ON PERFORMANCE PORTABILITY VIA RUNTIME ADAPTATION FOR VO/VSLAMS

A thesis submitted to the University of Manchester for the degree of Doctor of Philosophy in the Faculty of Science and Engineering

2021

Abdullah Khalufa

School of Computer Science
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>10</td>
</tr>
<tr>
<td>Declaration</td>
<td>11</td>
</tr>
<tr>
<td>Copyright</td>
<td>12</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>13</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>14</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>15</td>
</tr>
<tr>
<td>1.1.1 Research Questions</td>
<td>17</td>
</tr>
<tr>
<td>1.2 Thesis Contribution and Structure</td>
<td>18</td>
</tr>
<tr>
<td>1.2.1 Contributions</td>
<td>18</td>
</tr>
<tr>
<td>1.2.2 Thesis Structure</td>
<td>19</td>
</tr>
<tr>
<td>1.3 Publications</td>
<td>20</td>
</tr>
<tr>
<td><strong>2 Background and Related Work</strong></td>
<td>21</td>
</tr>
<tr>
<td>2.1 VSLAM Background</td>
<td>21</td>
</tr>
<tr>
<td>2.1.1 Introduction to VSLAM</td>
<td>21</td>
</tr>
<tr>
<td>2.1.2 VSLAM Formulations</td>
<td>22</td>
</tr>
<tr>
<td>2.1.3 VSLAM Sensors</td>
<td>24</td>
</tr>
<tr>
<td>2.1.4 Benchmarking Datasets</td>
<td>25</td>
</tr>
<tr>
<td>2.1.5 Benchmarking Frameworks</td>
<td>26</td>
</tr>
<tr>
<td>2.2 VSLAM Performance Metrics</td>
<td>26</td>
</tr>
<tr>
<td>2.2.1 VSLAM Accuracy Metrics</td>
<td>27</td>
</tr>
<tr>
<td>2.2.2 Frame Rate</td>
<td>31</td>
</tr>
<tr>
<td>2.2.3 Energy Efficiency</td>
<td>32</td>
</tr>
<tr>
<td>2.2.4 Memory Usage</td>
<td>35</td>
</tr>
</tbody>
</table>
2.3 Related Work .................................................. 35
  2.3.1 Design Space Exploration ............................... 35
  2.3.2 Keyframe-based Methods ............................... 36
  2.3.3 Information Theoretic-based Approaches .............. 37
  2.3.4 Thread-based Mapping and Partitioning ............... 38
  2.3.5 Approximation and Hyperparameter Tuning .......... 38
  2.4 Summary .................................................. 44

3 Preliminaries .................................................. 45
  3.1 The Case for VSLAM Runtime Adaptation ................. 45
    3.1.1 The Synthetic Dataset ............................... 46
    3.1.2 VSLAM Behaviour on the Dataset ................... 48
  3.2 The Basic Adaptation Model ................................ 49
    3.2.1 Translational Motion-based Metric .................. 53
  3.3 General Experimental Setup ................................ 53
  3.4 Parameter-metric Relationships .......................... 55
    3.4.1 Frame Skipping .................................... 56
    3.4.2 Dynamic Voltage and Frequency Scaling (DVFS) ..... 58
    3.4.3 Parameter Combination Behaviour .................... 60
  3.5 Performance on Environments with Realistic Motion .... 62
  3.6 Summary .................................................. 66

4 SLAM-Dunk – The Runtime Framework ...................... 67
  4.1 SLAM-Dunk Pipeline ....................................... 67
  4.2 Estimating Tracking Difficulty ........................... 69
    4.2.1 Computing Motion Metrics .......................... 71
    4.2.2 Computing The Pyramid-based EMD .................. 71
  4.3 Adaptation Policy – Normalising Measured Metrics .... 73
  4.4 Metrics Combination ...................................... 75
  4.5 Parameter Tuning and Configurations .................... 76
    4.5.1 Dynamic Voltage and Frequency Scaling (DVFS) .... 77
    4.5.2 Redundant Frame Skipping ........................... 78
  4.6 Summary .................................................. 79

5 Portable Performance on Sparse VSLAM .................... 80
  5.1 Experimental Setup ....................................... 80
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>VSLAM Metrics Evaluation and Portability</td>
<td>81</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Characterisation Accuracy of the Metrics</td>
<td>82</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Performance on Different Datasets</td>
<td>86</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Performance on the Two Platforms</td>
<td>87</td>
</tr>
<tr>
<td>5.3</td>
<td>Tuning Application-Specific Parameters</td>
<td>88</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Performance on RGB-D TUM</td>
<td>89</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Performance on the Stereo KITTI</td>
<td>92</td>
</tr>
<tr>
<td>5.4</td>
<td>Summary</td>
<td>94</td>
</tr>
<tr>
<td>6</td>
<td>Exploring performance on dense VSLAM</td>
<td>95</td>
</tr>
<tr>
<td>6.1</td>
<td>Experimental Setup</td>
<td>95</td>
</tr>
<tr>
<td>6.2</td>
<td>Exploring Parameters Adaptation Range</td>
<td>96</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Portable Configuration for Adaptations Range</td>
<td>100</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Map Accuracy and Quality</td>
<td>101</td>
</tr>
<tr>
<td>6.3</td>
<td>Summary</td>
<td>104</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion and Outlooks</td>
<td>105</td>
</tr>
<tr>
<td>7.1</td>
<td>Summary</td>
<td>105</td>
</tr>
<tr>
<td>7.2</td>
<td>Review of Contributions</td>
<td>108</td>
</tr>
<tr>
<td>7.3</td>
<td>Limitations and Future Work</td>
<td>109</td>
</tr>
</tbody>
</table>

Bibliography

A Supplementary Materials

Word Count: 25698
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>A taxonomy based on [1] showing different categories of VSLAM formulations. (*) These are the algorithms used for the evaluation.</td>
<td>25</td>
</tr>
<tr>
<td>3.1</td>
<td>The synthetic dataset properties.</td>
<td>47</td>
</tr>
<tr>
<td>3.2</td>
<td>EuRoC MAV and ICL-NUIM properties.</td>
<td>53</td>
</tr>
<tr>
<td>5.1</td>
<td>Results summary on RGB-D TUM.</td>
<td>91</td>
</tr>
<tr>
<td>5.2</td>
<td>A summary of the results from the large-scale KITTI dataset (sequence 00 to 10) shown as violin plots in Figure A.4. The results show the improvements made when adapting the MF value (MF510-2000) relative to the base case where MF is fixed at 2000 ((MF2000). (*) sequences ended prematurely due main memory limit on the Jetson.</td>
<td>93</td>
</tr>
<tr>
<td>6.1</td>
<td>The performance of EF base along with a real-time portable range configuration using the framework.</td>
<td>100</td>
</tr>
<tr>
<td>6.2</td>
<td>The impact of skipping redundant frames on the reconstruction error with respect to the ground truth map.</td>
<td>102</td>
</tr>
<tr>
<td>6.3</td>
<td>The impact of skipping redundant frames which manifests itself as a reduction (%) in the constructed map surface area relative to that of the EF base maps.</td>
<td>103</td>
</tr>
<tr>
<td>A.1</td>
<td>Results from [2] on a TX2 Nvidia Jetson. These results are used for comparison in Section 5.3.1.</td>
<td>127</td>
</tr>
</tbody>
</table>
A.2 Result on RGB-D TUM: the summary of mean values over ten runs shown as violins plot in Figure A.3, after adapting the maximum feature allowed for ORBSLAM2 between the range of 510 to 1000 (MF510-1000) relative to the default fixed value (MF1000). The combination of this parameter along both DVFS and frame skipping is seen to have extra improvements. (*) not included in the geometric mean computation as tracking terminated prematurely on all cases.
List of Figures

2.1 Sample scenes from different datasets. ............................ 25
2.2 The estimated trajectory (in blue) alignment and scaling to the
ground truth (black dashed lines). ................................. 27
2.3 A Dense 3D map reconstruction (mapping) resulting from running
Elastic Fusion on sequence LR0 using an RGB-D camera (a), and
the resulting sparse 3D reconstruction of the MH02 using DSO with
a monocular gray-scale camera (b). ............................... 29

3.1 A sample frame texture from the synthetic dataset (a). (b) shows
the same frame but with key-points tracked using the direct VS-
SLAM formulation, DSO, While (c) shows key-points tracked with
the indirect ORBSLAM2. ........................................... 46
3.2 The camera trajectory in meters (top subplot) aligned with camera
travelled translational distance (bottom subplot). .................... 47
3.3 Runtime power consumption and computation intensity of the
main threads of ORBSLAM2 (a) and DSO (b), running a syn-
thetic dataset with varying camera speed. ORBSLAM2 tracker
and mapper threads are insensitive to the variations. DSO mapper
is less sensitive to the variations compared to its tracker, but
shows a larger impact on power consumption. ......................... 48
3.4 The scale corrections required to match the ground truth trajec-
tory scale of 1.0. These statistics were collected over ten runs
on two different monocular algorithms using sequences from two
datasets and running on two platforms. ............................ 50
3.5 When the change D (green line) is at its historic peak (blue line),
the parameter $X_p$ (red line) is at its maximum level. .............. 51
3.6 The effects of increasing the maximum batch size of skipped frames
on the synthetic sequence. ......................................... 56
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.7</td>
<td>The behaviour using three different methods for adapting skipping frames.</td>
<td>58</td>
</tr>
<tr>
<td>3.8</td>
<td>The behaviour using different methods to adapting DVFS.</td>
<td>59</td>
</tr>
<tr>
<td>3.9</td>
<td>The impact of adapting both DVFS and frame skipping.</td>
<td>60</td>
</tr>
<tr>
<td>3.10</td>
<td>Runtime power consumption and computation intensity of ORB-SLAM2 (a) and DSO (b) main threads, running a synthetic dataset with varying camera speed while adapting both DVFS and frame skipping using the motion metric, $T'$.</td>
<td>61</td>
</tr>
<tr>
<td>3.11</td>
<td>Local pose outliers in one of the resulting base DSO trajectory; due to, possibly, the dark scene on sequence MH04.</td>
<td>62</td>
</tr>
<tr>
<td>3.12</td>
<td>Violin plots of benchmarking the framework on the i7 and using ICL-NUIM sequences.</td>
<td>64</td>
</tr>
<tr>
<td>3.13</td>
<td>Violin plots of benchmarking the framework on the i7 and using EuRoC MAV Machine Hall sequences.</td>
<td>65</td>
</tr>
<tr>
<td>4.1</td>
<td>An abstracted overview of the SLAM-Dunk framework which relies on the rate of change in the scene, quantified with a pyramid-based Earth Mover’s Distance (EMD), and the estimated sensor motions to adapt actuators on both VSLAM and the hosting platform.</td>
<td>68</td>
</tr>
<tr>
<td>4.2</td>
<td>A highlight of where the metric $P'$ is unable to capture the difference in rotation ($R'$) due to being on a different scale with relative to the translation difference ($T'$).</td>
<td>70</td>
</tr>
<tr>
<td>5.1</td>
<td>Violin plots of benchmarking the framework on the i7 and using EuRoC MAV Machine Hall sequences.</td>
<td>82</td>
</tr>
<tr>
<td>5.2</td>
<td>Violin plots of benchmarking the framework on the i7 and using ICL-NUIM living room sequences.</td>
<td>83</td>
</tr>
<tr>
<td>5.3</td>
<td>Resulting trajectories of MH04 from rotation-based adaptations which are applied to DSO running on the i7. Two critical regions are marked with transparent Red.</td>
<td>85</td>
</tr>
<tr>
<td>5.4</td>
<td>Per-frame profile of (DSO-i7-MH04) showing the behaviour of the adapted metrics along with their combination ($\sigma$). The highlighted regions indicate where the individual metrics adapt to unseen change in the environment which is prominent at the start of the sequence. The two parameters (DVFS) and (frame-skipping) are based on the combination ($\sigma$)</td>
<td>86</td>
</tr>
</tbody>
</table>
5.5 Side by side views of trajectories resulting from the Base and $\sigma$-based adaptations. ........................................ 88
5.6 Violin plots of benchmarking the framework on the i7 and using EuRoC MAV Machine Hall sequences. .......................... 89
5.7 Violin plots of benchmarking the framework on the Jetson and using ICL-NUIM living room sequences. ......................... 90
5.8 The incremental memory usage of ORBSLAM2, running on the sequence Fr3-long from TUM RGB-D, as a result the growing factor graph. The slow rise in memory using when adapting all three parameters (combined) compared to the base case (MF1000) is mainly due to skipping frames which filters redundant poses and subsequently reducing the factor graph size. ......................... 92

6.1 Pareto fronts of exploring EF .................................... 98
6.2 The impact of skipping redundant frames on the map quality and its reconstruction error using ICL-NUIM living room sequences with respect to the Base EF. The colour on the map represents the reconstruction error in meters. .................................................. 101

A.1 Sample sequence trajectories from RGB-D TUM where each subplot row represent the ten trajectories corresponding to the RMSE violin plots in Figure A.3 .................................................. 125
A.2 Sample trajectories from the Stereo KITTI ........................................ 126
A.3 Result on RGB-D TUM: Violin plots over ten runs when adapting the maximum feature allowed for ORBSLAM2 between the range of 510 to 1000 (MF510-1000) relative to the default fixed value (MF1000) and the combination of this parameter along both DVFS and frame skipping (Combined). (*) Tracking terminated prematurely on all cases. .................................................. 129
A.4 Violin plots of the ten runs on KITTI dataset. (*) sequences ended prematurely due main memory limit on the Jetson. ............... 130
Abstract

ON PERFORMANCE PORTABILITY VIA RUNTIME ADAPTATION FOR VO/VSLAMS
Abdullah Khalufa
A thesis submitted to the University of Manchester for the degree of Doctor of Philosophy, 2021

The development and use of Visual Simultaneous Localisation and Mapping (VSLAM) in many diverse navigation and vision applications has resulted in a wide variety of algorithmic implementations. Deploying these VSLAM implementations on energy-constrained platforms dictates balancing the target application performance objectives through the tuning of hyperparameters at runtime to ensure prolonged and robust operation. Balancing these performance objectives is difficult to achieve in a portable fashion due to the diverse settings in which different VSLAM implementations operate, leading to limited and over-fitted application-specific solutions. Further, balancing the performance objectives at runtime requires solutions that are lightweight and easy to maintain on existing and emerging VSLAM implementations.

This thesis explores the idea of performance portability at a VSLAM macro-benchmark level where improvements in performance are achieved via runtime adaptation. The research focuses on three aspects: 1. The use of portable metrics for characterising motion and scene changes exploited to obtain effective guidance for runtime adaptations; 2. A top-down approach to enable coping with different VSLAM deployment environments and settings to show that this can be achieved in an efficient and portable manner; 3. The capability of the proposed framework to improve runtime performance, explored and evaluated in diverse and challenging settings where the framework is shown to improve performance objectives with minimal impact on the overall accuracy and robustness of well-established VSLAM implementations. Further, portable performance is shown to be fully achievable using the framework for formulations with similar levels of computational intensity, and is generally achievable to a useful extent.
Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.
Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see http://www.manchester.ac.uk/library/aboutus/regulations) and in The University’s policy on presentation of Theses.
Words are unable to describe my gratitude to my supervisor Graham Riley for his unlimited support, advice and help throughout the ups and downs of my PhD journey and before. The thanks go both ways to you, Graham, and to my co-supervisor Mikel Lujañ for enlightening my knowledge with the fruitful and engaging discussions which shaped this thesis. I also want to thank my friends at the Advanced Processor Technologies group for their support and companionship.

Finally, I am grateful to my parents and wife for their patience, support and being there for me every step of the way.
Chapter 1

Introduction

Visual Simultaneous Localisation and Mapping (VSLAM) is employed at the heart of many emerging and cutting-edge robotics and vision application technologies. VSLAM mainly relies on the rich visual appearance of a scene captured with a moving camera sensor, mounted on a platform, in order to accurately track the platform’s ego-motion with respect to an unknown environment (tracking), while simultaneously reconstructing a real-world map from the observed scenery (mapping). Robotics-based examples include its use for navigation on Mars rovers [3], Unmanned Aerial Vehicles (UAV) [4], self-driving cars [5], four-legged robots [6], and home vacuum cleaners [7]. Vision-based applications include 3D spatial mapping and virtual, augmented, or mixed realities [8, 9, 10].

