and information gain.

6.2.3 Comparison of Model Performance with Baselines

Figure 6.5 shows the error characteristics of our optimized model compared to SOTA baselines, as as well as RAM and flash usage. From Figure 6.5, we see that our model, while bringing only 117 kB in size and requiring only 56 kB of SRAM, can achieve sub-meter (∼0.15 meter) trajectory projection without using GPS or large models for unseen trajectories. While classical models such as EKF provides superior latency, RAM and flash usage profiles, EKF is unusable without the aid of GPS or heuristics, diverging rapidly to thousands of meters away as seen in the figure due to drift and noise. In addition, our model is pareto-optimal with L-IONet, which is the only data-driven neural inertial architecture that can run in real-time but on smartphones. Our model has 2.5× lower activation and 6× lower flash requirements compared to L-IONet. One weakness of our model, however, is the model latency, as shown in Figure 6.6. On average, we observed that our model can run on STM32L476 series chips at a sampling rate of 7 Hz for 2-second windows (@100 Hz). While this is much lower than L-IONet or TinyEKF's sampling rate, there are several ways by which we could accelerate the process:

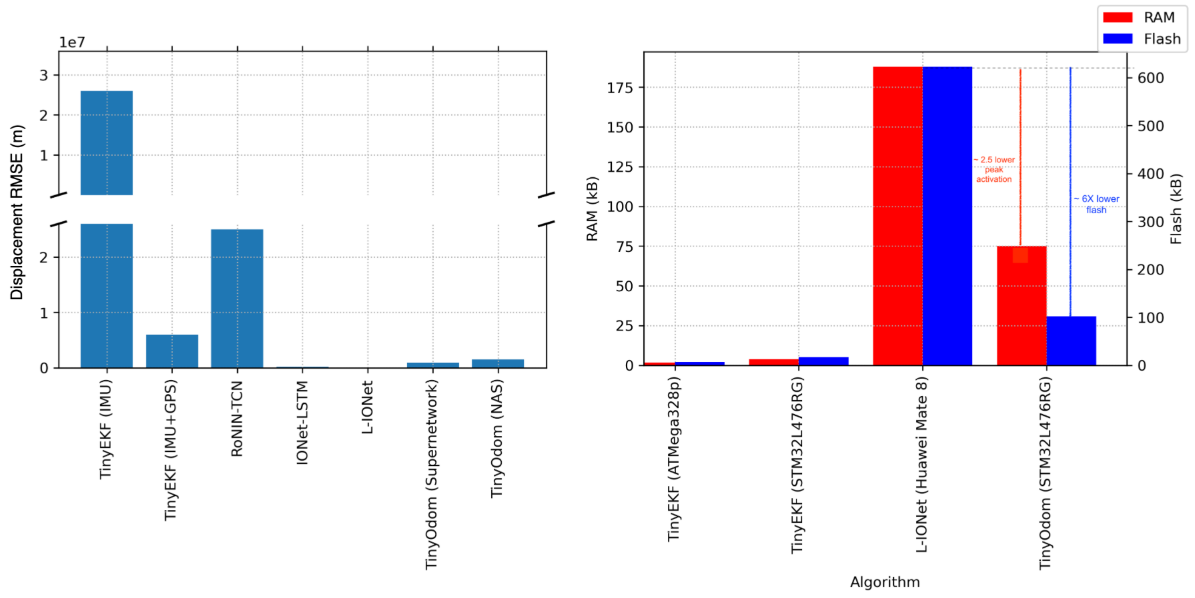


Figure 6.5: (Left to right) Displacement RMSE (meters) and SRAM and flash usage on target hardware for various SOTA inertial dead-reckoning algorithms. TinyOdom is our optimized network.

- Quantization of weights, as discussed in Chapter 4.
- Using only the operators necessary to run the model rather than clogging the device with all the operators.
- Stronger scalarization on FLOPS.

Overall, our model provides a trade-off among the baselines, reaching optimal error characteristics suitable for everyday use while using as little compute resources as possible on TinyML devices.

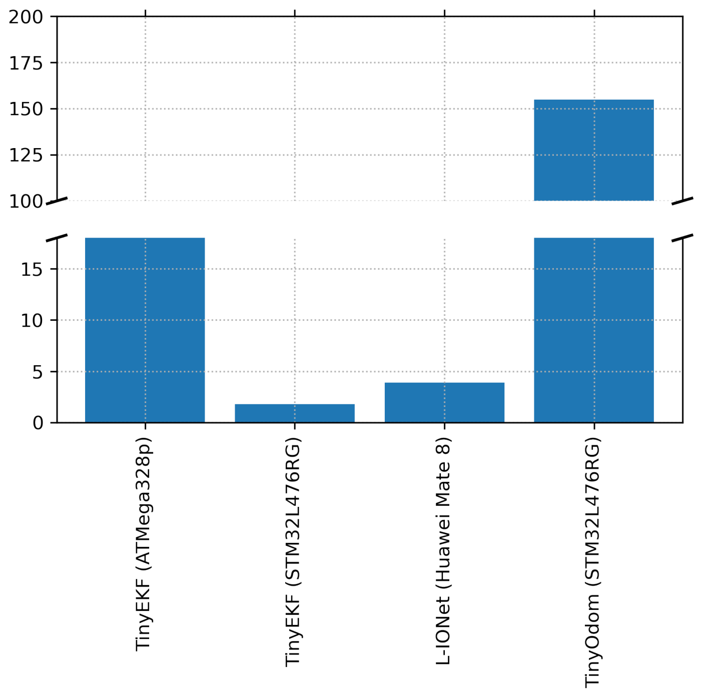


Figure 6.6: Latency on target hardware for various SOTA inertial dead-reckoning algorithms. TinyOdom is our optimized network.

6.2.4 Bayesian NAS versus SOTA NAS

SOTA in NAS for microcontrollers include MCUNet [24], which uses Once-for-All [23] and evolutionary search on an optimized search space based on FLOPS and MicroNets [28], which uses gradient driven NAS formulation through various device proxies. We could not implement MicroNets due to code unavailability, and MCUNet due to absence of complete code. While we do not claim that our NAS technique is clearly the better one, we can make some back-off-the-hand calculations in favor of our NAS approach over the SOTA.

- None of the papers talk about search cost and instead focus on accuracy, our obtained accuracies are pareto-optimal with respect to the backbone network (best achievable), similar to what the two NAS algorithms also show in their respective papers for their models.

- MCUNet reduces RAM and flash usage by 3.5× and 5.7× for Imagenet (larger network to begin with) while losing 10% accuracy over once-for-all, we reduce RAM and flash usage by 2.5× and 5× while losing only cm level precision (with the goal of retaining sub-meter accuracy) over a smaller subnetwork. It is harder to optimize a small network over a large network [43][44]. Note that our search space however, is not tiny, with 10^{14} combinations.

- With such a huge search space, TinyOdom only takes 12-24 GPU hours per network for combined training and search. Compare this with Once-for-All search used in MCUNet, which requires 40 GPU hours on 16000 subnetworks for search alone (training time for Once-for-All supernetwork calculation was inaccurate so we did not include it here).
- MCUNet uses proprietary engine and does not support wide-variety of devices or custom neural blocks, our NAS supports wide-variety of target hardware and neural blocks, coupled with support from TFLite Micro. In addition, TinyEngine from MCUNet does not provide any feasible improvements over TFLite Micro [28][63], rather a hassle to debug as it is compiled.

6.3 Performance of Madgwick and Altitude Filter

This section is concerned with the ideas presented in Chapter 4.3. Figure 6.7 shows the performance of Madgwick filter for headpose estimation in the NESL Earable Mobility Corpus. From Figure 6.7, we observe that the median delta motion (change from one pose to another pose) errors are 8.3 and 5.6 degrees in azimuth and elevation planes respectively. Ferlini et al. [180] proposed the fusion of 6DOF inertial data from two earable sensors in order to estimate final head position using quaternions, delta motions and complementary filters, achieving estimation errors as low as 5.4 degrees and 18.7 degrees for short and long movements (noisier) respectively. We achieved lower head-pose (and ultimately orientation) estimation error using the Madgwick filter with a single earable sensor.