This use of VSLAM in a wide range of robotics and vision applications has resulted in a variety of algorithmic formulations or implementations based on how the scene is interpreted and utilised in order to aid tracking and mapping. A VSLAM formulation, for example, can be characterised as direct if it uses the intensity of the pixels in the captured camera frame to perform tracking and mapping, indirect if geometrical features, e.g. points, lines or edges, are extracted and utilised [1], or a hybrid of the two (often referred to as semi-direct). Each of these VSLAM categories can be sub-classified based on the amount of information utilised from the camera frame, resulting in dense, semi-dense or sparse formulations. Well-established VSLAM algorithms, regardless of their formulation class, usually support a wide range of camera sensors types. The choice of sensor is typically subject to the deployment environments or the used platform.
1.1 Motivation

The accuracy of VSLAM algorithms is tested and evaluated on representative datasets of a particular deployment environment. These datasets usually contain sequences of the visual input captured with camera sensors, in the form of a sequence of frames, and the ground truth camera trajectory and, in particular environments, the ground truth 3d model map. For example, the EuRoC MAV dataset is widely used for evaluating VSLAM on micro aerial vehicles [11], while, KITTI [12] is used to assess VSLAM for autonomous urban driving. These datasets are used for benchmarking the accuracy of existing and emerging VSLAM algorithm. General benchmarking frameworks such as SLAMBench can be used to facilitate this evaluation [13, 14, 15].

The focus in VSLAM evaluation has been mainly on the accuracy of the camera trajectory or map. For example, in the context of autonomous driving, the vehicle must be aware of its accurate location to avoid collision with obstacles or pedestrians. For augmented reality, the location of the camera (mounted on a headset or smartphone) and the geometry of the observed scene is required to accurately render and interact with augmented 3D objects placed in the real-world. However, as these VSLAM applications are deployed to more computationally constrained platforms, other performance objectives besides the accuracy become critical. These objectives are usually quantified with many metrics including, but not limited to, power efficiency, real-time computation, and robustness to failure, especially in mission-critical scenarios (e.g. the case in autonomous driving) [13, 15, 16]. Key to the success of these applications is the ability of a VSLAM to maintain accuracy and quality of service (QoS) during its operation while being efficient in terms of computation and power use.

VSLAM formulations have diverse parameter spaces supporting general performance tuning. Meeting the required performance requirements can be achieved by exploring this space at design time, usually seeking an acceptable trade-off between several objectives. For example, accuracy may be traded for faster runtime performance [17, 18]. During VSLAM runtime, however, both the sensor motion and the observed scene may be dynamic and in most cases are not known in advance. This variability in motion and scene implies varying tracking difficulty for the formulation. Using a fixed set of pre-tuned parameters throughout the
tracking would likely result in sub-optimal overall performance. Alternatively, adapting these parameters at runtime requires accurate characterisation of the tracking difficulty encountered in the scene. Maintaining accurate tracking under challenging circumstances can be computationally expensive, leading to high power use, for example. Failure to maintain accurate tracking degrades accuracy and robustness and can lead to tracking loss.

One of the critical aspects of robust VSLAM performance, which is a new trend in VSLAM research, is the ability to adapt to such difficulties through dynamic self-tuning [19]. Yet, for a self-tuning adaptation solution to be generally useful, it should ideally be portable to the wide range of VSLAM applications and settings. Given this diversity in VSLAM applications, achieving a fully portable solution is a major challenge since the portability spans multiple levels of the overall problem. For example, at the highest level, the solution needs to be portable across different VSLAM formulations. Second, it needs to operate in different and challenging environments with different camera sensors. Third, the framework needs to be portable across different platforms and still maintain the required Quality of Service (QoS). Finally, the portability further expands to the framework characterisation metrics used to guide the adaptations, and the available tuning parameters and the range over which they may be adapted.

Another crucial aspect of this problem is the efficiency of the portable adaptation solution. This is mainly due to the deployment of VSLAM algorithms on computationally- and power-constrained platform. Additionally, VSLAM algorithms are usually used in conjunction with other vision applications, such as algorithms that are used for object detection, scene understanding or semantic labelling [20, 21]. This dictates that the portable adaptation solution must be lightweight to minimise the impact on VSLAM QoS, power consumption and ultimately the accuracy of tracking and mapping. The efficiency of the solution also extends to its design. For example, a good solution should have a minimal number of tunable thresholds and, so-called, ‘magic’ numbers to enable maintainability and portability. This is difficult to achieve under diverse and challenging settings.

In this thesis, we present a framework to explore the aforementioned challenges and investigate to what extent portable performance is feasible via runtime adaptation. Given the large scope of the problem, we focus the evaluation of our adaptation framework on well-established and open-source VSLAM systems
chosen from different classes where each system is treated as a macro-benchmark. We then diversify our evaluation settings widely while aiming to evaluate challenging cases to stress the framework. The portability is explored and tackled at the different levels of the problem mentioned above, using two high-level and generally available parameters to explore balancing the performance objectives at runtime. The framework demonstrates performance portability to a greater extent compared to similar recent approaches, including [22, 23, 2], given its ability to deal with the previous challenges.

The framework interface relies on information obtained from the user and the VSLAM system to operate. The user-specified information includes the adaptation range for the supported tuning parameters and how the values within this range correlate with VSLAM accuracy. On the other hand, the input obtained from the VSLAM system contains the pose trajectory and the observed scene, which are passed to the framework as the system progresses through the environment. The framework then uses this information and automatically adjust the parameters to balance or improve the performance objectives of the VSLAM system.

1.1.1 Research Questions

As described above, the main challenge of improving the runtime performance of VSLAM is the portability and efficiency of the approach. The main objective of this thesis is to improve the runtime power consumption of different VSLAM algorithms with minimal impact on their accuracy and robustness when compared with their default (base) case where no adaptation is performed. This is achieved in a portable and efficient fashion under a real-time constraint. This objective extends to improving the frame rate where the base VSLAM algorithm is not able to maintain a target QoS frame rate. To achieve this objective, the framework presented in this thesis can adapt portable or algorithm-specific tuning parameters.

The main research questions (RQs) that this thesis aims to answer are:

- To what extent is it possible to achieve portable performance on different algorithms, sensors, operation modes, datasets and platforms using the same portable runtime adaptation framework?

- How effective is the use of general tuning parameters in terms of improving
CHAPTER 1. INTRODUCTION

VSLAM performance, and to what extent can portable performance be achieved using only such parameters, given their range of adaptation values? Further, can these general parameters be used beside application-specific tuning parameters?

- How can the tracking difficulty be estimated and characterised reliably and efficiently while relying purely on the readily available motion and scene metrics from VSLAM systems? And to what extent do these metrics enable portable performance across the range of VSLAM formulations and their settings and the scenario environments in which they are deployed?

1.2 Thesis Contribution and Structure

1.2.1 Contributions

The thesis presents several contributions as a result of investigating runtime performance and its portability across diverse VSLAM settings. These contributions are as follows:

- **SLAM-Dunk**: A portable and modular lightweight runtime adaptation framework for balancing the performance objectives across different VSLAMs, environments and platform settings. The framework requires minimal changes to the targeted VSLAM implementations, yet demonstrates performance improvements on multiple well-established VSLAM implementations.

- The development of a novel adaptation policy based on normalising characterisation metrics employed to tackle two main challenges: first, to adapt to unknown variations in environments, regardless of their scale or the sensor used, and, second, to minimise the potentially large number of tunable hyperparameters involved, which pose portability and maintainability issues. The policy is lightweight and easy to maintain with only a single hyperparameter which is required mainly on VSLAMs with non-deterministic behaviour.

- The introduction of a novel efficient scene similarity metric. The metric quantifies the observed scene changes by maintaining the perceptual appearance of the scene and meets the real-time constraint of VSLAM. The
metric is based on comparing coarser Gray-scale Gaussian pyramids using the Earth Mover’s Distance (EMD) [24].

- A performance analysis of the framework when adapting two effective portable parameters, frame skipping and Dynamic Voltage and Frequency Scaling (DVFS). Both are first evaluated in a simulated environment, with the main goals of establishing their correlation with associated metrics, and then in realistic environments to explore the impact on the accuracy and robustness of various VSLAMs.

- A system-wide benchmarking to evaluate the performance portability of different motion and scene characterisation metrics, along with their use in combination, in a challenging setup. The evaluation spans two different sparse VSLAMs, scenes from two different environments and execution on two platforms using the portable parameters.

- A design space exploration of parameter ranges with the aim to improve multiple performance aspects of a dense, compute-intense VSLAM on a low-end platform, resulting in a qualitative and quantitative analysis of the impact on the accuracy of the trajectory and the quality of the reconstructed map.

1.2.2 Thesis Structure

The structure of the thesis is as follows. Chapter 2 presents the relevant background and related work. Here, Visual SLAM is briefly introduced with the focus on the well-established formulations and camera sensors which form the basis of the evaluation in this thesis. VSLAM performance objectives and metrics used for the evaluation are then described. This is followed by a critical review of the related work regarding the design and runtime improvement of VSLAM performance requirements, including benchmarking datasets and frameworks. Chapter 3 establishes the basic concepts and design choices proposed to improve the runtime performance of VSLAM and highlights the challenges faced when seeking a portable and lightweight approach. First, the argument for the need for runtime adaptation is presented; second, a basic adaptation model is motivated and described, then used to evaluate two portable parameters, frame skipping and Dynamic Voltage and Frequency Scaling (DVFS), and establish their
correlations to the selected characterisation metric in a simulated environment. Finally, the basic adaptation model is evaluated in realistic endowments, where certain limitations are exposed and discussed. Chapter 4 describes SLAM-Dunk, the presented runtime framework, which adds extensions to the basic model described in Chapter 3. Extensions are made to both the adaptation policy and the characterisation metrics. In Chapter 5, the aspect of portability is explored, under challenging settings with two prominent sparse VSALM algorithms, in terms of characterisation metrics, tuning parameters and SLAM-Dunk settings, using portable and application-specific tuning parameters. Chapter 6, explores the use of the two portable parameters to improve the performance of a dense form of VS-LAM on a low-end robotics platform. Conclusions, future work and limitations of the research are presented in Chapter 7.

1.3 Publications

The research in this thesis has resulted in the following publications:

Published:


In progress:

- Abdullah Khalufa, Graham Riley, and Mikel Luján. "SLAM-Dunk: A Framework Leveraging Motion and Scene Dynamics to Improve Robustness and Power-Efficiency across VO/VSLAMs".
Chapter 2

Background and Related Work

This chapter presents the relevant background and related work for this thesis. In Section 2.1, we provide a general background by first introducing the basic concepts behind VSLAM. Then we provide details about its different formulations and used sensors. We then discuss the datasets and the benchmarking solutions used to evaluate the performance of VSLAM. Next, we describe the metrics used to evaluate different VSLAM performance objectives in Section 2.2, which also includes a background on the approaches used to improve these objectives and tools which facilitate measurements. Section 2.3 presents the related literature for improving VSLAM performance at design and runtime. First, proposed methods to characterise the tracking difficulty are presented, followed by the general and VSLAM-specific runtime control approaches.

2.1 VSLAM Background

This section provides a brief background on VSLAM and its formulations, performance requirements, and evaluation methods and benchmarks.

2.1.1 Introduction to VSLAM

The main problem that VSLAM aims to solve is to accurately estimate the trajectory of a moving camera mounted on a platform while building a 3D world map of the observed scene or its landmarks by using the camera frames as input. VSLAM algorithms encompass the so-called Visual Odometry (VO) and
the ability to perform global map optimisation. VO is responsible for incrementally (frame-by-frame) estimating both the camera orientation and position while building a local map of the surroundings. As the camera progresses through the environment, small errors accumulate over time causing the estimated camera position to drift from its ground truth trajectory (translational, rotational or scale drift). If the system revisits a previous location (loop closure), it performs a global map optimisation which significantly reduces the accumulated error, keeping both camera track and the global map of the environment consistent.

VSLAM systems consist typically of two main components: a front and a back-end [19]. The front-end is responsible for the selection of scene features or input frames to be used. If the whole scene in the frame is used and all input frames are utilised, this results in a dense form of VSLAM. On the other hand, if only a subset of useful input frames is used, the VSLAM is considered as keyframe-based. The density is then determined by the amount of information utilised in the input frame. The front-end is also responsible for matching and tracking scene features across the observed scene and performing loop closure if a place is revisited. The back-end, however, is responsible for estimating and optimising the trajectory poses with respect to the location of features within the map using the selected data from the front-end. The underlying optimisation process involves the incremental refinement of the location of the previously observed poses, represented mainly as a factor graph, based on the new observations [19]. Alternatively, instead of optimising the factor graph of the trajectory poses, the map can solely be refined by optimising the map deformation graph [25]. The VSLAM field is large and diverse, thus going into details of the design and implementation of each formulation is beyond the scope of this thesis; however, more details can be found in these general tutorials [26, 27, 28, 19] or the respective publications of VSLAM algorithms.

2.1.2 VSLAM Formulations

The use of VSLAM in a wide range of robotics and vision applications has resulted in a variety of algorithmic formulations which can be characterised based two aspects: the first is how the observed scene is interpreted, which can be direct or indirect. The second is concerned with how much of the scene is utilised, that is dense or sparse. We discuss the difference between these formulations in the following subsections.
2.1. VSLAM BACKGROUND

2.1.2.1 Indirect Formulations

The indirect formulations interpret the observed scene, which is represented as raw camera measurements, by first extracting a set of features (also called keypoints or landmarks) using feature descriptor methods such as, FAST [29] or ORB [30]. These features, which can be in the form of points, edges or lines, are then matched across the visual input, and the geometric error between these matched features is minimised to accurately estimate the sensor pose. A key advantage of relying on well-selected features using the invariant descriptors is the robustness to noisy artefacts or brightness changes in the input images [31]. Example sparse indirect VSLAM formulations, which rely on a sparse set of features for matching, include MonoSLAM [32], PTAM [33] and ORBSLAM [34, 35, 36].

Using a sparse set of features results in a sparse map, however at the advantage of a lower computational cost. Approaches that rely on semi-dense feature points, such as the work in Mur-Artal et al. [37] or more recently VITAMIN-E [38], and approaches that use dense optical flow, such as VOLDOR [39], are proposed to build a much denser map. These approaches require more computation compared to sparse methods such as ORBSLAM2 [35] where an extra computation unit may be required, such as a GPU, which is the case for VOLDOR [39]. ORBSLAM2, which is a well-established algorithm supporting a wide range of sensors, is used as a representative system for evaluating our framework on sparse indirect methods. We believe that ORBSLAM2 is sufficient for this purpose since our framework is capable of improving its performance even though it is sparse and keyframe-based (and therefore computationally efficient).

2.1.2.2 Direct Formulations

Direct formulations skip feature extraction and directly use raw pixel intensity for matching across the aligned input frames where the photometric error is then minimised [1]. This enables having more access to a wide range of traceable visual entities for tracking across the frame rather than only using geometric entities (e.g. points or lines) which can be advantageous in scenes that lack structure or are low-textured. Further, maps constructed with the direct method have a rich visual representation as opposed to indirect methods. However, relying on pixel intensity is susceptible to illumination artefacts which cause changes in
the intensities of the pixels potentially leading to a mismatch and subsequently tracking loss [40, 41, 42]. This is often tackled online by the so-called photometric calibrations [31, 43].

There are many examples of Direct VSLAM formulation ranging from sparse to dense. Direct Sparse methods include Direct Sparse Odometry (DSO) [1] and its extensions [44, 45, 46]. LSD-SLAM [47] is an example of semi-dense direct methods. These sparse and semi-dense methods can run in real-time using only CPUs. On the other hand, dense methods, such as Dense Tracking and Mapping (DTAM) [48], KinectFusion [49] and Elastic Fusion (EF) [25] require a high-end GPU to operate in real-time due to the large number of matched points, usually in the range of hundreds of thousands [25]. In this category, we chose two algorithms from the two ends (sparse and dense), DSO and Elastic fusion as representative in the evaluation of our work.

2.1.2.3 Semi-Direct Formulations

The most prominent examples in this category are the Semi-direct Visual Odometry (SVO) [50, 51] and the Point Line (PL) SVO [52]. These methods are a hybrid between direct and indirect formulations. The front-end, where the frames are aligned, is based on direct pixel matching, these pixels are then represented geometrically to be used, indirectly, in the back-end where the optimisation is performed. SVO is very efficient in terms of computation, which makes it attractive for use on low-end processors [51]. Table 2.1 shows a taxonomy of the different VSLAM formulations and their examples based on [1].

2.1.3 VSLAM Sensors

VSLAMs rely on the information extracted from visual frames shot with a camera sensor to perform localisation and mapping. Depending on the target application, the sensor can be monocular (a single camera), stereo, RGB-D (produces coloured and depth frames), omnidirectional and more recently event cameras [54]. Camera sensors can also be augmented with additional modalities such as Inertial Measurements Units (IMU) for increased robustness [55, 56]. Well-established VSLAM algorithms, such as ORBSLAM2 and DSO, support multiple forms of sensors. The use of the pure monocular sensor is attractive due to the low budget
### 2.1. VSLAM BACKGROUND

<table>
<thead>
<tr>
<th>Sparse</th>
<th>Semi-direct</th>
<th>Direct</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonoSLAM [32]</td>
<td>SVO [50, 51]</td>
<td>DSO* [1, 44, 45]</td>
</tr>
<tr>
<td>PTAM [33]</td>
<td>PL-SVO [52]</td>
<td>DSM [46]</td>
</tr>
<tr>
<td>ORBLSLAM* [34, 35, 36]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Semi-dense</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mur-Artal et. al. [37]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VITAMIN-E [38]</td>
<td>LSD-SLAM [53, 47]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dense</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOLDOR [39]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: A taxonomy based on [1] showing different categories of VSLAM formulations. (*) These are the algorithms used for the evaluation.

and weight, and ease of calibration compared to other forms of sensing [57]. However, in terms of accuracy and robustness, relying purely on a monocular sensor (intensity-based) is considered the most challenging due to scale ambiguity and drift [57, 53].

### 2.1.4 Benchmarking Datasets

![Sample scenes from different datasets.](image)

VSLAM systems evolve rapidly leading to new datasets being proposed to evaluate their accuracy within different application environments and challenging settings. Traditionally, the accuracy of tracking and mapping of VSLAM algorithms are evaluated using datasets which contains input visual frames (images) along with a ground truth trajectory or 3D map model to quantify the drift.
The scene in these different datasets can be synthetic or realistic, structured or textured, static or with dynamic objects moving within the scene. Further, the scene can be captured in environments, indoor or outdoor, with variations in the illumination and brightness.

For example, EuRoC MAV [11] is widely used to evaluate VSLAM on a drone-based platform in indoor navigation. The drone flies and captures scenes in industrial settings and room-based environments using a stereo camera. While KITTI [12] provides scenes for large scale autonomous driving in urban and highway roads. For hand-held-based indoor RGB-D VSLAM, TUM RGB-D dataset [58] and ICL-NUIM [59] are the most prominent. The latter is a synthetic dataset with a 3D ground truth model. Figure 2.1 shows sample scenes from these datasets. Recent datasets have also emerged for a specific purpose. For example, for drone racing [60], or to provide diverse environments, weather conditions and motion patterns TartanAir [61].

2.1.5 Benchmarking Frameworks

Evaluating VSLAM algorithms involves multiple steps starting with running the desired macro-benchmark in order to collect statistic and visualising the resulting trajectory or map for analysis. To facilitate the VSLAM algorithm evaluation pipeline, frameworks such as SLAMbench [13, 14, 15] and GSLAM [62] are proposed to help automate the process. SLAMBench has evolved to support a wide range of algorithms and datasets compared to GSLAM. Its development occurred in parallel to the research in the thesis; however, SLAMBench is used to facilitate the evaluation in Chapter 6. Most of the evaluation in this thesis is based on custom-built python scripts to facilitate the experimentation. However, we rely on EVO [63] for plotting trajectories and quantifying the drift.

2.2 VSLAM Performance Metrics

This section describes the different metrics used for quantifying the relevant performance objectives, mainly the trajectory and robustness, map accuracy and quality, frame rate, and power consumption.
2.2. VSLAM PERFORMANCE METRICS

2.2.1 VSLAM Accuracy Metrics

The accuracy is the most widely used metric for benchmarking the different VSLAM formulations, as opposed to, for example, runtime performance and power consumption. There are two aspects to consider when evaluating the accuracy of VSLAMs. The first is the accuracy of the estimated sensor trajectory, and the second aspect is concerned with the accuracy of the constructed scene map. The accuracy is usually measured by evaluating the estimated trajectory and reconstructed map against an existing ground truth trajectory and a 3D model map, respectively. Although the accuracy of the trajectory may impact the map in some way, usually accurate mapping does not imply accurate tracking [58, 59]. In the following, we discuss both aspects of VSLAM accuracy.

![Figure 2.2: The estimated trajectory (in blue) alignment and scaling to the ground truth (black dashed lines).](image)

2.2.1.1 VSLAM Trajectory Accuracy

The estimated trajectory consists of a sequence of time-stamped camera poses consisting of translational and rotational components and spanning the whole trajectory. These poses can be in the form of a rigid transformation matrix or quaternions [64]. The number of these poses is subject to the type of VSLAM formulation and how fast VSLAM processes the frames. Thus, a time-based association with the ground truth poses is usually employed [59, 65]. The impact on the trajectory manifests itself as a drift from the ground truth. A number of steps may be taken to quantify this drift. For example, The resulting trajectory may not have the same frame of reference as the ground truth. This may require
CHAPTER 2. BACKGROUND AND RELATED WORK

aligning the estimated trajectory through similarity or rigid body transformation with the ground truth using, for example, Umeyama alignment [66, 65]. While the use of a pure monocular sensor with VSLAM results in scale drift which needs to be corrected before measuring the accuracy of the estimated trajectory.

Figure 2.2 shows an illustration of such a process where the estimated trajectory shown in (2.2a) is aligned (2.2b) and then scaled (2.2c) to the ground truth. The drift from the ground truth is then calculated using either the Relative Pose Error (RPE) or the Absolute Trajectory Error (ATE). The former measures the local motion error between the poses within a time period, while the latter measures the absolute difference (drift) between all poses, where the mean ATE or Root Mean Square Error (RMSE) are usually reported. Using the RMSE has the advantage of accounting for outliers compared to the mean ATE.

In this thesis, we mainly use the translational absolute trajectory RMSE to measure the drift. This is because the rotational error manifests itself as a translational one as the sensor moves [58]. The absolute trajectory RMSE of the translational components, after alignment and scaling, is defined as in the following:

\[
RMSE(P_{1:n}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \|\text{trans}(P_i)\|^2}
\]  

\(P_i\) is the absolute difference between the estimated and ground truth pose at the associated time-stamp \(i\). An advantage of using the ATE is to enable visualising the drift with respect to the ground truth and to evaluate the global accuracy of VSLAM. Further information about these metrics can be found in [58] and [65].

2.2.1.2 VSLAM Robustness

Robustness is a key metric for evaluating VSLAM algorithms with non-deterministic behaviour, such as [35] and [1]. This metric measures how the accuracy of VSLAM varies from run-to-run under the same settings. This variability can pose robustness implications in mission-critical applications. This non-deterministic behaviour can be attributed to different reasons or their combination. For example, the hard deadline constraint where frames may be dropped at different positions during the tracking to maintain QoS can lead to a varying accuracy over multiple runs. This is mainly due to the time dependency which is affected
by the runtime platform workload. Another possible cause may be due to the multi-threaded implementations of these algorithms, which may involve a form of synchronisation and thread locking. Finally, extreme motion patterns, especially rotational and scene depravity of tracking information can also lead to such behaviour [35, 67, 68].

Considering this metric when applying runtime adaptation is essential to observe if the runtime adaptations made an impact (positive or negative) on how the base VSLAM algorithm performs on a run-to-run basis. In this thesis, we evaluate this metric on VSLAM algorithms that are known to show such behaviour.

### 2.2.1.3 VSLAM Map Quality and Accuracy

![Dense mapping with EF](image1)

![Sparse mapping with DSO](image2)

Figure 2.3: A Dense 3D map reconstruction (mapping) resulting from running Elastic Fusion on sequence LR0 using an RGB-D camera (a), and the resulting sparse 3D reconstruction of the MH02 using DSO with a monocular gray-scale camera (b).

The reconstructed 3D map may have different representations; for example, it can be represented as a point cloud or a mesh. In order to evaluate the reconstructed map, the ground truth 3D model needs to be available for reference. The evaluation involves mainly two aspects: first the accuracy of the reconstructed map and second the quality of the reconstruction in terms of density and the covered surface area [59, 25]. The latter is mainly of interest when using dense mapping VSLAM algorithm such as Elastic Fusion (EF). Other sparse VSLAM algorithms favour lower computation intensity over the generation of dense maps, such as DSO and ORBSLAM2. These algorithms focus mainly on the sensor trajectory accuracy rather than the map, with few exceptions such as Direct Sparse Mapping (DSM) [46]. Figure 2.3 shows examples of dense and sparse 3D map
reconstruction. In the following, we describe the two aspects of evaluating the 3D map reconstruction and provide details about the used metrics and tools.

**Accuracy of 3D Reconstruction:** Measuring the accuracy of the 3D map reconstruction involves multiple steps. First, if the estimated map needs to be aligned with the ground truth 3D model due to the lack of a frame of reference, a coarse-to-fine registration process needs to be performed [59]. Coarse registration produces an initial global alignment with the reference 3D map registration. This global alignment of the reconstruction is then refined and possibly scaled to accurately match the reference 3D map.

Automatic coarse registration involves searching for correspondence between at least three matching pairs of points from the registered and the reference map. Based on these correspondences, a rigid transformation that brings these matching points closer is performed on the whole registered point cloud. The challenge arises from the high computational cost involved in this task which is subject to the number of points in the two clouds and their structural appearance [69]. Alternatively, this task is facilitated manually with the help of point cloud comparison tools and libraries such as Cloud Compare [70] and Open3D [70]. This involves a visual inspection of the two clouds to manually select matching points.

After the reconstructed map is coarsely registered with the ground truth map, additional steps such as sub-sampling and filtering outlier points or scaling may be required. Then, the fine registration is performed by applying an Iterative Closest Point algorithm [71, 72] which further minimises the distance between the two clouds by iteratively finding and aligning the closest matching points through rigid transformations.

Upon the completion of the registration process, the ground truth point cloud is then transformed into a surface mesh (a set of triangle faces spanning the map surface), and the perpendicular distance is measured between each point on the register map and the closest mesh triangle. Statistics about these distances are often used to quantify the accuracy of the 3D reconstruction [59], which we employ later in Chapter 6 for evaluation purposes.

**The Quality of 3D Reconstruction:** In terms of quality, we focus on the total area covered by the reconstruction as a metric. This metric can be used to quantify the tradeoff between faster execution time and the resulting reconstruction area after applying runtime adaptation. The tradeoff manifests itself as a relative reduction in the surface area between the two maps. The
metric can be calculated by first pre-processing the map (which may involve subsampling and outlier rejections) and then generating a surface mesh on both the reconstructed and reference maps, and finally the relative reduction on the total area is calculated.

### 2.2.2 Frame Rate

Frame rate is used to quantify the capability of VSLAM algorithms to process frames in a certain amount of time to maintain a quality of service (QoS). This usually implies a fixed rate or a hard deadline for processing each frame. Frames are dropped if the deadline is not met, which may lead to the degradation of the tracking accuracy. Single-threaded VSLAM algorithms, where tracking and mapping are performed in the same thread, usually struggle to achieve real-time performance on low-end devices. While algorithms that are based on parallel tracking and mapping [33], can achieve real-time performance due to the decoupling of mapping and global optimisation (loop closure) from tracking into separate parallel threads. This enables the tracking component to operate in real-time at a fixed higher rate, while mapping or global optimisation is performed on the separate parallel thread where this constraint is relaxed. If the tracker thread finishes tracking before the deadline, it usually sleeps until the next frame becomes available.

If a VSLAM algorithm fails to meet the real-time deadline constraint for the whole duration of tracking, it can operate in the so-called *linearised* mode [1]. In this mode, the algorithm can take as much time as needed to process a frame, meaning that processing time of some frames may exceed the deadline, while this may not be the case on others. In other words, the processing is done as fast as possible without dropping frames. In this case, the average per-frame time is usually used to quantify to improvements. The per-frame time is measured by dividing the total execution time by the number of processed frames.

Operating at a fixed frame rate, where frames can be dropped, may have an impact on the algorithm accuracy or robustness compared to linearised mode. Thus, it is essential to evaluate the adaptation framework in this challenging mode to observe if the framework has a negative impact on the processing time. This is usually the case if the tuning parameter increases the processing time. In this thesis, we evaluate our framework in these different cases.
2.2.3 Energy Efficiency

VSLAM algorithms are deployed mainly on robotics platforms which are essentially battery-based. Improving power consumption is important for prolonging navigation operations. In the following, we discuss how power can be measured on computing platforms, and discuss the software-based power sampling tools used, followed by the techniques using tunable parameters that are used to minimise power consumption.

2.2.3.1 Power Consumption

Power consumption, in terms of CMOS-based Integrated Circuits (ICs), is mainly denoted by the so-called Dynamic Power ($P_{\text{dynamic}}$) which is essentially the power needed to switch the state of transistors [73], defined as the following:

$$P_{\text{dynamic}} = CV^2Af$$

Where $C$ is the Capacitive load, $V$ is the voltage, $A$ is the Activity factor which determines switches per cycle and $f$ is the clock frequency. Besides $P_{\text{dynamic}}$, Static power is another form of power consumed by ICs. Defined as:

$$P_{\text{static}} = VI_{\text{leak}}$$

it results from current leakage, despite the transistor switching state. The static power is usually tackled by switching off the power supply of inactive processing components, also called power gating [73]. In this thesis, we focus on reducing the dynamic power consumption and discuss the portable and application-specific approaches used to reduce power and computation time.

**Dynamic Voltage and Frequency Scaling (DVFS):** Most computing platforms incorporate a form of Dynamic Voltage and Frequency Scaling. The commercial term on Intel platforms is TurboBoost, and for AMD is Turbo core. Although these commercial terms refer to changing the frequency of a processing core, the voltage must also be changed to achieve the desired frequency; voltage and frequency of processing cores are not independent, see Equation 2.2. Thus, for a given platform, we can capture the DVFS range available with the set of pairs, of voltage and frequency, at which the platform can operate. DVFS can be applied to the different processing units available, for example, CPU, GPU, etc.
2.2. VSLAM PERFORMANCE METRICS

Power consumption can be reduced by decreasing the clock frequency or voltage when there are low computing activities on the processing unit. This process is usually achieved with predefined policies implemented by scaling governors that characterise the workload and adjust the settings accordingly, based on the policy used by the governor. The governors and policies are specific to each scaling driver. For example, the policies of the performance governor on the Linux Kernel’s default scaling driver, acpi-cpufreq, are different from those of the intel_pstate driver [74, 75].

Beside governors, the user is allowed to control the clock frequency by setting minimum and maximum boundaries to a predefined set of frequency steps which are processor-dependent. This enables defining custom and application-specific frequency policies to be used where the default governors do not deliver the desired performance and energy savings. This case is explored in this thesis in the context of VSLAM where the DVFS value is adapted based on VSLAM tracking difficulty instead of using the default governors.

Frame Skipping: Frame skipping/dropping reduces the amount of computation performed by the processing unit in frame-based applications. Desirable outcomes may result in improved frame rate, power consumption or quality of the results. We employ this technique in the context of VSLAM under different operational modes, i.e. fixed and variable frame rate, described in Section 2.2.2. In the former, frame skipping can result in power reduction, while the latter may result in a reduction in both power and time. Further, in a VSLAM based on a pose graph, frame skipping results in fewer poses/nodes in the graph minimising the overall graph computations and memory usage.