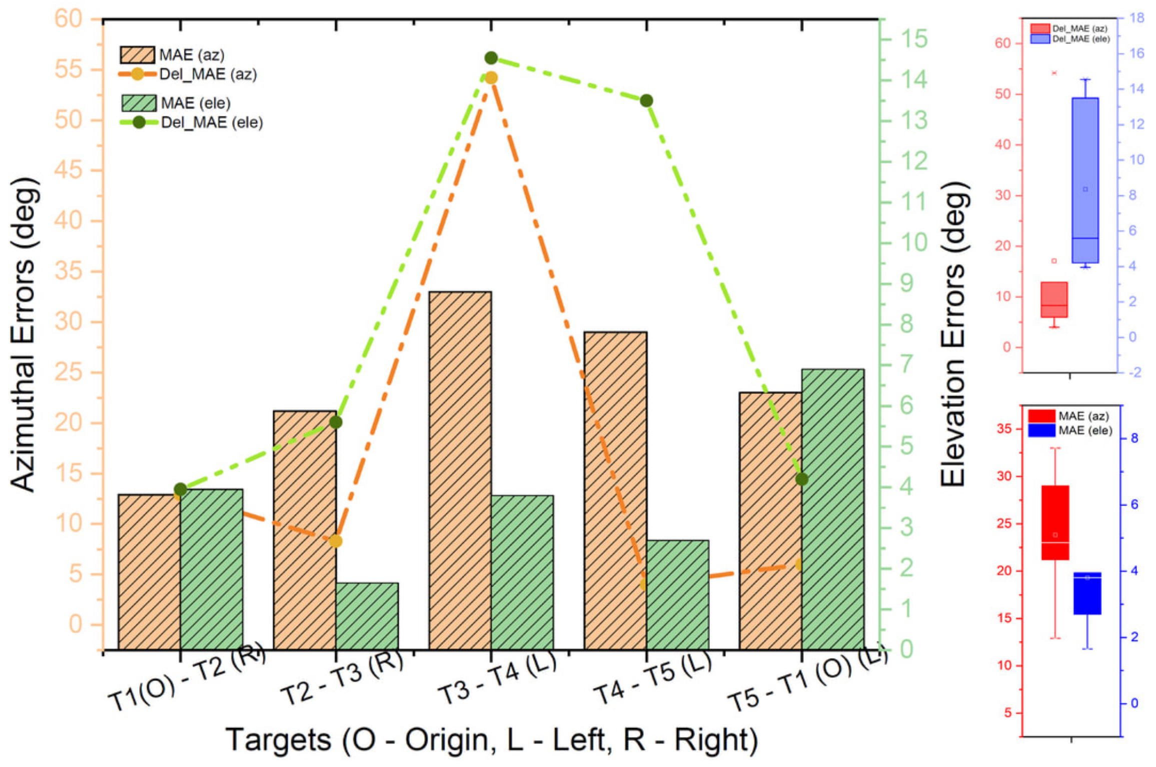


Figure 6.7: Madgwick filter performance (real-time) in tracking head-pose on the NESL Earable Mobility Corpus. The box plots showcase the error distribution.

The goal of the corpus was to use the headpose information to render binaural audio. The obtained orientation estimation errors were well within the human binaural sound localization range (±20 degrees). This was confirmed via our real-time localization test, where a subject aims to point out where a particular sound source might be based on hearing, illustrated in Figure 6.8. We observed a median range of 40 degrees, which is ±20 degrees of localization accuracy of humans. This complements the error of our filter, which is much less than ±20 degrees.

The memory and flash requirements of the Madgwick filter (implemented in Arduino IDE) for various devices, along with MPU-9250 IMU library, are as follows:

- STM32L476RG Nucleo: 1.64 kB SRAM (1.28%), 41 kB flash (4%).
- STM32F746 Discovery: 1.69 kB SRAM (0.5%), 42 kB flash (4%).

- STM32 Nucleo-144: 1.66 kB SRAM (0.16%), 44 kB flash (2%).
- Arduino BLE 33: 47 kB SRAM (18%), 110 kB flash (11%).
- Arduino Uno R3 ATMega328p: 1.34 KB SRAM (67%), 20 kB flash (63%).

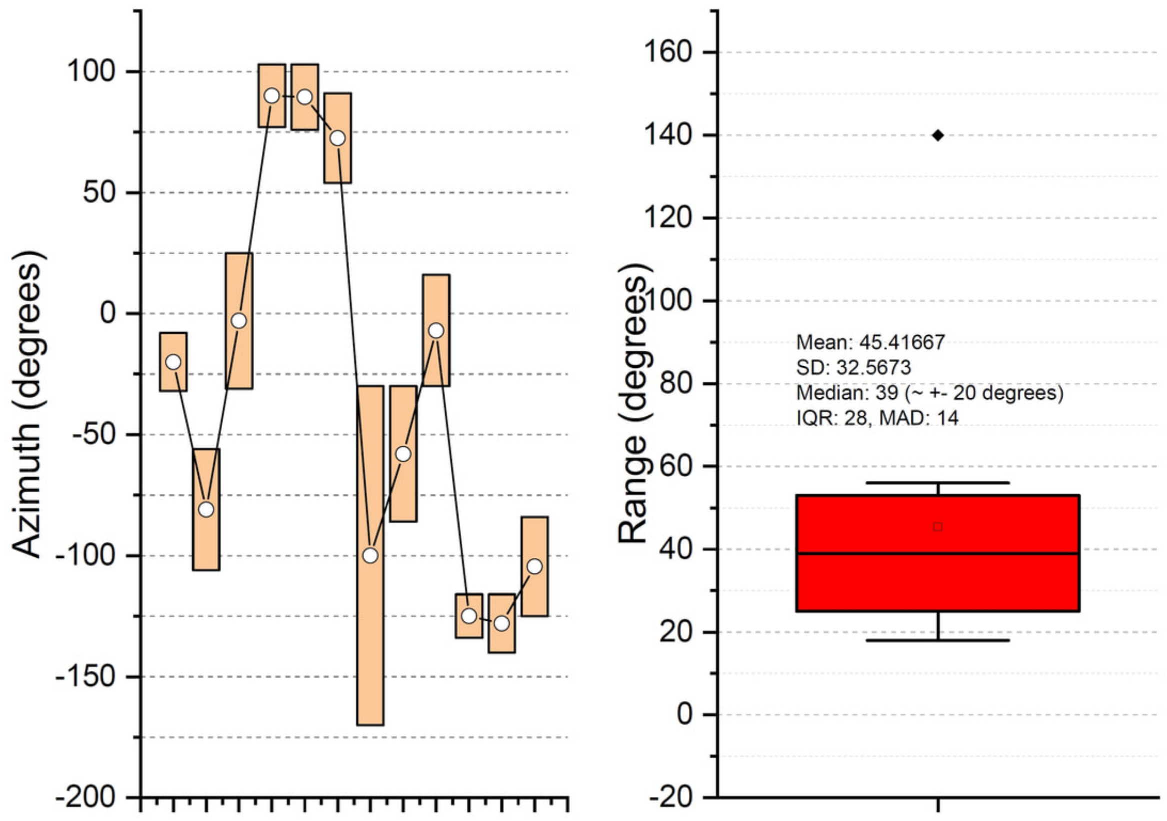


Figure 6.8: Sound localization test with real subjects to model performance of the filter.

The memory and flash requirements of the depth/altitude filter (implemented in Mbed Studio and Arduino IDE) for various devices are as follows:

- STM32L476RG Nucleo (Mbed Studio): 10.3 kB SRAM (8%), 48 kB flash (5%).
- STM32F746 Discovery(Mbed Studio): 25.7 kB SRAM (8%), 56.1 kB flash (5.6%).
- Arduino BLE 33 (Arduino IDE): 44 kB SRAM (17%) , 86 kB (8%)

CHAPTER 7

Conclusion and Future Work

In this thesis, we introduced a robust learning pipeline for hierarchical deep neural architectures to handle complex activity data from inertial sensors in the presence of missing data, misaligned samples, and variable sampling rates. We evaluate the applicability of the time-shift data augmentation, controlled window jitter, and masks to handle the hurdles mentioned above, benchmarking our pipeline using a hierarchical recurrent-convolutional architecture with DCBL neural blocks. Our evaluations yield 11% and 24% performance improvement for macro and micro-activity recognition over vanilla architectures, achieving 88% and 72% classification accuracy respectively for macro and micro-activity classification on the *Cooking Activity Recognition Dataset with Macro and Micro Activities*, exceeding performance over state-of-the-art vanilla activity classifiers. Our proposed uncertainty aware pipeline achieved 3rd place in the *2020 Cooking Activity Recognition Challenge* and was the best DL-based complex event processor among 78 teams. We showed that complex multimodal data-processing architectures require time-awareness and injection of runtime sensing uncertainty metadata into the training pipeline to take advantage of information-rich high-dimensional context.