Application-specific Parameter Tuning: The tuning of hyperparameters that are specific to a VSLAM can improve the computation time or power consumption. Tuning some of these hard-coded parameters may come at the expense of modifying the base VSLAM code in order to make it highly tunable [17]. However, given that parameter portability is a key aspect in this thesis, we only perform limited exploration on ORBSLAM2 using the maximum amount of features tracked across frames for tuning at runtime. This is to showcase the capabilities of the framework presented in this thesis to tune both application-specific and general parameters in different environments.
CHAPTER 2. BACKGROUND AND RELATED WORK

2.2.3.2 Power sampling tools

In this section, we described the two power sampling tools for the platforms used in this thesis, which are platform dependant:

- **TurboStat** [76], a Linux kernel tool, is a tool that reports general statistics about x86 processors. TurboStat provides software-based power sampling by reading the wattage information from RAPL (Running Average Power Limit) driver MSRs (Model-Specific Registers). RAPL is a model-based interface that provides accurate measurements based on modelling the information from hardware performance counters to estimate the active power consumption [77, 78, 79].

- **INA3221 power monitor**: is a software-based power driver that enables monitoring the power of different system components available on Nvidia Jetson AGX Xavier robotics platform [80]. The power can be monitored over two I2C addresses each with 3-channels, where the power information is accessible through reading `sysfs` nodes. The CPU, GPU and SOC power are accessible through the first address, while the SDRAM and other components of power are accessible through the second address. The reported measurements from this monitor are based on averaging the latest 512 reads of a particular channel with the accuracy of the measurements being within 5% [81].

2.2.3.3 VSLAM Power Consumption

In the context of VSLAM, power is usually measured per frame [13]. This can be achieved by either sampling power throughout the duration of frame processing and then taking the average\(^1\) or employing a method that samples the power when the tracker thread has finished. The former is more appropriate, especially on multi-threaded real-time implementations where the tracker may be waiting for a new frame to arrive while the mapper is running in the background. This is why the choice of the sampling rate is important in order to obtain accurate measurement while not impacting VSLAM performance or power consumption due to the measurement’s overhead. For example, using a sampling rate that is

\(^1\)Taking the average instead of the sum of the sample reduces the effects of outliers in power sampling; software-based power sampling methods are susceptible to issues with outliers.
slightly greater than the frame rate of VSLAM is ideal for capturing the power consumed by the different treads.

When a VSLAM algorithm operates under a real-time constraint, the average per-frame time is considered the same. In this case, it is sufficient to report the relative reduction in per-frame power after applying runtime adaptation. On the other hand, if the algorithm operates in the linearised mode, described in Section 2.2.2, the per-frame time is variable. In such a case energy needs to be measured, which we define as the product of the average per-frame time and power.

2.2.4 Memory Usage

Memory usage is another metric that is used to evaluate the performance of VSLAM [82]. This metric is important to consider when the usage increases over time due to the growth of the factor graph size as the sensor progresses through the environment, and new observation and poses are incorporated. Improvements on this metric are achieved by reducing the size of the graph which is explored in Section 5.3.1.

2.3 Related Work

The framework has been designed to operate at runtime and to achieve performance portability across multiple VSLAM implementations, with the ability to be tuned for a specific VSLAM system. The components of the framework span multiple areas of the literature. This section starts with a literature review of the approaches used to improve VSLAM performance objectives in offline and online contexts. We then focus our scope on the methods used to facilitate VSLAM approximation and hyperparameter tuning.

2.3.1 Design Space Exploration

The deployment of applications on platforms with limited computational and power resources is a challenging task and often involves trading-off different performance objectives with the help of parameter tuning and domain-specific knowledge [83]. In the context of VSLAM systems, performance objectives,
such as accuracy, power consumption and frame rate, can be balanced at design time. This is frequently undertaken by exploring the available parameter space [17, 67, 18, 84], where sophisticated and costly design space exploration techniques are frequently employed. These include, for example, Machine Learning techniques to help prune and navigate the search space.

However, some of the parameters involved ideally need to be varied at runtime to respond to the changing nature of the scene and sensor motion. In this work, we target mainly two effective and portable parameters which are adapted based on the change in the sensor motion and observed scene. (Currently, the ranges over which the parameters are adapted is predefined). Exploring the predefined range of a limited number of parameters to find the desired runtime trade-offs is achievable within a reasonable time as we demonstrate in Chapter 6 where sophisticated design space exploration methods are not required.

As the number of runtime tunable parameters increases and the adaptation is performed during VSLAM operation, this task becomes more difficult to achieve, mainly due to the potential negative impact induced by the parameters on each other (since the problem is a multi-objective optimisation problem). Domain-specific approaches, such as [85], aim to tackle this issue by learning the proper parameter configuration and range online to improve the performance of workloads of parallel computations. In term of VSLAM, the primary objective is the accuracy of tracking and mapping which is quantified post-mortem. This makes it difficult to learn the proper parameter adaptation range online to improve performance. This, however, can be a future line of inquiry to improve the overall performance of VSLAM, if the tracking accuracy can reliably be quantified with the help of, for example, a global positioning system (GPS) in an outdoor environment or traceable beacons in GPS-denied environments.

### 2.3.2 Keyframe-based Methods

These approaches improve VSLAM performance by selecting or filtering frames based on predefined criteria aiming to quantify their usefulness [86]. The criteria are usually specific to the VSLAM implementation; for example, DSO decides on selecting a new keyframe if there is a change in the field of view observed by the sensor or in the presence of occlusion and variation in brightness [1]. The frame selection criteria can also be based on both visual and inertial input, such as the approach used in [87] which relies on optical flow and IMU data to select
keyframes. Complementary to these approaches are methods that filter or skip frames before being fed to a VSLAM system. These methods are general and aim to relieve the algorithm from performing computations on frames that are already known to be of low value. The methods can also be used to improve the performance of non-keyframe based VSLAM. The work presented in [22] and in this thesis, which is published in [88], are examples of such approaches.

The approach proposed in [22] is based on dropping frame when the sensor rotation is below a specific threshold, implying that frames are dropped regardless of how fast is the translational motion. The method is evaluated on top of a stereo visual odometry library called libviso2 [89] using the KITTI [12] driving dataset and running in the linearised mode, described in Section 2.2.2. In this thesis, however, we evaluate our approach on multiple monocular algorithms and in real-time (fixed frame rate) on the EoRoC MAV dataset, which has challenging motion patterns, being a drone-based dataset. Our evaluation has exposed that relying on a single pure metric, whether rotational or translational, is limited and can result in an impact on the accuracy and robustness, especially under challenging settings, as explored in Section 5.2.

2.3.3 Information Theoretic-based Approaches

Information theoretic-based approaches aim to remove dispensable features or frames which hold minimum value for tracking. The work proposed in [90, 91, 92] aims to improve graph-based VSLAM efficiency and long-term operation by reducing or discarding less informative graph poses. While in [93], an information-driven point and keyframe selection method is proposed to improve the performance of a direct RGB-D VSLAM. The approach presented in this thesis, which is a distance-based, skips redundant frames based on a number of normalised distance metrics that are affected mainly by the change in sensor motion, achieving a similar goal but with the added benefit of being portable to a wide range of VSLAM formulations, regardless of the sensor and the optimisation method used. This is the case whether it is a pose graph-based VSLAM or VSLAMs that do not employ a pose graph, such as Elastic Fusion [25]. Elastic Fusion is evaluated in this work in Chapter 6.
2.3.4 Thread-based Mapping and Partitioning

Heterogeneous multiprocessor system-on-chips (MPSoCs) are widely adopted in most of the mobile systems. For example, ARM big.LITTLE technology [94] aims to enable performance and power efficiency with the heterogeneous integration of energy-efficient CPU cores for light workload and powerful CPU cores for a heavy workload. These CPU processes are integrated with graphical processing units (GPUs) or neural processing units (NPUs), among other components on the same chip. To help maximise the efficiency, the applications are tailored for, and mapped on, the most suitable processing unit with the help of general heterogeneous programming languages such as openCL [95] or domain-specific languages (DSL) such as the image and array processing language Halide [96].

VSLAM implementations are diverse, some being single-threaded [32], multi-threaded [33, 1, 35], or heterogeneous (CPU+GPU) [49, 25, 39]. Singh et al. [97], rely on offline profiling to improve the energy efficiency of Kfusion [49] on MP-SoCs. To achieve this, a fixed chosen frequency value is assigned to each CPU core, and each core (big or LITTLE) can be configured to be online or offline depending on the performance requirement. The workload is then partitioned between the CPU cores and/or GPU. In the framework presented in this thesis, portability is our main concern; thus, we mainly adapt DVFS value instead of using a predefined fixed value or performing workload partitioning. The adapted DVFS value, which is based on general metrics that estimate the tracking difficulty, can be applied to any processor that provides the support for DVFS. Adapting the DVFS value enables finer tuning and balancing between performance and power efficiency, with the possibility of maintaining the thread data locality on the target core’s exclusive memory levels.

2.3.5 Approximation and Hyperparameter Tuning

Methods to approximate VSLAM are proposed to improve its performance. For example, the performance can be improved using approximation methods such as reduced precision computations or loop perforation (which involves selective loop iteration skipping) as in [98, 99] which targets certain kernels in RGB-D-based VSLAM. In the former, a mix of 16-bit and 32-bit floating point precision is used to demonstrate that computation cost can be reduced with minimal impact on accuracy. Similarly, the latter utilises both reduced precision computations and
loop perforation in the selected VSLAM to improve computation time and power consumption with minimal impact on the accuracy.

Alternatively, online hyperparameter tuning can be used to achieve similar improvements for different performance objectives. These parameters can be general and portable, such as the number of frames skipped or DVFS, as demonstrated in this thesis, or VSLAM implementation-specific such as the work presented in [23]. Contrary to our approach, where both DVFS and frame-skipping is combined, Pei et al. [23] employ hierarchical tuning of parameters, where only one parameter is tuned at a time. The use of a parameter hierarchy, which is based on their assumed effectiveness, implies that these parameters may have an impact on each other. In this thesis, we perform a study of both frame skipping and DVFS, where both are adapted simultaneously to observe if such impact exists using these parameters in Section 3.4.3.

The work in [23] has been extended in a very recent publication [2], which we use for comparison and to highlight the major differences to the work presented in this thesis in Section 5.3.1. We compare several aspects, including the used characterisation metrics for tracking difficulty, runtime control, the evaluation of performance objectives, and the nature of algorithms and sensors used.

### 2.3.5.1 Characterising the Tracking Difficulty

Effective runtime adaptation requires accurate characterisation of the tracking difficulty to ensure minimal impact on the accuracy. Characterising the change in VSLAM scene state with a small number of metrics was proposed in [82]. Saeedi et al. used the Kullback-Leibler (KL) Divergence [100] as a measure for the distance between normalised intensity and depth histograms generated from two consecutive pairs of intensity and depth frames. Saeedi et al. show empirically that higher divergence leads to less accurate tracking and propose the use of such metrics in multiple use cases, such as the characterisation of datasets, runtime adaptation and frame management.

Multiple characterisation metrics have been proposed to characterise datasets in an offline context without relying on VSLAM algorithms. For example, in [101] multiple metrics are used to characterise the difficulty of datasets extracted from movement through a forest, as compared with trajectories through more structured and static datasets. One of the metrics used is the median value of all consecutive KL Divergence distances between intensity histograms. Relying purely
on intensity histograms without depth frames, which may not be available depending on the sensor used, is limited because it only captures the changes in the features’ illumination and does not account for their actual position within the frame. Our pyramid-based EMD metric does take into account the changes in illumination as well as spatial information, without the need for depth, making it more applicable to a wider range of sensors.

In very recent work in [61], the diversity of motion patterns in existing dataset sequences is explored as a characterisation metric by extracting the principal components of each of the translation and rotation motion. Saeedi et al. [102] and [60] propose an optical flow-based metric where the first paper proposes, as a measure, the second-order Wasserstein distance on the range and baring for pixel flow, while the latter paper uses the Farnebäck method [103] to measure the rate of pixel flow directly, and where the values are normalised on frame resolution or the camera focal length. A key difference between these two approaches is the decoupling of the motion sources in the former, where the distance is measured on each source. In online contexts, the motion sources (translation and rotation) are estimated by the VSLAM formulation or using an Inertial Measurement Unit (IMU), where each source may be on a different scale. Thus, in this thesis, we adopt a similar approach by decoupling the two, which we find to be useful, as is discussed in Section 4.2 and evaluated in Section 5.2.

Further, computing optical flow, given it is not normally available, would add significant extra overhead. Applying the same principle of using pyramids to reduce the computation time for computing optical flow is possible. However, we favour using the pyramid-based EMD since it also accounts for sudden changes in brightness, while optical flow assumes its consistency. In this thesis, we increase the reliability of estimating the tracking difficulty by relying on both motion patterns (decoupled translation and rotation) and the visual change in the scene. These changes in the observed scene are not exclusive to only the movement of the sensor. For example, the scene can change based on the brightness or due to the presence of dynamic objects. We demonstrate that the pyramid-based EMD used in conjunction with the readily available motion information provides an accurate and efficient online characterisation of the tracking difficulty. This combination adds extra reliability to the estimated tracking difficulty, unlike the single metric used in [22] which is based on pure rotation.

In the work presented in [23, 2] a translational-based metric is used with a
smooth surface detection. We argue that different VSLAM algorithms behave differently in the presence of a completely smooth surface where there are no tractable features for certain window frames. If this window is larger than the window used for tracking in the algorithm, it will lose the track temporarily and track can only be regained by revisiting a previous location (given that the method is a full VSLAM and not simply a VO). This implies that the need for a metric that detects smooth surface is subject to the length of the algorithm tracking window, and for the sake of keeping the adaptation framework minimal and portable, this metric was not explored in this thesis.

Learning the similarity of frames is also possible with the use of one-shot learning, which is based on the so-called Siamese Convolutional Neural Network [104, 105]. The resulting output is a distance that is smaller when the two frames are similar and larger otherwise. The possibility of applying this approach can be explored in future inquiry to replace or work in conjunction with the proposed EMD pyramid-based metric.

2.3.5.2 Runtime control

There is a wide spectrum of approaches to improve runtime efficiency; some are generic, while others are application-specific. The generic approaches do not assume any knowledge about the nature of the target application which can be utilised to provide extra improvements. Application-specific approaches, however, especially in the context of VSLAM, are usually too specific to a certain formulation, environment or platform, and require redesigning or tuning if any of the settings changes.

The goal in this thesis is to bridge the gap between the two ends of this spectrum by providing an approach which is both portable to different settings and takes advantage of the nature of VSLAM, in a general way, to balance the runtime performance and robustness.

POET [106] is one of the prominent examples of such a generic approach. It aims to improve energy efficiency under a soft real-time constraint in a portable fashion and across different micro-benchmarks. This is achieved by predicting the completion time required for the target application tasks and saving power without violating that predicted deadline, given the available compute resources. This is under the assumption that the same, repeated jobs will have similar user-specified base times. In the case where this time changes during the phase of
execution, POET employees a Kalman filter [107] to adjust accordingly.

In the context of VSLAM, each implementation is, in a sense, a macro-benchmark which can be evaluated in a standalone fashion or using one of the tools described in Section 2.1.5. VSLAM implementations are usually based on multiple execution threads which perform different and coordinated tasks such as tracking, mapping, loop closure and more. Some of these tasks are not executed on a frame-by-frame basis and differ in the implementations based on the type of VSLAM used. In addition, the amount of computation performed is affected by the changes in motion and observed in the environment, which are not known in advance. Even when running the VSLAM on the same scene repeatedly, there is variability from run-to-run, though this occurs mainly on formulations with non-deterministic behaviour. This leads to a per-frame time that is highly variable and non-linear, making specifying a base time for processing in a particular VSLAM challenging and often infeasible. Thus, using time alone is not adequate for guiding the runtime control for the adaptation.

At the other end of the spectrum are approaches that are specific to certain VSLAM formulations or to a particular target environment or sensor [22, 23, 2]. These approaches usually use information about the VSLAM to facilitate runtime control and parameter tuning. For example, in [22] frame skipping is controlled based on a rotation-based threshold applied to the stereo-based autonomous driving dataset, KITTI.