In addition, we presented an efficient neural inertial navigation framework that forms the basis and starting point for real-time sub-meter neural-inertial localization on resource-constrained edge devices void of infrastructure-dependent position fixes or motion heuristics that can run smoothly on a wide spectrum of target hardware. We showed the superiority of TCN for time-series processing in dead-reckoning over recurrent architectures, achieving gains not only in terms of error metrics, but also in terms of resource usage and optimization advantages. We formulated a simple and fast hardware-aware Bayesian NAS algorithm that takes ∼12-24 hours to train and search, which is faster than existing NAS algorithms, while achieving comparable pareto-optimal accuracy and resource usage reduction metrics with SOTA NAS algorithms built for microcontrollers. The networks found via NAS not only fit on target hardware, but also achieve pareto-optimal error metrics with the backbone network while requiring upto 3× less SRAM and 5.3× less flash. Our formulation of NAS is regularized by target hardware constraints, and uses different techniques for distinct target hardware. Compared to SOTA, our network requires 2.5× less SRAM and 6× less Flash, while achieving similar sub-meter accuracy. The optimized model on our designed hardware consumes only 56 kB SRAM and 117 kB flash for the *OxIOD Dataset*.

We also presented an ultra-low-power, low-cost, modular, self-sustaining, durable, light and high-resolution inertial hardware framework capable of running neural inertial navigation algorithms in real-time that pushed the SOTA in odometry hardware. We benchmarked the performance of Madgwick filter on the *NESL Earable Mobility Corpus* for orientation and headpose estimation, obtaining a median delta motion errors of 8.3 and 5.6 degrees in azimuth and elevation planes, outperforming SOTA while consuming only 1% SRAM and 4% flash on generic Cortex M4 class devices. We proposed a robust altitude filter with thermocline, salinity, noise and timestamp jitter mitigation, which consumed on average only 8% SRAM and 5% flash on generic Cortex M4 class devices.

7.1 Future Work

There are several future directions for our work concerning complex activity recognition. While we have focused on online learning from missing data, an alternative approach is to impute the missing samples synthetically using GAN [12][13][14], followed by training. Synthetic data generation may be useful when analyzing microscopic granularities in temporal streams as well as data augmentation. In particular, incorporation of information-rich and naturally & temporally aligned mo-cap data requires data generation for the inertial sensor domain. In addition, the proposed training pipeline can be benchmarked on other temporal HAR modalities other than inertial samples such as mmWave ambient sensing [213] as well as sensor placement and availability constraints requiring spatially independent approaches [9][10][214]. In addition, we could attempt to replace the DCBL neural blocks in hierarchical DNN architecture with more efficient TCN blocks for efficient complex activity recognition. In terms of human-AI interaction, a tool that allows layman users to use our uncertainty mitigation techniques using a graphical user interface may be useful [1].

For the efficient neural inertial navigation framework, scope of future work include:

- Systematic implementation and comparison with more localization and NAS baselines on different datasets apart from PDR.
- HIL energy, SRAM, flash and latency profiling on real-hardware during the NAS epochs rather than profiling post-NAS.
- Implement a parallel NAS formulation using Mango.
- Include quantization in the NAS formulation.
- Cross-discipline and cross-domain dead-reckoning (e.g., across land, air, sea and various sensors/sensor positions).
- Explore Sim2Real possiblities.
- Check if orientation inputs can make the neural network aware of current coordinate frame to allow for completely random sensor orientations.
- Explore whether magnetometer fusion results in heading RMSE improvement.

We should also benchmark the performance improvement our altitude filter brings in over linear pressure filters, and provide a systematic comparison of our proposed neural inertial hardware with existing hardware.

[1]We envision our pipeline being used in the backend here: `https://github.com/Human-AI-Interaction-Projects/GUI-GAN`

REFERENCES

[1] T. Hossain and S. Inoue, "A Comparative Study on Missing Data Handling Using Machine Learning for Human Activity Recognition", *2019 Joint 8th International Conference on Informatics, Electronics & Vision (ICIEV) and 2019 3rd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*, Spokane, WA, USA, 2019, pp. 124-129.

[2] T. Hossain, H. Goto, M. A. Rahman Ahad and S. Inoue, "A Study on Sensor-based Activity Recognition Having Missing Data", *2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*, Kitakyushu, Japan, 2018, pp. 556-561.

[3] T. Xing, M. Roig Vilamala, L. Garcia, F. Cerutti, L. Kaplan, A. Preece and M. Srivastava, "DeepCEP: Deep Complex Event Processing Using Distributed Multimodal Information", *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, Washington, DC, USA, 2019, pp. 87-92.

[4] J. V. Jeyakumar, E.S. Lee, Z. Xia, S. S. Sandha, N. Tausik and M. Srivastava, "Deep Convolutional Bidirectional LSTM Based Transportation Mode Recognition." *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers.* 2018.

[5] L. Wang, H. Gjoreski, M. Ciliberto, P. Lago, K. Murao, T. Okita and D. Roggen, "Summary of the Sussex-Huawei Locomotion-Transportation Recognition Challenge 2019." *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers.* 2019.

[6] Z. Che, S. Purushotham, K. Cho, D. Sontag and Y. Liu, "Recurrent Neural Networks for Multivariate Time Series with Missing Values." *Sci Rep 8*, 6085. 2018.

[7] J. Yoon, W. R. Zame and M. van der Schaar, "Estimating Missing Data in Temporal Data Streams Using Multi-Directional Recurrent Neural Networks", in *IEEE Transactions on Biomedical Engineering*, vol. 66, no. 5, pp. 1477-1490, May 2019.

[8] R. Ohmura and R. Uchida. "Exploring Combinations of Missing Data Complement for Fault Tolerant Activity Recognition." *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication.* 2014.

[9] J.V. Jeyakumar, L. Lai, N. Suda and M. Srivastava. "SenseHAR: A Robust Virtual Activity Sensor for Smartphones and Wearables." *Proceedings of the 17th Conference on Embedded Networked Sensor Systems (SenSys).* 2019.

[10] T. Xing, S. S. Sandha, B. Balaji, S. Chakraborty, and M. Srivastava. "Enabling Edge Devices that Learn from Each Other: Cross Modal Training for Activity Recognition." *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*, pp. 37-42. 2018.

[11] Liu, S., Yao, S., Huang, Y., Liu, D., Shao, H., Zhao, Y., ... & Abdelzaher, T. (2020). "Handling Missing Sensors in Topology-Aware IoT Applications with Gated Graph Neural Network." *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(3), 1-31.

[12] J. Yoon, J. Jordon and M. van der Schaar. GAIN: Missing Data Imputation using Generative Adversarial Nets. *Proceedings of the 35th International Conference on Machine Learning, in PMLR* 80:5689-5698. 2018.

[13] S. C. X. Li, B. Jiang and B. M. Marlin. MisGAN: Learning from Incomplete Data with Generative Adversarial Networks. *7th International Conference on Learning Representations (ICLR)*. 2019.

[14] M. Alzantot, S. Chakraborty and M. Srivastava, SenseGen: A Deep Learning Architecture for Synthetic Sensor Data Generation, *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Kona, HI, 2017, pp. 188-193.

[15] Saha S.S., Sandha S.S., Srivastava M. (2020) "Deep Convolutional Bidirectional LSTM for Complex Activity Recognition with Missing Data." In: Ahad M.A.R., Lago P., Inoue S. (eds) *Human Activity Recognition Challenge. Smart Innovation, Systems and Technologies*, vol 199. Springer, Singapore, 39-53.

[16] Dhar, S., Guo, J., Liu, J., Tripathi, S., Kurup, U., & Shah, M. (2019). "On-device Machine Learning: An Algorithms and Learning Theory Perspective." *arXiv preprint* arXiv:1911.00623.

[17] Benmeziane, H., Maghraoui, K. E., Ouarnoughi, H., Niar, S., Wistuba, M., & Wang, N. (2021). "A Comprehensive Survey on Hardware-Aware Neural Architecture Search." *arXiv preprint* arXiv:2101.09336.

[18] Elsken, T., Metzen, J. H., & Hutter, F. (2019). "Neural architecture search: A survey." *J. Mach. Learn. Res.*, 20(55), 1-21.

[19] B. Zoph and Q. V. Le. "Neural Architecture Search with Reinforcement Learning." *International Conference on Learning Representations (ICLR)* (2017).

[20] Baker, B., Gupta, O., Naik, N., & Raskar, R. (2016). "Designing neural network architectures using reinforcement learning." *International Conference on Learning Representations (ICLR)* (2017).

[21] Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). "Learning Transferable Architectures for Scalable Image Recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8697-8710).

[22] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). "MNASNet: Platform-aware Neural Architecture Search for Mobile." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 2820-2828).

[23] Cai, H., Gan, C., Wang, T., Zhang, Z., & Han, S. (2020). "Once-for-all: Train One Network and Specialize it for Efficient Deployment." *International Conference on Learning Representations (ICLR)*, 2020.

[24] Lin, J., Chen, W. M., Lin, Y., Cohn, J., Gan, C., & Han, S. (2020). "MCUNet: Tiny Deep Learning on IoT Devices." *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, Vancouver, Canada.