In [23], a PID controller was used to improve the performance of KinectFusion [49], a dense form of VSLAM, by adapting a selected set of its parameters based mainly on the sensor translational motion. In their recent work [2], a degree of portability is demonstrated, where several VSLAM algorithms, operating in the linearised mode described in Section 2.2.2, are used, each with a specific type of dataset dominated mainly by RGB-D sensors. Application-specific parameters are used to tune each algorithm in a coarse manner. To adapt to different environments, Pei et al. rely on the recorded min and max value of the translation distance, which is similar to an extent to our strategy presented in [88].

In terms of having an internal prediction validated by external observations, the adaptation system may rely on external information sources such as IMUs, GPS or beacons to validate the estimated VSLAM pose or enhance the estimation reliability. However, the challenge arises when such external information is not available in the deployment environment, and this was the main focus of this
thesis. In [23, 2], a form of extrapolation of the pose distance is used to reject outliers by assuming that movements between subsequent frames are small. In the case where the distance exceeds a certain threshold, extrapolation is performed. In our case, the framework does not make any assumption about these outliers correctness in the different distance metrics used; its presence is treated as difficult tracking (The highest settings are used), and a filtering window is employed to minimise their effects on the adaptation process. This enables the adaptation policy and control model to cope with changes, as long as the VSLAM system is capable of doing so. This approach has the advantage of eliminating the pose correctness assumption to support our portability goal. In the future, incorporating external information sources such as IMUs can be explored and used to enhance the reliability of the pose estimation besides the information from the observed scene.

We demonstrate the portability of our framework to a greater degree with very challenging settings and constraints. We apply the same adaptation framework using portable parameters on two classes of sparse VSLAM algorithm, relying purely on monocular sensors, where scale drift can affect the distance metrics, and under hard deadline constraints on two datasets and two platforms. Unlike [2], we evaluate the impact on robustness for algorithms with non-deterministic behaviour. Further, parameter tuning can be performed at very fine levels or steps over the parameter’s range. The adaptation policy and control algorithm are portable and are easier to maintain compared to PID controllers, which usually require significant tuning efforts. In addition, our framework provides an essentially instant adaptation response since it responds based on the dynamically monitored variations in the metrics measured distance, with low overhead.

Currently, the framework relies on the divergence of the characterisation metric distances from an average of maximum peaks window as feedback to guide the control process. However, parameter tuning is not the only aspect that affects these distances; The tuning ensures that the system works at its highest capacity in the case of high divergence. If the tracking is easy, for example, applying the highest or lowest setting of a parameter can result in very similar and accurate trajectories but with implication on, for example, power efficiency. Currently, the framework does not implement active control over the platform movements or speed. For example, the framework may control the platform movements based
on a reference distance that balance or improve performance objectives, while using the current distance in the feedback loop to adjust the platform movement. Such control is platform-specific and beyond the scope of this thesis.

2.4 Summary

In this chapter, we have presented the background and related work of the thesis. In the background section (2.1), a brief introduction to VSLAM components was provided, and background on VSLAM formulations and common sensors presented. Followed by VSLAM benchmarking frameworks and datasets. We finally discussed the performance metrics used for the evaluation in this thesis (Section 2.2). In the related work section (2.3), we presented and reviewed different approaches from the literature focusing principally on those aiming at improving VSLAM performance at the design and runtime stages and pointed out certain limitations which the research in this thesis addresses.
Chapter 3
Preliminaries

This chapter motivates the need for improving the runtime performance of VSLAM. It also establishes the basis and methods to achieve this goal. First, the case for VSLAM runtime adaptations is discussed in Section 3.1. Then, Section 3.2 presents the argument for the need for a lightweight and general approach that utilises VSLAM ego-motion and its incremental nature. It also describes the basic idea of the approach for reducing power consumption with minimal impact on the accuracy and in a real-time setting. Section 3.3 described the general experimental setup used in this chapter and Chapter 5. An evaluation, in a simulated environment, of two portable parameters and their relation to the characterisation metric, is presented in Section 3.4. In Section 3.5, an evaluation of the base adaptation model is presented on realistic environments and motivates further exploration.

3.1 The Case for VSLAM Runtime Adaptation

In an environment where a VSLAM sensor experiences changing motion, VSLAM performance objectives such as power consumption and accuracy can be balanced in accordance with the change. For example, if the sensor moves slowly and steadily, the observed scene may be essentially redundant within the tracking frame window. In this case, power can be reduced with little impact on accuracy, given that the observed scene is static and rich in tracking information.

One key advantage of using the sensor motion change for guiding the adaptations is its applicability and portability across different VSLAM applications, independent of the portability of the tuned parameters. This is because VSLAM
3.1.1 The Synthetic Dataset

To illustrate the approach, a synthetic sequence is designed to isolate certain aspects of tracking in order to observe the behaviour of different VSLAM algorithms, mainly by having a single varying type of motion but with adequate tracking scene information. To achieve this, a simulated down-facing monocular camera observes a synthetic scene which was generated using the open-source software Gimp [108]. The scene is blur-free and rich in features throughout the sequence. This is to ensure robust tracking on different VSLAM algorithms and to rule out any effects that may be caused by the changes in the observed scene itself.

Figure 3.1a shows a sample frame of such a scene. As can be seen, the texture is rich in detail, having both edges and distinct pixel intensities distributed across the frame. This allows both direct and indirect VSLAM formulations to have the appropriate tracking information or the so-called key-points. Figures 3.1b and 3.1c demonstrate this by showing the distribution of key-points, on the same frame (Figure 3.1a), tracked with the direct sparse odometry (DSO) and the indirect ORBSLAM2, respectively. The key-points are well-spread, which implies that both algorithms are capable of utilising information from the whole frame, increasing the tracking accuracy and robustness.

Another property of this sequence is that the camera has a pure translational
3.1. THE CASE FOR VSLAM RUNTIME ADAPTATION

Figure 3.2: The camera trajectory in meters (top subplot) aligned with camera travelled translational distance (bottom subplot).

<table>
<thead>
<tr>
<th>Frames</th>
<th>Duration</th>
<th>Path length</th>
<th>V Min.</th>
<th>V Avg.</th>
<th>V Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3745</td>
<td>124.8 s</td>
<td>88.5 m</td>
<td>0.17 m/s</td>
<td>0.71 m/s</td>
<td>1.11 m/s</td>
</tr>
</tbody>
</table>

Table 3.1: The synthetic dataset properties.

and sinusoidal motion with slow speed on the sine wave curve peaks but being faster in between, as shown in Figure 3.2; this means that the observed scene is most redundant and easier to track at the peaks. Such motion allows us to observe the effect of varying motion on different VSLAM algorithms and explore whether it can be taken advantage of to improve runtime performance.

A summary of the dataset properties is shown in Table 3.1. A total of 3745 frames is generated and associated with time-stamps with a fixed frame rate of 30 FPS. The frames have a resolution of 640 by 480 px. which is typical in most synthetic datasets [59, 102]. The dataset has a path length of 88.5 meters and an average velocity of 0.71 m/s which is comparable to an extent to those reported in the EuRoC MAV dataset sequences used for evaluation in later chapters.
3.1.2 VSLAM Behaviour on the Dataset

We use the synthetic sequence to observe the behaviour of a direct and an indirect VSLAM formulation, DSO and ORBSLAM2, respectively, in terms of how the runtime computations are affected when the speed varies. To facilitate this, we pinned the main two threads of each algorithm, the tracker and mapper threads, to separate CPU cores and measured the number of cycles executed per second along with the power consumed by both cores. The measurements were obtained using Linux Turbostat on the same desktop machine (see Section 3.3) with the same frequency scaling governor (Performance) and under a fixed frame rate (30 FPS). Although both ORBSLAM2 and DSO perform tracking based on keyframes, their computation intensity is partially influenced by the camera dynamics. We ran both ORBSLAM2 and DSO on the synthetic dataset to observe how they are affected in terms of computation intensity and power consumption. The default settings on both algorithms are used.

![Graphs showing runtime power consumption and computation intensity](image)

Figure 3.3: Runtime power consumption and computation intensity of the main threads of ORBSLAM2 (a) and DSO (b), running a synthetic dataset with varying camera speed. ORBSLAM2 tracker and mapper threads are insensitive to the variations. DSO mapper is less sensitive to the variations compared to its tracker, but shows a larger impact on power consumption.

The amount of computation performed by ORBSLAM2 tracker and mapper threads does not seem to have a strong correlation with the change in the camera motion, shown in Figure 3.2, as can be seen from the three subplots of Figure 3.3a. This is more prominent for the ORBSLAM2 tracker thread, while its mapper thread, counter-intuitively, seems to be doing more computations when the movement is slower. On the other hand, DSO tracker and mapper threads are
affected to a certain degree by the change in motion, as can be seen in Figure 3.3b, where relatively less power consumption can be observed when the camera moves slowly. Looking at the same figure, the DSO mapper is more dominant in terms of the amount of computation and shows more impact on power consumption compared to the tracker. Still, it is less sensitive to the change in camera motion. Given the fact that both ORBSLAM2 and DSO algorithms take little advantage of slow camera motion and redundant scenery to perform fewer computations for saving power, we see an opportunity to perform runtime adaptations based on the sensor motion to reduce power consumption with minimal impact on the VSLAM accuracy.

3.2 The Basic Adaptation Model

To have an effective runtime adaptation, knowledge about the nature of the variation in the camera motion is required to be able to characterise the tracking difficulties attributed to the motion. In practice, the type of variations in the camera motion will depend on its hosting platform, e.g. drones or smartphones, and will usually be unpredictable in advance. In extreme cases, motion can be highly nonlinear and jittery without knowing its extremum. The adaptation model, in this case, needs to be sensitive and responsive to these variations while being able to cope with their unpredictability.

Another challenge arises when the hosting platforms do not have sensors that quantify motion, such as an Inertial Measurement Unit (IMU), or when the VSLAM algorithm does not support these sensors. In such a case, the variations in motion can only be obtained by the VSLAM algorithm being used through metrics which characterise the tracking difficulty. The efficacy of these metrics will largely depend on the accuracy of the estimation of tracking difficulty and the type of camera sensor used. VSLAM algorithms are becoming more sophisticated and accurate, making such metrics attractive as they can be computed with minimal additional cost while being portable across different VSLAM algorithms.

In terms of the camera sensor type, well-established VSLAM algorithms such as [36, 35] and [45] continue to support a wide variety of sensors. For the sake of portability, the adaptation model is required to be applicable to different VSLAMs regardless of the sensor used. The challenge is most severe when relying purely on monocular sensors where the actual scale of travelled distance cannot
be retrieved during the tracking [57, 65]. This is beside the presence of scale drift, or when different algorithms with monocular sensors initialise to different scales. In our experiments, the scale varies from run-to-run on the same combination of algorithm, sequence and machine, in addition to the differences in the scale between different algorithms and/or datasets. This makes models that rely on many predefined thresholds in their adaptation algorithms which transform sensor data into estimates of tracking difficulty, such as [22, 23], difficult to maintain and lack portability on monocular sensors. Figure 3.4 shows such variations in the form of corrections that have to be applied to the scale relative to the ground truth trajectory for a number of scenes taken from existing datasets (Further details of these datasets can be found in Section 2.1.4 and 3.3).

The challenges described above dictate designing a generalised and lightweight adaptation model to meet the performance portability goal across the diverse problem settings. These settings include unknown deployment environments, different VSLAM algorithms - with deterministic and non-deterministic behaviour - observing the scene with different sensors, and/or hosting platforms with different computing capabilities. This thesis aims to investigate to what extent
performance portability can be achieved while addressing these issues in a systematic top-down way, starting with a minimal basic model and subsequently adding extensions, in response to experimentation and evaluation.

![Figure 3.5](image_url)

Figure 3.5: When the change D (green line) is at its historic peak (blue line), the parameter $X_p$ (red line) is at its maximum level.

The adaptation model, in its essence, is based on normalising the observed change in a monitored metric at runtime to a value between 0 and 1. The resulting normalised value provides a standard way to use any suitable metric to characterise the tracking difficulty in order to adapt different tuning parameters.

The normalisation is a progressive process which inherently enables coping with unknown variations in the metric. Let $m_t$ be any monitored metric measured at time $t$, the goal of the adaptation model is to adapt $m_t$ to an appropriate value, $m'_t$, by normalising to the maximum peak value observed so far, $m_j$, where $j \leq t$, and where $m'_t$ is defined as:

$$m'_t = \frac{m_t}{m_j}$$

(3.1)

The resulting value is always between zero and one, where zero implies the easiest tracking, while one implies difficult tracking. This minimal model is computationally efficient and easily maintained, since it can adapt to changes in many metrics capable of monitoring the current VSLAM state.

The normalised metric value, $m'_t$, is then used to tune a related parameter, say $X_t$, within a predefined range: $\{X_t | X_{min} \leq X_t \leq X_{max}\}$; for example, a motion metric can be used to determine the number of frame skipped. The tuning is based on the relation between the metric and the parameter, i.e. how the parameter value changes and is correlated with the monitored metric value (which is a characterisation of the tracking difficulty). The correlation can be established
intuitively or experimentally, as demonstrated in the following sections. Based on the correlation type (positive or negative) between the scaled change, \( m'_t \), and the applied change to the controlled parameter, \( X_t \), the model can be written as the following for positive correlation (resulting in \( X_{p_t} \)):

\[
X_{p_t} = X_{\min} + (X_{\max} - X_{\min})m'_t
\]  

(3.2)

And for negative correlation (resulting in \( X_{n_t} \)) as:

\[
X_{n_t} = X_{\max} - (X_{\max} - X_{\min})m'_t
\]  

(3.3)

This model can be used for adaptation of parameters associated with both the VSLAM formulation and the target hardware platform, as demonstrated in this work.

The advantages of this approach are two-fold: first, it provides the ability to adapt and respond quickly to larger changes as they are encountered by the VSLAM without the need for prior knowledge about the nature of motion or scene. Assuming positive correlation, for example, when the change is at its peak \((m_t = m_j)\) the model immediately ensures that the adapted parameter operates at its highest value, leading to enhanced robustness, as illustrated in Figure 3.5. The second advantage is that it decouples the variation in the metric from the adaptation process, which means the model can work with any metric. In this chapter, we use metrics based on the change in sensor motion as a heuristic for guiding the adaptation process. In the case of constant motion, the control parameter \( X \) will operate on its highest or lowest level, depending on the correlation. In such a case, the value \( X_{\max}/X_{\min} \) can easily be tuned to achieve the desired tradeoff.

The model is used next to explore two general portable parameters that do not involve significant changes to the VSLAM base code and default settings, which can be useful if the formulation parameters and settings are already tuned through design space exploration, for example. Further, these two parameters can be applied to a variety of VSLAM formulations and platforms. The first parameter attempts to identify and skip redundant frames dynamically before they are fed to the VSLAM system, thereby indirectly affecting power consumption. The second targets the platform through DVFS adaptations, thus directly influencing power consumption.
3.2.1 Translational Motion-based Metric

Given that the presented synthetic sequence, described in Section 3.1.1, has pure translational motion, it can be used to characterise the tracking difficulty where faster motion implies more difficult tracking, while slow-motion implies higher redundancy in the observed scene. Where in this latter case, the tracking can be characterised as easy. We refer to these regions as **non-critical regions**.

The 3-D Euclidean distance is used as a metric which is measured between the translational parts of the most recent camera poses at time $t-1$ and $t-2$.

$$\Delta T = \sqrt{(x_{t-1} - x_{t-2})^2 + (y_{t-1} - y_{t-2})^2 + (z_{t-1} - z_{t-2})^2} \quad (3.4)$$

Equation 3.1 can then be used to normalise $\Delta T$, and the resulting metric, $T'$, is used to adapt the different tuning parameters. In this thesis, we focus mainly on the two parameters: frame skipping and DVFS. These two parameters are found to be effective and portable, as demonstrated in this thesis.

3.3 General Experimental Setup

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Sequence</th>
<th>Frames/Length (m)</th>
<th>Avg. Vel. (m/s)</th>
<th>Scene</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ICL-NUIM</strong></td>
<td>LR0</td>
<td>1510/6.54</td>
<td>0.126</td>
<td>Living Room</td>
</tr>
<tr>
<td>(Synthetic @ 30 FPS)</td>
<td>LR1</td>
<td>967/6.73</td>
<td>0.063</td>
<td>Living Room</td>
</tr>
<tr>
<td></td>
<td>LR2</td>
<td>882/8.43</td>
<td>0.282</td>
<td>Living Room</td>
</tr>
<tr>
<td></td>
<td>LR3</td>
<td>1242/7.83</td>
<td>0.263</td>
<td>Living Room</td>
</tr>
<tr>
<td><strong>EuRoC MAV</strong></td>
<td>MH01</td>
<td>3652/80.6</td>
<td>0.44</td>
<td>Machine Hall/Easy</td>
</tr>
<tr>
<td>(Real @ 20 FPS)</td>
<td>MH02</td>
<td>3040/73.5</td>
<td>0.49</td>
<td>Machine Hall/Easy</td>
</tr>
<tr>
<td></td>
<td>MH03</td>
<td>2700/130.9</td>
<td>0.99</td>
<td>Machine Hall/Medium</td>
</tr>
<tr>
<td></td>
<td>MH04</td>
<td>4083/91.7</td>
<td>0.93</td>
<td>Machine Hall/Difficult</td>
</tr>
<tr>
<td></td>
<td>MH05</td>
<td>2273/97.6</td>
<td>0.88</td>
<td>Machine Hall/Difficult</td>
</tr>
</tbody>
</table>

Table 3.2: EuRoC MAV and ICL-NUIM properties.

This section describes the general setup for the experiments used for the evaluation in this chapter and Chapter 5. The experiments aim to evaluate the accuracy, robustness and power reduction obtained after applying the framework to two well-established VSLAM systems, ORBSLAM2 [35] and DSO [1]. Both are sparse formulations; however, the latter is direct visual odometry as opposed to ORBSLAM2, which is an indirect VSLAM with loop closure. To account for the non-deterministic runtime behaviour of the two algorithms, and test whether the robustness is impacted, we perform ten consecutive runs on each sequence. Violins plots are used to show the distribution of these runs across different
performance metrics. The median value is marked with a black horizontal line while the mean is marked with a white line. These violin plots are insightful since they provide better visualisation of the impact on robustness compared to only using, for example, box plots or reporting the mean or median values of the ten runs.

We evaluate a selected metric using the synthetic sequence and also scenes from two different standard datasets. The first dataset is the Machine Hall (MH) scenes from EuRoC MAV [11] which have real footage of an industrial environment captured with a stereo camera mounted on a drone. The scenes differ in length, speed and difficulty. The sequences in this dataset start with a shaky motion for calibrating the MAV IMU and end with a hard landing of the drone. This may result in a bad initialisation or a large discrepancy in the estimated trajectory for DSO. To ensure robust tracking on these sequences, we use the frames where the drone is airborne, in accordance with what has been done in [1]. In contrast to DSO, we use the whole sequence for ORBSLAM2. The second dataset is the ICL-NUIM living room sequences which have realistic trajectories estimated with the Kintinuous VSLAM system in a real-world environment. These trajectories are then used to generate RGB-D frames in a synthetic 3D model [59]. Table 3.2 shows a summary of the two datasets.

We use the monocular version of both systems; thus, we convert the coloured frames of ICL-NUIM to grey-scale frames and use the left frames of EuRoC MAV. The use of the monocular sensor is discussed in Section 3.2. In this chapter we target mainly a desktop platform with an Intel Skylake i7-6700 multi-core and 16 GB of RAM [109], running Ubuntu 16.04, to establish the basic concepts behind the framework. Linux TurboStat [76] is used for recording power measurements on this platform. A second platform is an Nvidia Jetson AGX Xavier robotics platform which is used mainly for the evaluation in Chapter 5 and 6. The Jetson has a System on Chip (SoC) consisting of ARM-based Carmel processing cores and a Volta-based GPU. It has 16 GB of RAM and runs Ubuntu 18.04. Power consumption is calculated using the software-based power consumption monitor available with the Jetson platform [80]. We use the default frequency scaling drivers on the Jetson and the `acpi-cpufreq` driver on Intel multi-core. To facilitate DVFS tuning, we pin the main thread/threads of the formulations to separate processing cores, while using the Performance Governor and adjusting the maximum allowed frequency (as one of the adaptation parameters). The two
platforms are denoted as i7 and Jetson, respectively, throughout the thesis.

We use the same default settings for ORBSLAM2 and DSO on both platforms, the only difference being the conversion of the Intel SSE instructions used in DSO into ARM NEON instructions, to be utilised on the Jetson. We evaluate the translational absolute trajectory RMSE on both ORBSLAM2 and DSO using EVO benchmarking framework [63] after scaling and aligning the trajectory.

We evaluate the framework against the baseline (denoted as the base) of each VSLAM system. The baseline version of each system is run at the highest frequency available on each processor without any forms of adaptation, or CPU throttling (For example, Intel Turboboost is disabled). Both ORBSLAM2 and DSO baseline cases are run at a fixed frame rate, defined by the dataset, which both formulations are capable of achieving. If the frame rate constraint is not met in a particular time during the tracking, frames are skipped to maintain the QoS; this is similar to what has been done in [1]. Since a fixed frame rate is maintained, the total execution time is fixed for each particular sequence. The average power consumed during the tracking is computed, and the reduction in power use relative to the baseline is reported.

### 3.4 Parameter-metric Relationships

It is desirable to observe how the accuracy and robustness are impacted when the adaptations are applied in comparison to that of the base VSLAM by evaluating the relationship between the sensor motion and different parameters adaptations. The correlation between the two can be established intuitively since VSLAM is affected by how fast the sensor moves, where slow-motion usually implies easier tracking, given adequate tracking information. For example, a higher number of frames can be skipped if the change in the motion metric is small, leading to a minimal impact on the accuracy due to the large overlap between the tracked frames. This implies that a negative correlation for frame skipping is required. On the other hand, higher values of DVFS allow computation to be performed faster in order to meet the QoS when the change is large to ensure robust and accurate tracking in such a critical situation. When the change in the measured metric is small, implying slow movements, dropping the frequency would lead in the worst case to frames being skipped in non-critical regions. Thus DVFS, intuitively, has a positive correlation.
CHAPTER 3. PRELIMINARIES

In this section, we use the synthetic dataset to experimentally confirm the correlation between the two portable parameters and the rate of change in the selected motion characterisation metric using the models described in Section 3.2. Since the synthetic dataset has a pure translational motion, the parameter adaptation is applied using the \( T' \) metric introduced in Section 3.2.1 but with negative and positive correlations. Each of these adaptations is compared against the base version. In addition, for frame skipping we evaluate a case where frames are skipped in a controlled fashion, similar to loop perforation used in approximate computing [110], while for DVFS, two governors are used in the comparison. This is to highlight the need for guided adaptation based on characterisation metrics.

### 3.4.1 Frame Skipping

![Graphs showing effects of frame skipping on VSLAM accuracy and robustness](a) DSO (b) ORBSLAM2)

Figure 3.6: The effects of increasing the maximum batch size of skipped frames on the synthetic sequence.

Frame skipping can have a direct impact on VSLAM accuracy and robustness; the goal in this section is to generally establish how skipping frames affects VSLAM and how this parameter relates to the metric \( T' \). Three cases are evaluated against the base. These are: skipping frames based on negative correlation with the metric \( T' \) (the intuitive correlation), a manually controlled naive frame skipping, and finally, skipping frames based on positive correlation.

For the first case, a negative correlation, Equation 3.3, is used with the normalised metric \( T' \). In this case, both DSO and ORBSLAM2 are stressed by
increasing the maximum skipped frame batch size $X_{\text{max}}$, in separate experiments (each consisting of ten runs), up until the algorithm loses track in at least one run. The result of this test is shown in Figure 3.6 where a maximum batch of four frames ($X_{\text{max}} = 4$) is the highest number that both DSO and ORBSLAM2 could handle on this sequence before the accuracy and robustness is significantly impacted. The impact on accuracy and robustness with a batch size of up to 4 is negligible relative to that of the base of each algorithm but, significantly, there is up to 39.7% and 50% improvement on DSO and ORBSLAM2 power consumption, respectively.

Breaking at the same value of $X_{\text{max}}$ (labelled 5 in Figure 3.6) in this experiment with the synthetic sequence is expected for both algorithms since both track key-points uniformly from across the frame, as shown in Figure 3.1b and 3.1c. However, it should be noted that this may not be the case in realistic scenarios; the value will depend on how each algorithm utilises the information in the real scene, and how fast is the sensor motion. Thus, naively skipping frames has clear implications for portability.

Based on the violin plots in Figure 3.6, DSO is seen to be more accurate and robust than ORBSLAM2 on this trajectory; thus, we use it to study the parameter-metric relation in the second and third cases by comparing with the result from the first test, with $X_{\text{max}} = 4$ using DSO. This setting results in skipping more than half of the frames from the synthetic sequence. Thus, in the second case, frames are skipped in a controlled and repetitive fashion with no attempt to consider the significance of the selected skipped frames on the tracking ability of the algorithm. This is achieved by skipping a single frame in a fixed and frequent manner (alternating between skipping and processing frames) so that the total number of skipped frames will be less than or equal to the total number of frames skipped in the first case. This is to ensure the same power savings are achieved in both cases. To an extent, this is similar to loop perforation with a skip factor of 1, but at the system level rather than at the kernel level. The third case is similar to the first case, except that the correlation is positive with respect to the used metric.

Figure 3.7 shows the result of the three cases relative to the base real-time version (Red). As mentioned above, skipping frames based on negative correlation achieves similar accuracy and robustness to the base case (conforming with the intuition) but with a relatively average power reduction of 39.7%. For the
controlled frame skipping case, the accuracy and robustness are significantly impacted with the loss of tracking toward the end of the sequence (thus the slight increase in power reduction compared to the first case). This is due to skipping frames in critical regions of the trajectory because of the arbitrary choice of controlling the skipped frames in a simple, repetitive fashion. This implies that the same power and accuracy achieved in the first case cannot be obtained with simple loop perforation. The third case is the worst-case scenario where the highest number of frames are skipped in the most critical regions where the change in the normalised metric $T'$ is at its peaks.

These experiments confirm that a negative correlation is appropriate for frame skipping and demonstrates clearly the capability of the model to achieve more power-efficient tracking with accuracy and robustness similar to that of the base if appropriate frames can be selected for skipping.

### 3.4.2 Dynamic Voltage and Frequency Scaling (DVFS)

DVFS has an indirect impact on VSLAM since it only affects per-frame processing time which may lead to skipping frames in order to meet the QoS constraint. The DVFS range is chosen experimentally in this test so that this effect is minimised (resulting in a choice of the frequency range between 1.0 - 3.4 GHz inclusive on
3.4. PARAMETER-METRIC RELATIONSHIPS

Figure 3.8: The behaviour using different methods to adapting DVFS.

the i7). Another possible impact on robustness may also result from the multi-threaded nature of the DSO and ORBSLAM2 algorithms which involve thread synchronisation, as discussed in Section 2.2.1.2.

Similar to the previous experiments, we evaluate positive and negative correlation with respect to the metric $T'$, where the tuned DVFS value is rounded to the available hardware frequency steps. Further, we evaluate the performance when using the ondemand and powersave governors with Intel Turboboost being enabled by default.

The results are shown in Figure 3.8. The ondemand governor, which is the default governor on the acpi-cpufreq scaling driver on the i7, is more power-hungry (about 26% increase in power consumption over the base) with no added improvement in accuracy and robustness compared to the DSO base. The reason for the increased power consumption is that when Intel Turboboost is enabled, the ondemand governor can use higher frequencies which are not accessible to the user [111], and since the base maximum frequency used here (3.4GHz, as mentioned above) is less than these frequencies, this increase in power is expected. This shows that the default governor is not power efficient.

On the other hand, the powersave governor shows reduced power consumption but at the significant expense of the accuracy and robustness. This implies that at low frequencies, frames are being skipped in regions that are critical to maintaining the QoS. Both of these governors follow predefined general policies
that do not utilise specific information about the workload or its characteristics.

When adapting based on positive correlation (coloured in orange in the figure), an average of 40% power reduction is achieved with similar accuracy and robustness to that of the base. In contrast, when a negative correlation is used for the adaptation, this leads to a major impact on the accuracy and robustness, though with a similar impact on power.

### 3.4.3 Parameter Combination Behaviour

This section explores whether the two parameters, frame skipping and DVFS, have a negative impact on each other in terms of the accuracy and robustness when combined and adapted together. We perform this test on both DSO and ORBSLAM2 using the settings from the previous two sections, (3.4.1 and 3.4.2), which shows that the desired goal of saving power with similar accuracy and robustness to that of the base can be achieved. That is, skipping a maximum frame batch size of 4 with negative correlation and using the same DVFS range used in the previous section with a positive correlation.

Figure 3.9 shows the two parameters combination (in Green) relative to adapting each parameter individually, DVFS (in Orange) and frame skipping (in Blue). The combination has similar accuracy and robustness to the base, but with extra
3.5. **PERFORMANCE ON ENVIRONMENTS WITH REALISTIC MOTION**

Figure 3.10: Runtime power consumption and computation intensity of ORBSLAM2 (a) and DSO (b) main threads, running a synthetic dataset with varying camera speed while adapting both DVFS and frame skipping using the motion metric, $T'$. 

Power reduction, with an average of 66% for DSO and 79% for ORBSLAM2, compared to adapting each individual parameter alone. ORBSLAM2 seems to benefit more from frame skipping compared to DSO likely due to it being a slightly more computationally intense algorithm.

These results imply that combining both parameters does not have a negative impact on accuracy and robustness and can deliver increases in power savings. This combination is therefore used throughout the rest of this thesis due to its ability to reduce power consumption.

Figure 3.10 shows profiles of the effect on ORBSLAM2 and DSO computational intensity (as reflected by the average DVFS frequency throughout the sequence) and power consumption after adapting both parameters. This Figure, when compared with Figure 3.3, shows both the tracker and mapper threads computations are most reduced where the change in the translation distance metric is at its minimum, as can be seen along the time axis in Figure 3.1, leading to a power reduction in these non-critical regions. Relying on characterisation metrics such at $T'$ for the adaptation of tunable parameters like frame skipping and DVFS forms the basis of the remaining research in this thesis.
3.5 Performance on Environments with Realistic Motion

The previous results showed the effectiveness of the presented basic adaptation model, described in Section 3.2, and the pure translation metric, in producing more power-efficient tracking with accuracy and robustness similar to that of the base. In this section, the goal is to explore and evaluate the performance of the basic adaptation model on well-known and widely used datasets, EuRoC MAV [11] and ICL-NUIM [59]. That is, we aim to tune the adaptation parameters’ range to these datasets and only evaluate the performance adaptation model and a characterisation metric $P_t$. These datasets have realistic and challenging motion patterns combining both rotation and translational motion. Further, the scene varies in terms of difficulty due to the presence of smooth surfaces, dark scenes and sudden changes in exposure. This prompted the initial use of the new metric, $P_t$, that combines both rotation and translation to characterise the tracking difficulty, under the assumption that scene in these sequences has adequate
3.5. PERFORMANCE ON ENVIRONMENTS WITH REALISTIC MOTION

and sufficient tracking features. The metric is defined as follows: For each frame fed to VSLAM, there are estimated translational and rotational values computed for the camera pose. The most recent pair of pose values are available at time \( t - 1 \) and \( t - 2 \) but are used to make decisions at time \( t \). The change in camera motion is defined at time \( t \) between two subsequent frames as:

\[
P_t = \Delta_p^T \Delta_p
\]

(3.5)

where \( \Delta_p \) is the absolute difference between camera pose at time \( t - 1 \) and \( t - 2 \).

The metric \( P_t \) can then be used with Eq. 3.1 to guide the adaptations.

Since this metric relies on the resulting VSLAM poses to be computed, it is essential for its accuracy that these poses are outliers-free. However, this cannot be guaranteed even in the resulting base trajectory, which may contain locally inaccurate poses. The problem that the presence of these outliers poses is that they are not consistent from run to run, leading to variability in the results of formulations with non-deterministic behaviour such as DSO and ORBSLAM2. Figure 3.11 shows an example of such poses from one of the ten runs of the base DSO on the difficult sequence \texttt{MH04}. To tackle this problem, a running geometric mean window is introduced on the \( n \) latest values of the measured normalised metric \( P' \). The results of this test are collected under a fixed-frame rate.