[25] Liu, H., Simonyan, K., & Yang, Y. (2018). "DARTS: Differentiable Architecture Search." *International Conference on Learning Representations (ICLR)*. 2018.

[26] Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., ... & Keutzer, K. (2019). "FBNet: Hardware-aware Efficient Convnet Design via Differentiable Neural Architecture Search." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 10734-10742).

[27] Cai, H., Zhu, L., & Han, S. (2018). "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. *International Conference on Learning Representations (ICLR)*. 2018.

[28] Banbury, C., Zhou, C., Fedorov, I., Matas, R., Thakker, U., Gope, D., ... & Whatmough, P. (2021). "MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers." *Proceedings of Machine Learning and Systems*, 3.

[29] Fedorov, I., Adams, R. P., Mattina, M., & Whatmough, P. N. (2019). "SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers." *Advances in Neural Information Processing Systems, 32.*

[30] Kumar, A., Goyal, S., & Varma, M. (2017, July). "Resource-efficient Machine Learning in 2 kB RAM for the Internet of Things." In *International Conference on Machine Learning* (pp. 1935-1944). PMLR.

[31] Gupta, C., Suggala, A. S., Goyal, A., Simhadri, H. V., Paranjape, B., Kumar, A., ... & Jain, P. (2017, July). "ProtoNN: Compressed and Accurate kNN for Resource-Scarce Devices." In *International Conference on Machine Learning* (pp. 1331-1340). PMLR.

[32] Sanchez-Iborra, R., & Skarmeta, A. F. (2020). "TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities." *IEEE Circuits and Systems Magazine*, 20(3), 4-18.

[33] Denton, E., Zaremba, W., Bruna, J., LeCun, Y., & Fergus, R. (2014, December). "Exploiting Linear Structure within Convolutional Networks for Efficient Evaluation." *In Proceedings of the 27th International Conference on Neural Information Processing Systems*-Volume 1 (pp. 1269-1277).

[34] Lee, D. D., & Seung, H. S. (1999). "Learning the Parts of Objects by Non-Negative Matrix Factorization." *Nature*, 401(6755), 788-791.

[35] Balakrishnama, S., & Ganapathiraju, A. (1998). "Linear Discriminant Analysis-A Brief Tutorial." *Institute for Signal and information Processing*, 18(1998), 1-8.

[36] McInnes, L., Healy, J., Saul, N., & Großberger, L. (2018). "UMAP: Uniform Manifold Approximation and Projection." *Journal of Open Source Software*, 3(29), 861.

[37] Hinton, G., & Roweis, S. T. (2002, December). "Stochastic Neighbor Embedding." In *NIPS* (Vol. 15, pp. 833-840).

[38] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing. Vol 1: Foundations.* MIT Press, Cambridge, MA, 1986.

[39] Ballard, Dana H. "Modular Learning in Neural Networks." *Proceedings of the Sixth National Conference on Artificial Intelligence-Volume 1.* 1987.

[40] Han, S., Mao, H., & Dally, W. J. (2016). "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding." *International Conference on Learning Representations (ICLR)* (2016).

[41] Han, S., Pool, J., Tran, J., & Dally, W. J. (2015, December). "Learning Both Weights and Connections for Efficient Neural Networks. *In Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 1* (pp. 1135-1143).

[42] Gupta, S., Agrawal, A., Gopalakrishnan, K., & Narayanan, P. (2015, June). Deep Learning with Limited Numerical Precision. *In International conference on machine learning* (pp. 1737-1746). PMLR.

[43] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018)." Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 2704-2713).

[44] Krishnamoorthi, R. (2018). "Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper." *arXiv preprint* arXiv:1806.08342.

[45] Wang, K., Liu, Z., Lin, Y., Lin, J., & Han, S. (2019). HAQ: Hardware-aware Automated Quantization with Mixed Precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 8612-8620).

[46] Rusci, M., Capotondi, A., & Benini, L. (2020). "Memory-driven Mixed Low Precision Quantization for Enabling Deep Network Inference on Microcontrollers." *Proceedings of the 3rd MLSys Conference*, Austin, TX, USA, 2020.

[47] Capotondi, A., Rusci, M., Fariselli, M., & Benini, L. (2020). "CMix-NN: Mixed Low-Precision CNN Library for Memory-constrained Edge Devices." *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(5), 871-875.

[48] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). "SqueezeNet: AlexNet-level Accuracy with 50× Fewer Parameters and <0.5 MB Model Size." *arXiv preprint* arXiv:1602.07360.

[49] Hu, J., Shen, L., & Sun, G. (2018). "Squeeze-and-excitation Networks." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 7132-7141).

[50] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." *arXiv preprint* arXiv:1704.04861.

[51] Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 6848-6856).

[52] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). "Densely Connected Convolutional Networks." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 4700-4708).

[53] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). "Mobilenetv2: Inverted Residuals and Linear Bottlenecks." In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (pp. 4510-4520).

[54] Tan, M., Pang, R., & Le, Q. V. (2020). "EfficientDet: Scalable and Efficient Object Detection." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 10781-10790).

[55] Wang, C. Y., Bochkovskiy, A., & Liao, H. Y. M. (2020). "Scaled-YOLOv4: Scaling Cross Stage Partial Network." *arXiv preprint* arXiv:2011.08036.

[56] Wang, C. Y., Liao, H. Y. M., Wu, Y. H., Chen, P. Y., Hsieh, J. W., & Yeh, I. H. (2020). "CSPNet: A New Backbone that can Enhance Learning Capability of CNN." In Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition Workshops (pp. 390-391).

[57] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). "You Only Look Once: Unified, Real-time Object Detection." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 779-788).

[58] Kusupati, A., Singh, M., Bhatia, K., Kumar, A., Jain, P., & Varma, M. (2018, December). "FastGRNN: a Fast, Accurate, Stable and Tiny Kilobyte Sized Gated Recurrent Neural Network. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (pp. 9031-9042).

[59] Dennis, D. K., Pabbaraju, C., Simhadri, H. V., & Jain, P. (2018, December). "Multiple Instance Learning for Efficient Sequential Data Classification on Resource-constrained Devices." In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (pp. 10976-10987).

[60] Dennis, D. K., Acar, D. A. E., Mandikal, V. M., Sadasivan, V. S., Saligrama, V., Simhadri, H. V., & Jain, P. (2019). "Shallow RNN: Accurate Time-Series Classification on Resource Constrained Devices." In *Proceedings of the 34th International Conference on Neural Information Processing Systems*

[61] Thakker, U., Fedorov, I., Beu, J., Gope, D., Zhou, C., Dasika, G., & Mattina, M. (2019). "Pushing the Limits of RNN Compression. ", *Workshop on Energy Efficient Machine Learning and Cognitive Computing, NEURIPS* 2019.

[62] Warden, P., & Situnayake, D. (2019). "TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers." O'Reilly Media, Inc.

[63] David, R., Duke, J., Jain, A., Reddi, V. J., Jeffries, N., Li, J., ... & Warden, P. (2021). "TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems." *Proceedings of Machine Learning and Systems 3 pre-proceedings (MLSys 2021).*

[64] Tan, N. (Accessed: May 30, 2021). "uTensor: TinyML AI Inference Library.", Available at: https://github.com/uTensor/uTensor.

[65] Lai, L., Suda, N., & Chandra, V. (2018). "CMSIS-NN: Efficient Neural Network Kernels for ARM Cortex-M CPUs." *arXiv preprint* arXiv:1801.06601.

[66] Saha, O., Kusupati, A., Simhadri, H. V., Varma, M., & Jain, P. (2020). "RNNPool: Efficient Non-Linear Pooling for RAM Constrained Inference." *Advances in Neural Information Processing Systems 33* (2020).

[67] Goyal, S., Raghunathan, A., Jain, M., Simhadri, H. V., & Jain, P. (2020, November). "DROCC: Deep Robust One-Class Classification." In *International Conference on Machine Learning* (pp. 3711-3721). PMLR.

[68] STMicroelectronics. (Accessed: May 30, 2021). "X-CUBE-AI - AI expansion pack for STM32CubeMX - STMicroelectronics.", Available at: https://www.st.com/en/embedded-software/x-cube-ai.html.

[69] da Silva, L. T., Souza, V. M., & Batista, G. E. (2019, November). "EmbML Tool: Supporting the Use of Supervised Learning Algorithms in Low-Cost Embedded Systems." In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)* (pp. 1633-1637). IEEE.

[70] Wang, X., Magno, M., Cavigelli, L., & Benini, L. (2020). "FANN-on-MCU: An Open-Source Toolkit for Energy-Efficient Neural Network Inference at the Edge of the Internet of Things." *IEEE Internet of Things Journal*, 7(5), 4403-4417.