Figures 3.12 and 3.13 show the initial exploration of datasets with realistic motion patterns. The figures show the impact on accuracy and robustness along with the power reduction achieved relative to the base. The ORBSLAM2 base case fails to initialise on \texttt{LR1}, thus this sequence is removed from the ORBSLAM2 plots. Overall, applying the adaptation to ORBSLAM2 shows greater benefit in terms of power reduction compared to DSO. This is consistent with analysis on the tracker and mapper threads behaviour of each algorithm in Section 3.1.2. It should be noted that ORBSLAM2 has the advantage of being a full VSLAM algorithm compared to DSO, which is only visual odometry, and therefore does not correct and re-localise the track in the case of loop closure. This is apparent in the overall accuracy and robustness results in Figure 3.13a showing EuRoC MAV sequences.

Considering the results for each sequence and the accuracy results obtained from applying the metric, they fall into three categories based on the impact on the accuracy and robustness compared to the base:

**Improved Robustness:** Applying the adaptation of both frame skipping
Figure 3.12: Violin plots of benchmarking the framework on the i7 and using ICL-NUIM sequences.

and DVFS (labelled combined) shows an improvement in terms of the robustness of the ten runs performed after applying the adaptations compared to the baseline. See, for example, MH05 in Figure 3.13b, for which the scene is characterised as difficult with fast motion and dark scene, and LR3 (Figure 3.12b). Improvements were also made on DSO running on LR2 (Figure 3.12a) and MH02 (Figure 3.13a). Skipping redundant frames enables the algorithms to have useful frames within their tracking window, resulting in improved robustness. While adapting DVFS reduces power with minimal impact on the accuracy achieved when combined with frame skipping, leading to improvements in robustness and power reduction compared to the base.

**Marginal Impact on Accuracy and Robustness:** The sequences, in this
3.5. PERFORMANCE ON ENVIRONMENTS WITH REALISTIC MOTION

Figure 3.13: Violin plots of benchmarking the framework on the i7 and using EuRoC MAV Machine Hall sequences.

case, have similar accuracy to the base, with marginal variability in the RMSE values of the 10 runs (i.e. there is high robustness). This is the case for LR1, LR2 and MH03 on DSO, and LR1, LR0, MH01 and MH02 on ORBSLAM2. The combination of both parameters results in a power reduction of between 65-75% over the base power consumption of ORBSLAM2, and between 39-60% with DSO. The reason for such improvement, while producing a minimal impact on the accuracy, is that the change in camera motion corresponds to redundancy in the observable tracking information. This is where the presented adaptation meets its goal and ensures that skipping redundant frames and lowering the CPU frequency have minimal impact on the accuracy and robustness but provide significant improvements in power consumption.
**Significant Accuracy and Robustness Impact:** Looking at Figures 3.12 and 3.13 the most impacted sequences, in terms of robustness after applying the adaptations to both algorithms, are the ones characterised to have a medium and difficult scene and motion properties. This the case for LR0, LR3 and MH03–05 on DSO, and LR2, LR0, MH03, and MH04 on ORBSLAM2. Upon inspecting the impacted trajectories, it seems that the impact happens at particular regions during the tracking, which we refer to as critical points. The inability of the used metric ($P'$) to identify these critical tracking points when they are encountered, i.e. when the metric has low values at these points, leads to producing inappropriate adaptation values which impact the accuracy. Further, a possible reason is due to the reduced responsiveness of the adaptation model due to the use of the running geometric window introduced to reject outliers on the metric $P'$.

### 3.6 Summary

The results from the previous section show that it is possible to reduce power consumption in a real-time setting and with minimal or no impact on the accuracy and robustness in simulated (synthetic) environments. Loop perforation-based frame skipping results in a significant impact on accuracy and robustness at similar power consumption, while predefined DVFS governors do not deliver the desired balance between power reduction and accuracy. This implies that adaptations guided by the distance-based characterisation metric are essential to obtain the desired balance between power reduction and accuracy. However, when the basic adaptation model is deployed on real datasets, it doesn’t perform well in terms of accuracy and robustness across the range of sequences and VS-LAM formulations, meaning that it is not portable. This motivates the continued search for better metrics that are capable of identifying specific types of motions or scene complexities that give rise to critical points in maintaining tracking. In addition, it is possible that further improvements to the adaptation model can be achieved by increasing its responsiveness to changes. These two aspects are tackled in the following two chapters.
Chapter 4

SLAM-Dunk – The Runtime Framework

In this chapter, we formalise the adaptation framework which is named SLAM-Dunk and contribute a novel lightweight adaptation policy which normalises characterisation metrics. This policy has its basis introduced in the previous chapter. Further, we introduce a novel efficient scene metric based on the earth mover’s distance to quantify the changes in the scene.

This chapter first presents a general overview of the SLAM-dunk pipeline in Section 4.1. Then we motivate the need for decoupling the motion characterisation metrics and presents the pyramid-based EMD in Section 4.2. Section 4.3 presents the adaptation policy which aims to normalise the different metrics used to characterise the tracking difficulty. In Section 4.4, we describe how the different characterisation metrics can be combined to improve the estimation reliability. Finally, we describe the configurations of the two general parameters, DVFS and frame skipping in Section 4.5.

4.1 SLAM-Dunk Pipeline

An overview of our runtime adaptation framework is shown in Figure 4.1. The aim is to improve the efficiency of various VSLAM systems in a portable fashion. The framework exploits both the rate of change in the estimated sensor motion which can be translational ($T$) or rotational ($R$) and changes in the observed scene (EMD) and uses these to estimate the current tracking difficulty. The idea is that performance metrics can be improved, or traded, by taking advantage of
Figure 4.1: An abstracted overview of the SLAM-Dunk framework which relies on the rate of change in the scene, quantified with a pyramid-based Earth Mover’s Distance (EMD), and the estimated sensor motions to adapt actuators on both VSLAM and the hosting platform.

... slow-motion or redundant scene information, where sufficient tracking accuracy can be maintained with reduced computational cost, resulting in minimal impact on the tracking accuracy [82, 88, 23]. The estimates of tracking difficulty can be used to make changes to parameters which directly affect, or tune, performance at runtime, as the perceived tracking difficulty varies. The parameters that are changed can be related to both the VSLAM formulation - for example, for identifying and skipping frames with low new information content - and the hosting platform, by, for example, changing the operating frequency of the platform (using DVFS).

The framework pipeline consists of three main stages: first, the tracking difficulty is estimated based on the monitoring of three different metrics. These metrics are then adapted based on a predefined policy to cope with the (unknown) metric variations encountered in the environment in which the sensor is deployed. Finally, the resulting adapted metric values are used to control the tuning parameters associated with either the VSLAM itself or the hosting platform.

The framework is modular in the sense that it decouples the tracking difficulty...
metrics used from the adaptation process, meaning that different characterisation metrics can be used with the adaptation policy and different adaptation policies can be developed to use existing metrics. The same applies to the type of parameters chosen to be tuned. Next, the requirements for each of the stages of the framework is reviewed and discussed in the light of related work.

4.2 Estimating Tracking Difficulty

The ability to quantify the visual scene and the sensor motion patterns with a small number of metrics is beneficial for several reasons, such as: characterising visual datasets based on their difficulty and diversity; for VSLAM failure diagnosis or for improving the runtime performance of VSLAM [82, 102, 60]. The latter requires metrics with low overhead so as not to detrimentally affect the original, base, performance of the VSLAM or when VSLAM is integrated with algorithms for scene understanding or semantic labelling. For example, the change in sensor motion (mainly translational), estimated by VSLAM, has been used as a characterisation metric in [88, 23] to improve various aspects of VSLAM runtime performance, due to its good correlation with the tracking error and efficient computation. However, relying purely on a single aspect of the change in sensor motion as a characterisation metric is limited since the sensor experiences a combination of different motion patterns with each causing different VSLAM algorithms to be impacted in different ways, as we demonstrate in this work.

Further, relying purely on sensor motion without taking into account how the scene changes is also limited. For example, if there are dynamic objects encountered or sudden changes in brightness, either locally or globally, in the tracked frames. Such changes can affect VSLAM tracking accuracy, especially on those based on direct formulations [31, 43].

To enhance the characterisation accuracy and reliability, we incorporate multiple types of information about the change in VSLAM sensor motion and the observed scene. First, we treat the translational and rotational motion as separate metrics. This was motivated by the results obtained in Section 3.5 which shows that there an impact on accuracy and robustness on the challenging sequences. This is due to the fact that the used normalised metric, which combines both rotation and translation, \( P' \), is affected by the magnitude of variations from either of the sources, i.e. each source varies on a different scale. Figure 4.2 shows
Figure 4.2: A highlight of where the metric $P'$ is unable to capture the difference in rotation ($R'$) due to being on a different scale with relative to the translation difference ($T'$).

an example case obtained using the ground truth data from ICL-NUIM LR2. The figure has three subplots, the normalised metric $P'$ along with the decoupled normalised metrics for translation ($T'$) and rotation ($R'$) respectively. As can be seen from the figure, the metric $P'$ is highly influenced by the sensor translational motion as opposed to the rotational motion, where the two peaks on $R'$ are barely registered on $P'$. This is because the scale of rotation is much smaller than the translation on this sequence. Thus to ensure the changes from either motion sources is captured, we separate each source to a single metric. This decoupling can also be performed on the VSLAM level as shown in [112]. Further, even though fast rotation is typically the main source of tracking degradation or failure [113, 114], due to the lack of correspondence of features selected by the algorithm for tracking between the frames, extreme translational motion can also lead to a similar impact on some VSLAM algorithms, as we demonstrate in our results.

We utilise the change in pose estimated by the VSLAM system to measure the two metrics (translational and rotational) at no additional computational cost. The estimated pose can be used alone or in conjunction with information from an Inertial Measurement Unit (IMU), if one is available (or if the use of an IMU is not supported by the VSLAM formulation) for extra reliability. In this work, however, we rely only on the pose change estimated by the formulation, since it
4.2. ESTIMATING TRACKING DIFFICULTY

applies to a wide range of formulations and thus supports our portability goal.

4.2.1 Computing Motion Metrics

Let $t$ be the time when a new camera frame is acquired after initialising the system but has not yet been passed to the VSLAM pipeline. The change in the camera pose $\Delta P$ can be measured directly from the estimated pair of camera poses computed at $t-1$ and $t-2$, available to be used at time $t$, denoted by $P_{t-1}$ and $P_{t-2}$. The change in translation, $\Delta T$, is defined as the Euclidean distance between the sensors 3D position $[x, y, z]$ at $t-1$ and $t-2$. The use of quaternions to quantify the rotation is efficient both computationally and spatially [115], thus it is used to measure the change caused by the rotation $\Delta R$ is defined as $\Delta R = |\Delta Q| |\Delta Q| ^T$, where $|\Delta Q|$ is defined as the difference between the element-wise absolute value of the quaternions vector $[q_x, q_y, q_z, q_w]$ at $t-1$ and $t-2$. This metric highly correlates with rotational motion always resulting in a positive number with rotation and zero otherwise.

4.2.2 Computing The Pyramid-based EMD

We also use the change in the scene as an additional metric for extra reliability in characterising the tracking difficulty. We treat the change in the scene as an image similarity problem. As discussed in Section 2.3.5.1, we use a direct approach based on the first order of Wasserstein distance, also known as the Earth Mover’s Distance (EMD) [24]. The Earth Mover’s Distance is a metric that quantifies the similarity between two distributions, as proposed by [24]. The metric is based on finding the least amount of effort needed to transform one distribution to the other. For example, in the context of image retrieval, measuring EMD between two identical images will result in a distance equal to zero and a positive number otherwise.

In the context of VSLAM, lower EMD values between subsequent images mean a similar scene, which usually implies higher redundancy in the tracked features in the scene. Such a scene can be skipped during tracking to save power, for example, with minimal impact on accuracy. EMD is computationally expensive with exponential complexity in the worst case. One way to address this issue is to transform the coloured images to be compared into a lower dimension representation, for example, HSV [24] or a grey-scale, to reduce the computation cost.
In the context of characterising the scene for VSLAM, it is desirable to preserve the spatial information of pixels and their respective values, making a grey-scale representation more suitable.

However, even with the use of grey-scale images, direct matching between images is still computationally expensive and currently infeasible to achieve under real-time constraints. To tackle this, we down-scale each grey-scale frame using Gaussian pyramids to the level \( n \), a user-specified parameter. This level is chosen empirically so that the average computations of EMD between a pair of pyramids, resulting from the most recent pair of frames \( I_t \) and \( I_{t-1} \), has a negligible overhead relative to the VSLAM algorithm itself. The use of a high-level pyramid approximates the frame but still preserves the perceptual appearance at a coarser level. We have found this sufficient for measuring the scene similarity in our case. We treat both pyramids as distributions, where each is then normalised to have a total ‘mass’ of one, as depicted in Figure 4.1. The reason for having the same total mass is to ensure a true EMD distance, as discussed in [24]. The normalised distributions are then converted into a signature, \( S_t \) and \( S_{t-1} \). Each signature is an array-like data structure of size \( 3 \times c \), where the columns \( c \) is equal to the total number of pixel in the pyramid, and the three rows containing the normalised pixel value and its (x,y) coordinates in the pyramid, respectively. The distance between the two signatures, which accounts for both intensity and spatial information in the frame, is measured using the OpenCV implementation of EMD [116] based on [24]. The distance will be zero if both frames are identical and a positive number otherwise.

This direct approach of measuring EMD between the \( n^{th} \) pyramid of each pair of frames has several advantages besides reducing computation time. First, it quantifies the change in the scene resulting from the sensor motion (rotational, \( \Delta R \) or translational, \( \Delta T \)), enhancing the reliability or from the effects of dynamic objects in the scene. Second, it captures global (i.e. those affecting the whole frame) or local changes in brightness which may degrade the accuracy of tracking, particularly on direct VSLAM formulations [31][43]. Finally, the change in the scene can be quantified before the frame \( I_t \) is passed to the VSLAM pipeline, allowing for the appropriate action (for example skipping the frame) to take place in advance.
4.3 Adaptation Policy – Normalising Measured Metrics

The changes in both sensor motion and scene are usually unknown in advance and largely dependant on the environment in which the sensor is deployed. For example, a sensor mounted on quad-captor typically experiences high and non-linear motion compared to a ground robot moving on a smooth surface. This reinforces the requirement to use a lightweight mechanism for adapting to these unknown and non-smooth variations without relying on ground truth data (which will not exist for a new deployment) or on a large number of predefined, user-specified, thresholds, as is currently common. Key advantages of having a minimal design for the adaptation mechanism are low computational cost, ease of maintenance and the increased portability across different VSLAM algorithms, deployment environments and platforms.

For the adaptation stage, we, therefore, present a portable policy capable of adapting to large or sudden changes in both sensor motion and the observed scene without assuming any knowledge about the range of variation expected to be encountered in the environment. The policy, in its essence, is a metric normalisation algorithm which is an improved version of the basic adaptation model described in Chapter 3. It takes each specific characterisation metric at time $t$, designated by $m_t$, as an input. It then produces a normalised real value (between zero and one) to a running window average, $\alpha$, of the most recent high values of that metric observed during the tracking up to the current time. For example, when the camera sensor translation accelerates to previously unseen levels, the adaptation policy will output a value of 1; an indication that the VSLAM algorithm and/or the hosting platform needs to be at their full capability as maintaining tracking accuracy is likely to be difficult. If the VSLAM algorithm can handle this new level of change, then the new level will be used for parameter tuning. The pseudocode for the policy is presented in Algorithm 1:

- The goal of this algorithm is to adapt the measured metric value $m_t$ at time $t$ to an appropriately normalised value, $m'_t$, based on the stored peak values in $W$ of previous observed $m_j$, where $j < t$, for a time window of maximum size $z$.