[71] Walker, J. S., Jones, M. W., Laramee, R. S., Holton, M. D., Shepard, E. L., Williams, H. J., ... & Wilson, R. P. (2015). "Prying into the intimate secrets of animal lives; software beyond hardware for comprehensive annotation in 'Daily Diary'tags." *Movement Ecology*, 3(1), 1-16.

[72] Wilson, R. P., Liebsch, N., Davies, I. M., Quintana, F., Weimerskirch, H., Storch, S., ... & McMahon, C. R. (2007). "All at sea with animal tracks; methodological and analytical solutions for the resolution of movement." *Deep Sea Research Part II: Topical Studies in Oceanography*, 54(3-4), 193-210.

[73] Shen, S., Gowda, M., & Roy Choudhury, R. (2018, October). "Closing the gaps in inertial motion tracking." In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking* (pp. 429-444).

[74] Yang, Z., Wu, C., Zhou, Z., Zhang, X., Wang, X., & Liu, Y. (2015). "Mobility increases localizability: A survey on wireless indoor localization using inertial sensors." *ACM Computing Surveys* (CSUR), 47(3), 1-34.

[75] Yan, H., Shan, Q., & Furukawa, Y. (2018). "RIDI: Robust IMU Double Integration." In *Proceedings of the European Conference on Computer Vision* (ECCV) (pp. 621-636).

[76] Kok, M., Hol, J. D., & Schön, T. B. (2017). "Using Inertial Sensors for Position and Orientation Estimation." *Foundations and Trends in Signal Processing*, 11(1-2), 1-153.

[77] Borenstein, J., Ojeda, L., & Kwanmuang, S. (2009, May). "Heuristic reduction of gyro drift in IMU-based personnel tracking systems." In *Optics and Photonics in Global Homeland Security V and Biometric Technology for Human Identification VI* (Vol. 7306, p. 73061H). International Society for Optics and Photonics.

[78] Prikhodko, I. P., Bearss, B., Merritt, C., Bergeron, J., & Blackmer, C. (2018, March). "Towards self-navigating cars using MEMS IMU: Challenges and opportunities." In *2018 IEEE International Symposium on Inertial Sensors and Systems* (INERTIAL) (pp. 1-4). IEEE.

[79] Frosio, I., Pedersini, F., & Borghese, N. A. (2008). "Autocalibration of MEMS accelerometers." *IEEE Transactions on Instrumentation and Measurement*, 58(6), 2034-2041.

[80] Zhou, P., Li, M., & Shen, G. (2014, September). "Use it free: Instantly knowing your phone attitude." In *Proceedings of the 20th annual international conference on Mobile computing and networking* (pp. 605-616).

[81] Lötters, J. C., Schipper, J., Veltink, P. H., Olthuis, W., & Bergveld, P. (1998). "Procedure for in-use calibration of triaxial accelerometers in medical applications." *Sensors and Actuators A: Physical*, 68(1-3), 221-228.

[82] Herath, S., Yan, H., & Furukawa, Y. (2020, May). "RoNIN: Robust Neural Inertial Navigation in the Wild: Benchmark, Evaluations, & New Methods." In *2020 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 3146-3152). IEEE.

[83] Goyal, P., Ribeiro, V. J., Saran, H., & Kumar, A. (2011, September). "Strap-down pedestrian dead-reckoning system." In *2011 international conference on indoor positioning and indoor navigation* (pp. 1-7). IEEE.

[84] Hou, X., & Bergmann, J. (2020). "Pedestrian Dead Reckoning with Wearable Sensors: A Systematic Review." *IEEE Sensors Journal*, 21(1), 143-152.

[85] Wu, Y., Zhu, H. B., Du, Q. X., & Tang, S. M. (2019). "A survey of the research status of pedestrian dead reckoning systems based on inertial sensors." *International Journal of Automation and Computing*, 16(1), 65-83.

[86] Aqel, M. O., Marhaban, M. H., Saripan, M. I., & Ismail, N. B. (2016). "Review of visual odometry: types, approaches, challenges, and applications." *SpringerPlus*, 5(1), 1-26.

[87] Kuutti, S., Fallah, S., Katsaros, K., Dianati, M., Mccullough, F., & Mouzakitis, A. (2018). "A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications."*IEEE Internet of Things Journal*, 5(2), 829-846.

[88] Crassidis, J. L., Markley, F. L., & Cheng, Y. (2007). "Survey of nonlinear attitude estimation methods." *Journal of guidance, control, and dynamics*, 30(1), 12-28.

[89] Giannitrapani, A., Ceccarelli, N., Scortecci, F., & Garulli, A. (2011). "Comparison of EKF and UKF for spacecraft localization via angle measurements." *IEEE Transactions on aerospace and electronic systems*, 47(1), 75-84.

[90] Narayana, S., Prasad, R. V., Rao, V., Mottola, L., & Prabhakar, T. V. (2020, April). "Hummingbird: energy efficient GPS receiver for small satellites." In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking* (pp. 1-13).

[91] Wilson, R. P., Shepard, E. L. C., & Liebsch, N. (2008). "Prying into the intimate details of animal lives: use of a daily diary on animals." *Endangered species research*, 4(1-2), 123-137.

[92] Wilson, R. P., Holton, M. D., di Virgilio, A., Williams, H., Shepard, E. L., Lambertucci, S., ... & Duarte, C. M. (2018). "Give the machine a hand: A Boolean time-based decision-tree template for rapidly finding animal behaviours in multisensor data." *Methods in Ecology and Evolution*, 9(11), 2206-2215.

[93] Harle, R. (2013). "A survey of indoor inertial positioning systems for pedestrians." *IEEE Communications Surveys & Tutorials*, 15(3), 1281-1293.

[94] Foxlin, E. (2005). "Pedestrian tracking with shoe-mounted inertial sensors." *IEEE Computer graphics and applications*, 25(6), 38-46.

[95] Jiménez, A. R., Seco, F., Prieto, J. C., & Guevara, J. (2010, March). "Indoor pedestrian navigation using an INS/EKF framework for yaw drift reduction and a foot-mounted IMU." In 2010 *7th Workshop on Positioning, Navigation and Communication* (pp. 135-143). IEEE.

[96] Alzantot, M., & Youssef, M. (2012, April). "UPTIME: Ubiquitous pedestrian tracking using mobile phones." In *2012 IEEE Wireless Communications and Networking Conference (WCNC)* (pp. 3204-3209). IEEE.

[97] Wang, H., Sen, S., Elgohary, A., Farid, M., Youssef, M., & Choudhury, R. R. (2012, June). "No need to war-drive: Unsupervised indoor localization." In *Proceedings of the 10th international conference on Mobile systems, applications, and services* (pp. 197-210).

[98] Levi, R. W., & Judd, T. (1996). U.S. Patent No. 5,583,776. Washington, DC: U.S. Patent and Trademark Office.

[99] Kalman, R. E. (March 1, 1960). "A New Approach to Linear Filtering and Prediction Problems." *ASME. J. Basic Eng.* March 1960; 82(1): 35–45.

[100] Al-Sharman, M. K., Zweiri, Y., Jaradat, M. A. K., Al-Husari, R., Gan, D., & Seneviratne, L. D. (2019). "Deep-learning-based neural network training for state estimation enhancement: Application to attitude estimation." *IEEE Transactions on Instrumentation and Measurement*, 69(1), 24-34.

[101] Chen, H., Aggarwal, P., Taha, T. M., & Chodavarapu, V. P. (2018, July). :Improving inertial sensor by reducing errors using deep learning methodology." In *NAECON 2018-IEEE National Aerospace and Electronics Conference* (pp. 197-202). IEEE.

[102] Brossard, M., Barrau, A., & Bonnabel, S. (2020). "AI-IMU dead-reckoning." *IEEE Transactions on Intelligent Vehicles*, 5(4), 585-595.

[103] Brossard, M., Bonnabel, S., & Barrau, A. (2020). "Denoising IMU Gyroscopes with Deep Learning for Open-Loop Attitude Estimation." *IEEE Robotics and Automation Letters*, 5(3), 4796-4803.

[104] Tekinalp, O., & Ozemre, M. (2001, August). "Artificial neural networks for transfer aligment and calibration of inertial navigation systems." In *AIAA Guidance, Navigation, and Control Conference and Exhibit* (p. 4406).

[105] Wu, F., Luo, H., Jia, H., Zhao, F., Xiao, Y., & Gao, X. (2020). "Predicting the Noise Covariance With a Multitask Learning Model for Kalman Filter-Based GNSS/INS Integrated Navigation." *IEEE Transactions on Instrumentation and Measurement*, 70, 1-13.

[106] Xu, F., & Fang, J. (2008). "Velocity and position error compensation using strapdown inertial navigation system/celestial navigation system integration based on ensemble neural network." *Aerospace Science and Technology*, 12(4), 302-307.

[107] Zhao, X., Deng, C., Kong, X., Xu, J., & Liu, Y. "Learning to Compensate for the Drift and Error of Gyroscope in Vehicle Localization." In *2020 IEEE Intelligent Vehicles Symposium (IV)* (pp. 852-857). IEEE.

[108] Brossard, M., & Bonnabel, S. (2019, May). "Learning wheel odometry and IMU errors for localization." In 2019 *International Conference on Robotics and Automation* (ICRA) (pp. 291-297). IEEE.

[109] Gao, X., Luo, H., Ning, B., Zhao, F., Bao, L., Gong, Y., ... & Jiang, J. (2020). "RL-AKF: An adaptive Kalman filter navigation algorithm based on reinforcement learning for ground vehicles." *Remote Sensing*, 12(11), 1704.

[110] Brossard, M., Barrau, A., & Bonnabel, S. (2019, November). "RINS-W: Robust Inertial Navigation System on Wheels." In 2019 *IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS) (pp. 2068-2075). IEEE.

[111] Graves, A., & Schmidhuber, J. (2005). "Framewise phoneme classification with bidirectional LSTM and other neural network architectures." *Neural networks*, 18(5-6), 602-610.

[112] Abbeel, P., Coates, A., Montemerlo, M., Ng, A. Y., & Thrun, S. (2005, June). "Discriminative Training of Kalman Filters." In *Robotics: Science and systems* (Vol. 2, p. 1).

[113] Wagstaff, B., & Kelly, J. (2018, September). "LSTM-based zero-velocity detection for robust inertial navigation." In 2018 *International Conference on Indoor Positioning and Indoor Navigation* (IPIN) (pp. 1-8). IEEE.

[114] Wagstaff, B., Peretroukhin, V., & Kelly, J. (2019). "Robust data-driven zero-velocity detection for foot-mounted inertial navigation." *IEEE Sensors Journal*, 20(2), 957-967.

[115] Beauregard, S. (2006, March). "A helmet-mounted pedestrian dead reckoning system." In *3rd International Forum on Applied Wearable Computing* 2006 (pp. 1-11). VDE.

[116] Cortés, S., Solin, A., & Kannala, J. (2018, September). "Deep learning based speed estimation for constraining strapdown inertial navigation on smartphones." In 2018 IEEE 28th *International Workshop on Machine Learning for Signal Processing* (MLSP) (pp. 1-6). IEEE.

[117] Liu, W., Caruso, D., Ilg, E., Dong, J., Mourikis, A. I., Daniilidis, K., ... & Engel, J. (2020). "TLIO: Tight Learned Inertial Odometry." *IEEE Robotics and Automation Letters*, 5(4), 5653-5660.

[118] Song, S., Liu, J., Guo, J., Wang, J., Xie, Y., & Cui, J. H. (2020). "Neural-Network-Based AUV Navigation for Fast-Changing Environments." *IEEE Internet of Things Journal*, 7(10), 9773-9783.

[119] Xie, Y. X., Liu, J., Hu, C. Q., Cui, J. H., & Xu, H. (2016, October). "AUV dead-reckoning navigation based on neural network using a single accelerometer." In P*roceedings of the 11th ACM International Conference on Underwater Networks & Systems* (pp. 1-5).

[120] Chen, C., Lu, X., Markham, A., & Trigoni, N. (2018, April). "IONet: Learning to cure the curse of drift in inertial odometry." In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 32, No. 1).

[121] Diamant, R., & Jin, Y. (2013). "A machine learning approach for dead-reckoning navigation at sea using a single accelerometer." I*EEE Journal of Oceanic Engineering*, 39(4), 672-684.

[122] Edel, M., & Köppe, E. (2015, October). "An advanced method for pedestrian dead reckoning using BLSTM-RNNs." In *2015 International Conference on Indoor Positioning and Indoor Navigation* (IPIN) (pp. 1-6). IEEE.

[123] Esfahani, M. A., Wang, H., Wu, K., & Yuan, S. (2019). "AbolDeepIO: A novel deep inertial odometry network for autonomous vehicles." *IEEE Transactions on Intelligent Transportation Systems*, 21(5), 1941-1950.

[124] Zhang, X., He, B., Li, G., Mu, X., Zhou, Y., & Mang, T. (2020). "NavNet: AUV Navigation Through Deep Sequential Learning." *IEEE Access*, 8, 59845-59861.

[125] Gao, R., Xiao, X., Zhu, S., Xing, W., Li, C., Liu, L., ... & Chai, H. (2021). "Glow in the Dark: Smartphone Inertial Odometry for Vehicle Tracking in GPS Blocked Environments." *IEEE Internet of Things Journal*.

[126] Klein, I., & Asraf, O. (2020). "StepNet—Deep learning approaches for step length estimation." *IEEE Access*, 8, 85706-85713.

[127] Silva do Monte Lima, J. P., Uchiyama, H., & Taniguchi, R. I. (2019). "End-to-end learning framework for imu-based 6-dof odometry." *Sensors*, 19(17), 3777.

[128] Esfahani, M. A., Wang, H., Wu, K., & Yuan, S. (2019). "OriNet: Robust 3-D orientation estimation with a single particular IMU." *IEEE Robotics and Automation Letters*, 5(2), 399-406.

[129] Chen, C., Miao, Y., Lu, C. X., Xie, L., Blunsom, P., Markham, A., & Trigoni, N. (2019, July). "Motiontransformer: Transferring neural inertial tracking between domains." In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 33, No. 01, pp. 8009-8016).

[130] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017, December). "Attention is all you need." In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (pp. 6000-6010).

[131] Chen, C., Zhao, P., Lu, C. X., Wang, W., Markham, A., & Trigoni, N. (2020). "Deep-Learning-Based Pedestrian Inertial Navigation: Methods, Data Set, and On-Device Inference." *IEEE Internet of Things Journal*, 7(5), 4431-4441.

[132] Sandha, S. S., Noor, J., Anwar, F. M., & Srivastava, M. (2020, April). "Time awareness in deep learning-based multimodal fusion across smartphone platforms." In 2020 *IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation* (IoTDI) (pp. 149-156). IEEE.

[133] Stisen, A., Blunck, H., Bhattacharya, S., Prentow, T. S., Kjærgaard, M. B., Dey, A., ... & Jensen, M. M. (2015, November). "Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition." In *Proceedings of the 13th ACM conference on embedded networked sensor systems* (pp. 127-140).

[134] Hossain, T., Ahad, M., Rahman, A., & Inoue, S. (2020). "A Method for Sensor-Based Activity Recognition in Missing Data Scenario." *Sensors*, 20(14), 3811.

[135] Sandha, S. S., Noor, J., Anwar, F. M., & Srivastava, M. (2019, September). "Exploiting smartphone peripherals for precise time synchronization." In 2019 *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication* (ISPCS) (pp. 1-6). IEEE.

[136] Ordóñez, F. J., & Roggen, D. (2016). "Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition." *Sensors*, 16(1), 115.

[137] Hammerla, N. Y., Halloran, S., & Plötz, T. (2016, July). "Deep, convolutional, and recurrent models for human activity recognition using wearables." In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence* (pp. 1533-1540).

[138] Ha, S., Yun, J. M., & Choi, S. (2015, October). "Multi-modal convolutional neural networks for activity recognition." In 2015 *IEEE International conference on systems, man, and cybernetics* (pp. 3017-3022). IEEE.

[139] Panwar, M., Dyuthi, S. R., Prakash, K. C., Biswas, D., Acharyya, A., Maharatna, K., ... & Naik, G. R. (2017, July). "CNN based approach for activity recognition using a wrist-worn accelerometer." In 2017 *39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society* (EMBC) (pp. 2438-2441). IEEE.

[140] Zeng, M., Nguyen, L. T., Yu, B., Mengshoel, O. J., Zhu, J., Wu, P., & Zhang, J. (2014, November). "Convolutional neural networks for human activity recognition using mobile sensors." In *6th International Conference on Mobile Computing, Applications and Services* (pp. 197-205). IEEE.

[141] Valenti, R. G., Dryanovski, I., & Xiao, J. (2015). "Keeping a good attitude: A quaternion-based orientation filter for IMUs and MARGs." *Sensors*, 15(8), 19302-19330.

[142] Lea, C., Vidal, R., Reiter, A., & Hager, G. D. (2016, October). "Temporal convolutional networks: A unified approach to action segmentation." In *European Conference on Computer Vision* (pp. 47-54). Springer, Cham.

[143] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... & Kavukcuoglu, K. "WaveNet: A Generative Model for Raw Audio." In *9th ISCA Speech Synthesis Workshop* (pp. 125-125).

[144] Brookner, E. (1998). "Tracking and Kalman Filtering Made Easy." Wiley Online Library.

[145] Fofonoff, N., & Millard Jr, R. (1983). "Algorithms for Computation of Fundamental Properties of Sea Water." *Unesco Tech. Pap.* Mar. Sci., 44.

[146] Madgwick, S. O., Harrison, A. J., & Vaidyanathan, R. (2011, June). "Estimation of IMU and MARG orientation using a gradient descent algorithm." In *2011 IEEE international conference on rehabilitation robotics* (pp. 1-7). IEEE.

[147] Sandha, S. S., Aggarwal, M., Fedorov, I., & Srivastava, M. (2020, May). "Mango: A Python Library for Parallel Hyperparameter Tuning." In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing* (ICASSP) (pp. 3987-3991). IEEE.

[148] Stöckel, T., Jacksteit, R., Behrens, M., Skripitz, R., Bader, R., & Mau-Moeller, A. (2015). "The mental representation of the human gait in young and older adults." *Frontiers in psychology*, 6, 943.

[149] Saha, S. S., Rahman, S., Rasna, M. J., Zahid, T. B., Islam, A. M., & Ahad, M. A. R. (2018). "Feature extraction, performance analysis and system design using the DU mobility dataset." *IEEE Access*, 6, 44776-44786.

[150] Saha, S. S., Rahman, S., Rasna, M. J., Hossain, T., Inoue, S., & Ahad, M. A. R. (2018, October). "Supervised and neural classifiers for locomotion analysis." In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers* (pp. 1563-1570).

[151] Lee, S., & Nirjon, S. (2020, November). "Learning in the Wild: When, How, and What to Learn for On-Device Dataset Adaptation." In *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things* (pp. 34-40).

[152] Cai, H., Gan, C., Zhu, L., & Han, S. (2020). "TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning." *Advances in Neural Information Processing Systems*, 33.

[153] Bai, S., Kolter, J. Z., & Koltun, V. (2018). "Convolutional sequence modeling revisited." *International Conference on Learning Representations (ICLR). 2018.*

[154] Alia, S. S., Lago, P., Takeda, S., Adachi, K., Benaissa, B., Ahad, M. A. R., & Inoue, S. (2021). "Summary of the cooking activity recognition challenge." I*n Human Activity Recognition Challenge* (pp. 1-13). Springer, Singapore.

[155] Lago, P., Takeda, S., Alia, S. S., Adachi, K., Bennai, B., Charpillet, F., & Inoue, S. (2020). "A dataset for complex activity recognition with micro and macro activities in a cooking scenario." *arXiv preprint* arXiv:2006.10681.

[156] Ahad, A. R., Lago, P., & Inoue, S. (2020). "Human Activity Recognition Challenge." Springer Singapore Pte. Limited.

[157] Schubert, D., Goll, T., Demmel, N., Usenko, V., Stückler, J., & Cremers, D. (2018, October). "The TUM VI benchmark for evaluating visual-inertial odometry." In 2018 *IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS) (pp. 1680-1687). IEEE.

[158] Geiger, A., Lenz, P., & Urtasun, R. (2012, June). "Are we ready for autonomous driving? the KITTI vision benchmark suite." In 2012 *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 3354-3361). IEEE.

[159] Jeong, J., Cho, Y., Shin, Y. S., Roh, H., & Kim, A. (2019). "Complex urban dataset with multi-level sensors from highly diverse urban environments." *The International Journal of Robotics Research*, 38(6), 642-657.

[160] Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., ... & Beijbom, O. (2020). "nuScenes: A multimodal dataset for autonomous driving." In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 11621-11631).

[161] Yu, F., Chen, H., Wang, X., Xian, W., Chen, Y., Liu, F., ... & Darrell, T. (2020). "Bdd100k: A diverse driving dataset for heterogeneous multitask learning." In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 2636-2645).

[162] Scale, H. (Accessed: June 2, 2021) "PandaSet: Public Large-Scale Dataset for Autonomous Driving." 2019. https://scale.com/open-datasets/pandaset

[163] Burri, M., Nikolic, J., Gohl, P., Schneider, T., Rehder, J., Omari, S., ... & Siegwart, R. (2016). "The EuRoC micro aerial vehicle datasets." The *International Journal of Robotics Research*, 35(10), 1157-1163.

[164] Ferrera, M., Creuze, V., Moras, J., & Trouvé-Peloux, P. (2019). "AQUALOC: An underwater dataset for visual–inertial–pressure localization." The *International Journal of Robotics Research,* 38(14), 1549-1559.

[165] Simon, D. Levy, (Accessed: June 2, 2021) "TinyEKF: Lightweight C/C++ Extended Kalman Filter with Python for prototyping". Available at: https://github.com/simondlevy/TinyEKF

[166] Kawsar, F., Min, C., Mathur, A., Montanari, A., Acer, U. G., & Van den Broeck, M. (2018, November). "eSense: Open Earable Platform for Human Sensing." In P*roceedings of the 16th ACM Conference on Embedded Networked Sensor Systems* (pp. 371-372).

[167] Yang, Z., Wei, Y. L., Shen, S., & Choudhury, R. R. (2020, September). "Ear-AR: indoor acoustic augmented reality on earphones." In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking* (pp. 1-14).

[168] Choudhury, R. R. (2021, February). "Earable Computing: A New Area to Think About." In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications* (pp. 147-153).

[169] O'Flynn, B., Sanchez-Torres, J., Tedesco, S., & Walsh, M. (2019, December). "Challenges in the Development of Wearable Human Machine Interface Systems." In 2019 *IEEE International Electron Devices Meeting* (IEDM) (pp. 10-4). IEEE.

[170] Jang, J., & Adib, F. (2019). "Underwater backscatter networking." In *Proceedings of the ACM Special Interest Group on Data Communication* (pp. 187-199).

[171] Ghaffarivardavagh, R., Afzal, S. S., Rodriguez, O., & Adib, F. (2020, November). "Underwater Backscatter Localization: Toward a Battery-Free Underwater GPS." *In Proceedings of the 19th ACM Workshop on Hot Topics in Networks* (pp. 125-131).

[172] Iervolino, R., Bonavolontà, F., & Cavallari, A. (2017, September). ”A wearable device for sport performance analysis and monitoring.” In 2017 *IEEE International Workshop on Measurement and Networking* (M&N) (pp. 1-6). IEEE.

[173] Bani, N. A., Hassan, M. Z., Kaidi, H. M., Muhtazaruddin, M. N., Sarip, S., Izhar, M. A. M., ... & Rashidi, A. H. M. (2018, July). ”Harvesting Sustainable Energy from Salt Water: Part I–Effect of Types of Electrodes.” In 2018 *2nd International Conference on Smart Sensors and Application* (ICSSA) (pp. 84-87). IEEE.

[174] Chasteen, S. V., Chasteen, N. D., & Doherty, P. (2008). ”The salty science of the aluminum-air battery.” *The Physics Teacher*, 46(9), 544-547.

[175] Texas Instruments. (Accessed: June 2, 2021) ”BQ25570 Nano Power Boost Charger and Buck Converter for Energy Harvester Powered Applications, 2019.” Available at: https://www.ti.com/product/BQ25570

[176] Lee, E. S., Jeyakumar, J. V., Balaji, B., Wilson, R. P., & Srivastava, M. (2017, November). ”AquaMote: Ultra low power sensor tag for animal localization and fine motion tracking.” In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems* (pp. 1-2).

[177] TE Connenctivity. (Accessed: June 2, 2021) ”MS5837-30BA, Ultra-small, gel-filled, pressure sensor with stainless steel cap, 2019”, Available at: https://www.te.com/usa-en/product-CAT-BLPS0017.html

[178] OriginGPS. (Accessed: June 2, 2021) ”Hornet ORG1411 GPS module with integrated antenna”, Available at: https://origingps.com/products/hornet-org1411/.

[179] Lewkowicz, D., & Delevoye-Turrell, Y. (2016). ”Real-Time Motion Capture Toolbox (RTMocap): an open-source code for recording 3-D motion kinematics to study action–effect anticipations during motor and social interactions.” *Behavior research methods*, 48(1), 366-380.

[180] Ferlini, A., Montanari, A., Mascolo, C., & Harle, R. (2019, September). ”Head motion tracking through in-ear wearables.” In P*roceedings of the 1st International Workshop on Earable Computing* (pp. 8-13).

[181] Höflinger, F., Müller, J., Zhang, R., Reindl, L. M., & Burgard, W. (2013). ”A wireless micro inertial measurement unit (IMU).” *IEEE Transactions on instrumentation and measurement*, 62(9), 2583-2595.

[182] Lara, O. D., & Labrador, M. A. (2012). ”A survey on human activity recognition using wearable sensors.” *IEEE communications surveys & tutorials*, 15(3), 1192-1209.

[183] Ahad, M. A. R., Antar, A. D., & Ahmed, M. (2020). ”IoT sensor-based activity recognition.” Springer.

[184] Reddy, S., Mun, M., Burke, J., Estrin, D., Hansen, M., & Srivastava, M. (2010). ”Using mobile phones to determine transportation modes.” *ACM Transactions on Sensor Networks* (TOSN), 6(2), 1-27.

[185] O'Reilly, M., Caulfield, B., Ward, T., Johnston, W., & Doherty, C. (2018). "Wearable inertial sensor systems for lower limb exercise detection and evaluation: a systematic review." *Sports Medicine*, 48(5), 1221-1246.

[186] Brigante, C. M., Abbate, N., Basile, A., Faulisi, A. C., & Sessa, S. (2011). "Towards miniaturization of a MEMS-based wearable motion capture system." *IEEE Transactions on industrial electronics*, 58(8), 3234-3241.

[187] Oskiper, T., Samarasekera, S., & Kumar, R. (2012, November). "Multi-sensor navigation algorithm using monocular camera, IMU and GPS for large scale augmented reality." In 2012 *IEEE international symposium on mixed and augmented reality* (ISMAR) (pp. 71-80). IEEE.

[188] Bloesch, M., Hutter, M., Hoepflinger, M. A., Leutenegger, S., Gehring, C., Remy, C. D., & Siegwart, R. (2013). "State estimation for legged robots-consistent fusion of leg kinematics and IMU." *Robotics*, 17, 17-24.

[189] Hsu, Y. L., Chou, P. H., Chang, H. C., Lin, S. L., Yang, S. C., Su, H. Y., ... & Kuo, Y. C. (2017). "Design and implementation of a smart home system using multisensor data fusion technology." *Sensors*, 17(7), 1631.

[190] Do, H. M., Pham, M., Sheng, W., Yang, D., & Liu, M. (2018). "RiSH: A robot-integrated smart home for elderly care." *Robotics and Autonomous Systems*, 101, 74-92.

[191] Mordor Intelligence. (Accessed: June 3, 2021) "Inertial Measurement Unit Market - Growth, Trends, and Forecast (2020-2025)." Available at: https://www.mordorintelligence.com/industry-reports/global-inertial-measurement-units-market-industry

[192] Saha, S. S., Rahman, S., Rasna, M. J., Islam, A. M., & Ahad, M. A. R. (2018, June). "DU-MD: An open-source human action dataset for ubiquitous wearable sensors." In *2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)* (pp. 567-572). IEEE.

[193] Badino, H., Yamamoto, A., & Kanade, T. (2013). "Visual odometry by multi-frame feature integration." In *Proceedings of the IEEE International Conference on Computer Vision Workshops* (pp. 222-229).

[194] Chen, L. J., Henawy, J., Kocer, B. B., & Seet, G. G. L. (2019, November). "Aerial Robots On the Way to Underground: An Experimental Evaluation of VINS-Mono on Visual-Inertial Odometry Camera." In 2019 *International Conference on Data Mining Workshops* (ICDMW) (pp. 91-96). IEEE.

[195] Lago, P., Alia, S. S., Takeda, S., Mairittha, T., Mairittha, N., Faiz, F., ... & Inoue, S. (2019, September). "Nurse care activity recognition challenge: summary and results." In *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers* (pp. 746-751).

[196] Alia, S. S., Lago, P., Adachi, K., Hossain, T., Goto, H., Okita, T., & Inoue, S. (2020, September). "Summary of the 2nd nurse care activity recognition challenge using lab and field data." In *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers* (pp. 378-383).

[197] Kaidarova, A., Karimi, M. A., Amara, S., Shamim, A., Gerali, N. R., Duarte, C. M., & Kosel, J. (2018, October). "Sensor for real-time animal condition and movement monitoring." In 2018 *IEEE SENSORS* (pp. 1-4). IEEE.

[198] Kantor, G., Singh, S., Peterson, R., Rus, D., Das, A., Kumar, V., ... & Spletzer, J. (2003). "Distributed search and rescue with robot and sensor teams." *In Field and Service Robotics* (pp. 529-538). Springer, Berlin, Heidelberg.

[199] Waharte, S., & Trigoni, N. (2010, September). "Supporting search and rescue operations with UAVs." In 2010 *International Conference on Emerging Security Technologies* (pp. 142-147). IEEE.

[200] Chandrasekhar, V., Seah, W. K., Choo, Y. S., & Ee, H. V. (2006, September). "Localization in underwater sensor networks: survey and challenges." In *Proceedings of the 1st ACM international workshop on Underwater networks* (pp. 33-40).

[201] Dyo, V., Ellwood, S. A., Macdonald, D. W., Markham, A., Mascolo, C., Pásztor, B., ... & Yousef, K. (2010, November). "Evolution and sustainability of a wildlife monitoring sensor network." In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems* (pp. 127-140).

[202] Starek, J. A., Açıkmeşe, B., Nesnas, I. A., & Pavone, M. (2016). "Spacecraft autonomy challenges for next-generation space missions." In *Advances in Control System Technology for Aerospace Applications* (pp. 1-48). Springer, Berlin, Heidelberg.

[203] Abdelzaher, T., Ayanian, N., Basar, T., Diggavi, S., Diesner, J., Ganesan, D., ... & Veeravalli, V. (2018, July). "Will distributed computing revolutionize peace? the emergence of battlefield IoT." In *2018 IEEE 38th International Conference on Distributed Computing Systems* (ICDCS) (pp. 1129-1138). IEEE.

[204] White, G., Pierson, S., Rivera, B., Touma, M., Sullivan, P., & Braines, D. (2019, May). "Dais-ita scenario." *In Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications* (Vol. 11006, p. 110061F). International Society for Optics and Photonics.

[205] Xiao, J., Zhou, Z., Yi, Y., & Ni, L. M. (2016). "A survey on wireless indoor localization from the device perspective." *ACM Computing Surveys* (CSUR), 49(2), 1-31.

[206] Renner, B. C., Heitmann, J., & Steinmetz, F. (2020). "ahoi: Inexpensive, Low-power Communication and Localization for Underwater Sensor Networks and uAUVs." *ACM Transactions on Sensor Networks* (TOSN), 16(2), 1-46.

[207] Shepard, D. P., Humphreys, T. E., & Fansler, A. A. (2012). "Evaluation of the vulnerability of phasor measurement units to GPS spoofing attacks." *International Journal of Critical Infrastructure Protection*, 5(3-4), 146-153.

[208] Yang, Z., Wu, C., & Liu, Y. (2012, August). "Locating in fingerprint space: wireless indoor localization with little human intervention." In *Proceedings of the 18th annual international conference on Mobile computing and networking* (pp. 269-280).

[209] Gray, A. J., Jenkins, D., Andrews, M. H., Taaffe, D. R., & Glover, M. L. (2010). "Validity and reliability of GPS for measuring distance travelled in field-based team sports." *Journal of sports sciences*, 28(12), 1319-1325.

[210] Maghdid, H. S., Lami, I. A., Ghafoor, K. Z., & Lloret, J. (2016). "Seamless outdoors-indoors localization solutions on smartphones: Implementation and challenges." *ACM Computing Surveys* (CSUR), 48(4), 1-34.

[211] Tariq, Z. B., Cheema, D. M., Kamran, M. Z., & Naqvi, I. H. (2017). "Non-GPS positioning systems: A survey." *ACM Computing Surveys* (CSUR), 50(4), 1-34.

[212] Li, Y., Zahran, S., Zhuang, Y., Gao, Z., Luo, Y., He, Z., ... & El-Sheimy, N. (2019). "IMU/magnetometer/barometer/mass-flow sensor integrated indoor quadrotor UAV localization with robust velocity updates." *Remote Sensing*, 11(7), 838.

[213] A. D. Singh, S. S. Sandha, L. Garcia, and M. Srivastava. "RadHAR: Human Activity Recognition from Point Clouds Generated through a Millimeter-wave Radar." In *Proceedings of the 3rd ACM Workshop on Millimeter-wave Networks and Sensing Systems* (mmNets'19). Association for Computing Machinery, New York, NY, USA, 51–56. 2019.

[214] S. S. Saha, S. Rahman, Z. R. R. Haque, T. Hossain, S. Inoue, and M. A. Rahman Ahad. "Position Independent Activity Recognition using Shallow Neural Architecture and Empirical Modeling." In *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers* (UbiComp/ISWC '19 Adjunct). Association for Computing Machinery, New York, NY, USA, 808–813. 2019.