- Line 1: The initialisation is done once at $t = 0$. The window, $W$, has a
Algorithm 1 Metric Normalisation Algorithm

1: Initialise:
   \( W \leftarrow [], \alpha \leftarrow -1, z \leftarrow \text{win\_size}, c \leftarrow 0 \)
2: Input:
   \( m_t \)
3: Output:
   \( m'_t \)
4: if \( m_t > \alpha \) then
5:   \( m'_t \leftarrow 1 \)
6:   Enqueue\((W,m_t)\),
7:   \( \alpha \leftarrow \text{mean}(W) \)
8: if \( c < z \) then
9:   \( c \leftarrow c + 1 \)
10: else
11:   \( c \leftarrow 0 \)
12:   \( W \leftarrow [] \)
13: end if
14: else
15:   \( m'_t \leftarrow m_t / \alpha \)
16: end if
17: return \( m'_t \)

fixed size used for storing the most recent maximum observed values (new peaks) of the given metric \( m_t \). The value \( \alpha \) is the mean of the values in window \( W \), and it is initialised to a minimal value to prevent division by zero in Line 15. The size of the window \( W \), \( z \), determines how conservative is the adaptation to new unseen levels, where the larger the window, the more conservative is the adaptation.

- Lines 4-15: This part of the algorithm is responsible for performing the normalisation to unseen levels of change in the metric \( m_t \). These levels depend on the metric and how it varies in the deployed environment and are usually unknown in advance. If the change, measured using the input metric \( m_t \), is greater than the initial value of \( \alpha \) (i.e. a new peak has been encountered), \( m_t \) is added to the window \( W \), if \( W \) is not yet at full capacity (\( z \)) (Lines 8-9). If the window \( W \) is full, the new average value is assigned to \( \alpha \), and the window is emptied (Lines 6-7 and 11-12). In either case, the new high value of \( m_t \) compared with \( \alpha \) produces a value of 1 (resulting from Line 5) indicating that the change is ascending to unseen level and, thus,
4.4 METRICS COMBINATION

the tracking is likely to be difficult.

- Line 15: The current metric value $m_t$ is then normalised to the averaged peak values, $\alpha$, resulting in the adapted metric value $m'_t$ (line 17).

Applying the running average window $W$ to the peaks values of the distance metric instead of the current value of the normalised metric as was the case in Section 3.5, has the advantage of improving the responsiveness to the change in the metric. The size of the window $W$ is set to one (no averaging) by default, however, it is sometimes useful to calculate the mean over a larger window, especially if there are known, or anticipated, outliers in a metric values, which is sometimes the case in VSLAM algorithms with non-deterministic behaviour, as shown in Section 3.5. The geometrical mean can be more useful in such case compared to the arithmetic mean, and it is employed throughout our experiments. The window exists only because the use of the pose estimated by such algorithms, in the case where there is a reliable IMU sensor or optical-flow based method to be used for quantifying the change in motion it is not needed.

This normalisation policy produces a value between zero and one for each metric which is used to tune the selected control parameters of the algorithm and platform in the next stage of the pipeline. We evaluate each metric individually, where $T'$, $R'$ and EMD' denote the normalised metrics based on pure translation, pure rotation and the pyramid-based Earth Mover’s Distance, respectively. Each normalised metric has a real number value ranging from 0 to 1.

4.4 Metrics Combination

We also evaluate the use of a combination of the three metrics, $\sigma_t$, computed as $\sigma_t = T' + R' + EMD'$ and $\sigma_t = 1$ if the summation of the three normalised metrics is greater than one (the metrics are currently not weighted, future work will explore the use of different weights). We found that $\sigma_t$ gives improved results over the use of any individual metric. This is expected because the difficulty increases if large change encountered from all of the measured metrics accumulate.

We use these metrics for online parameter tuning, where lower values imply easier tracking due to some redundancy in the observed scene. This is under the assumption that the scene contains adequate and sufficient tracking information, of course, and we will compare our results with the performance of the
unmodified VSLAM formulations that we use, with no tuning, on each of our selected platforms. This will give us a set of base cases for each formulation and platform, which will also show whether the scenes we use have adequate tracking information for the base VSLAM formulations on the platforms.

4.5 Parameter Tuning and Configurations

This stage is responsible for the parameter tuning decisions that are made at runtime for the parameters (actuators) available in the specific VSLAM and execution platform. The tuning decisions are based on the change, $D_t$, in a metric. The change may be computed based on a single metric ($T'$, $R'$ or EMD'), or, in general, using a combined metric $\sigma_t$, thus, $D_t = \sigma_t$. The change is computed after the VSLAM system is initialised and after each frame, $I_t$, is captured.

The adjustment of a parameter is essentially instantaneous and is performed before the frame $I_t$ is passed to the VSLAM pipeline. As discussed, the tuning parameters are usually specific to each VSLAM implementation and execution platform; Hereafter, we use two parameters to demonstrate that the SLAM-Dunk framework is effective and portable across VSLAM implementations and platforms. The parameters are adjusted using linear functions, a slightly modified version of those described in Section 3.2 for positive and negative correlations, to reduce power consumption while maintaining accuracy. With frame skipping, SLAM-Dunk follows an approach making the tuned parameter converges towards the minimum. On the other hand, for DVFS, SLAM-Dunk uses an approach that makes the parameter converges towards the maximum DVFS. This is in order to minimise the impact on accuracy or robustness as much as possible. This subsequently, makes the framework conservative in terms of improving power consumption at the expense of accuracy or robustness.

Although the main focus is on the two general parameters DVFS and frame skipping, we evaluate a case where this framework is used to tune an application-specific parameter. We use ORBSLAM2 maximum amount of traceable features as such parameter. The reason behind the choice is three folds, first to compare the framework presented in this thesis to the one that relies on PID control in [2]. Second to evaluate an ORBSLAM2 specific parameter since it has non-deterministic behaviour. And finally, to evaluate the possibility of simultaneously adapting both general and application-specific parameters.
4.5. PARAMETER TUNING AND CONFIGURATIONS

4.5.1 Dynamic Voltage and Frequency Scaling (DVFS)

Power savings from DVFS can result from running VSLAM computations at lower frequency levels. Such savings should take advantage of periods with small changes in the selected characterisation metric, as to not violate the quality of service (QoS) requirement which for VSLAM is a real-time constraint in terms of frames per second. For example, with a keyframe-based VSLAM, if the normalised characterisation metric has a value near zero, which implies that the observed scene has a large amount of redundant information, the per-frame computation time is usually short. In this case, running at a low frequency can save power without violating the QoS, as long as the base (non-tuned) VSLAM already satisfies the QoS when operating at its full potential.

Let $X_{\text{max}}$ and $X_{\text{min}}$ be the maximum and minimum frequencies available as a parameter to SLAM-Dunk. The frequency value at time $t$ is calculated as

$$X_t = \text{round}(X_{\text{min},t} + (X_{\text{max}} - X_{\text{min},t})D_t),$$

(4.1)

where $\text{round}()$ is a function that rounds the resulting real number to the nearest available scaling frequency step of the processing unit being targeted, $X_{\text{min},t}$ is the current recorded minimum frequency while still delivering the QoS, and $D_t$ is the change in the normalised metric calculated as described in Section 4.3.

To maintain the QoS, we adjust the minimum frequency $X_{\text{min},t}$ to a level which aims to avoid dropping frames which could be valuable to the tracking accuracy. This is achieved by increasing the minimum frequency to the next highest available frequency while the per-frame compute time is close to (currently within 5 ms of) the dataset’s specified QoS frame time deadline (e.g. 50 ms for a 20 FPS QoS). The minimum frequency may eventually meet the maximum frequency if the VSLAM algorithm is unable to maintain the QoS at any particular region during the tracking. If the per-frame time is not close to the frame time, then $X_{\text{min},t}$ is decreased to the next lowest available hardware frequency step. If the per-frame time exceeds the frame deadline, then the next frame will be skipped as usual.
4.5.2 Redundant Frame Skipping

Frame skipping aims to reduce the amount of computation, and consequently power consumption, by dropping frames where the scene information is highly redundant. For example, when there is a large overlap between subsequent frames in the scene. Such ‘redundant’ scene information can be the result of a slow-moving sensor. Skipping the redundant frames usually has minimal impact on the trajectory accuracy, since traceable information is present across multiple frames within the tracking window. We skip frames up to a (tuned) maximum batch size before a new frame is passed to the VSLAM pipeline.

The batch size of frames to be skipped at time $t$ is determined by

$$Y_t = \text{round} (Y_{\text{max}} - (Y_{\text{max}} - Y_{\text{min}})D_t),$$  \hspace{1cm} (4.2)

where $Y_{\text{max}}$ is the maximum number of frames that can be skipped in succession and $Y_{\text{min}} = 0$ in all of our experiments. The larger the change in $D_t$, i.e. when it has a value near one, the fewer frames that are skipped. When the change in $D_t$ is small, more frames will be skipped.

Should a VSLAM implementation not be able to handle alternating processing a frame and skipping the next (i.e. the smallest batch size) because accuracy is being compromised, due to reasons unrelated to how the scene or motion changes, but related to the scene being deprived of adequate or sufficient tracking information specific to that VSLAM implementation, it is desirable to seek to continue to skip a smaller number of frames only, in order to maintain accuracy and still make some power savings. To achieve this, it is possible to introduce a condition based on the $D_t$ value. As an example, a condition could be to skip only a one frame batch if $D_t$ is very low, e.g. $< 0.25$. During our first set of experiments, we found that this case is present in the combination (DSO-LR0). In Section 5.2, we use this same portable configuration in our experiments to evaluate the possibility of achieving full portability, where this condition is applied to DSO and ORBSLAM2 on both platforms when using all the dataset sequences.

We investigate the impact of using the two general parameters on the trajectory accuracy of frame skipping on both sparse (direct and indirect) and dense VSLAM implementations. Also, for the dense implementation, Elastic Fusion, we evaluate the effects on the dense 3D map construction quality.
4.6 Summary

In this chapter, we have presented SLAM-dunk, a runtime adaptation framework to improve the performance objectives of VSLAM. In Section 4.1, a general overview of the framework pipeline, consisting of three stages related to estimating the tracking difficulty, is described in Section 4.2 then the adaptation policy is presented in Section 4.3, and a metrics combination for extra reliability is described in Section 4.4. Finally, parameter configurations and tuning are described in Section 4.5.

In the next two chapters, we explore and evaluate different portability scenarios which are achievable using the SLAM-Dunk framework.
Chapter 5

Portable Performance on Sparse VSLAM

In this chapter, we evaluate the performance of SLAM-dunk in two main scenarios. In Section 5.1, the experimental setup of both scenarios is described. The goal in the first scenario is to achieve portable performance on two sparse VSLAM algorithms under the same general settings of SLAM-dunk and parameter ranges. Section 5.2 presents the first scenario, and evaluates the performance when using the characterisation metrics in terms of portability on the two sparse algorithm datasets and platforms. In Section 5.3 the second scenario is presented, where the framework is further explored when an application-specific parameter is tuned along with the general parameters to improve the performance requirements of ORBSLAM2 using two extra datasets.

5.1 Experimental Setup

The setup for the first scenario is similar to the one described in Section 3.3. However, for the second scenario, we only evaluate the results on the Jetson since it is a low-end robotics platform. We use two additional datasets, the RGB-D TUM dataset [58] and KITTI stereo dataset [12]. The evaluation is performed on ORBSLAM2 given its support for both RGB-D and stereo sensors. Memory usage statistics are collected by reading the VmRSS (Virtual memory Resident Set Size) value from the VSLAM process status.
5.2 VSLAM Metrics Evaluation and Portability

We evaluate the performance of the framework in terms of the percentage power reduction achieved and the accuracy and robustness of the estimated trajectory produced. Robustness means the consistency in the trajectory produced over ten runs of the same case. This is similar to what has been done in Section 3.4.1 and 3.4.2, where we compared the performance of the framework with existing techniques (e.g. comparing the performance of adjusting DVFS using the framework with two existing DVFS governors, on-demand and power save).

We evaluate a number of aspects of SLAM-Dunk. We consider the achieved accuracy using each of the characterisation metrics on their own and also in combination. In addition, we consider the portability of each metric (i.e. how well it performs), and their combinations, across the datasets and platforms used for the experiments.

To facilitate the comparisons, we use the same settings across the VSLAMs, datasets, and platforms. That is, we use the same update window $W$, the same frequency range on each platform, as well as the same maximum batch size for skipped frames. Although this is the case for the framework settings, we acknowledge the fact that the two platforms have different micro-architectures with different capabilities; our aim is to observe how the framework behaves under these various scenarios.

Figure 5.1 and 5.2 present the benchmarking results for the framework guided by the different characterisation metrics on the i7, while Figure 5.6 and 5.7 are for the Jetson. For each platform, VSLAM and sequence combination power reduction relative to the base case execution of each algorithm (without SLAM-Dunk) is shown in the top subplot, while trajectory accuracy (RMSE) is shown in the bottom subplot.

The framework is evaluated using the fixed frame rate versions of DSO and ORBSLAM2 running with the four ICL-NUIM living room sequences (LR0–3) and the five machine hall sequences (MH01–05) from the EuRoC MAV drone dataset which have different difficulty levels, as shown in Table 3.2. The ORBSLAM2 base case fails to initialise on LR1, thus this sequence is removed from the ORBSLAM2 plots.

Each violin plot in the figures represents the distribution of measurements over ten runs of a case, and the results are colour-coded based on the adaptation
metric, or the combination, employed, as indicated in the x-axis.

We analyse the results from multiple perspectives in the following sections.

### 5.2.1 Characterisation Accuracy of the Metrics

A key advantage of the framework is its modularity; that is, it decouples the metric used from the adaptation policy and the parameter tuning actions. This enables an individual metric to be evaluated in terms of its ability to characterise tracking difficulty. To illustrate this, we discuss the results on the i7 platform for the DSO and ORBSLAM2 formulations which are shown in Figure 5.1 and 5.2. The use of the same platform allows us to rule out any impact related to the processors compute capability.

![Figure 5.1: Violin plots of benchmarking the framework on the i7 and using EuRoC MAV Machine Hall sequences.](image-url)
5.2. VSLAM METRICS EVALUATION AND PORTABILITY

Figure 5.2: Violin plots of benchmarking the framework on the i7 and using ICL-NUIM living room sequences.

Looking at the base case for the two formulations (labelled *Base* in the figures), ORBSLAM2 is seen from the violin plots to be more accurate and robust than DSO. This is to be expected since ORBSLAM2 is a full VSLAM with loop closure whereas DSO is pure visual odometry.

In terms of metric accuracy, the adaptation resulting from the pyramid-based EMD (EMD’) performs, on average, better than the adaptations based on rotation (R’) and translation (T’) alone, both in terms of accuracy and robustness. This is because EMD’ implicitly captures most of the change in the sensor motion (both rotational and/or translational change) and frame-to-frame changes in lighting. However, for some sequences, notably MH03 and LR0 in Figure 5.1a and 5.2a respectively, EMD’ does not perform as robustly as the *Base*. This is because the metric reports lower values when the scene has very smooth and similar
surfaces (which is the case for LR0) or when there is a fast, forward translational motion, where the centre of the frame remains highly similar (which is the case for MH03). These effects could be minimised by measuring EMD on finer/lower pyramid levels at the expense of increased computation but instead, we use the metric in conjunction with the two motion metrics estimated by the formulation, \( R' \) and \( T' \), in order to gain the extra reliability seen in the plots, when using \( \sigma \).

The figures also show that using the metric \( R' \) results in more accurate and robust tracking compared to that of \( T' \), in most cases. \( R' \) also achieves better power reduction compared to the other metrics. This is because rotation is not the dominant motion for the duration of most of the sequences, and this leads to a higher power reduction. However, in the presence of fast translational motion, when adapting based solely on rotation, the trajectory accuracy and robustness is impacted; this effect can be seen clearly to be prominent for DSO in both the MH04 and MH05 scenes in Figure 5.1a. To try to understand this impact, we take MH04 as an example and examine the resulting ten trajectories, shown in Figure 5.3, when adapting based on \( R' \) alone; Figure 5.4 depicts a per-frame profile capturing multiple aspects of the framework behaviour on this sequence.

Looking at the trajectories in Figure 5.3, which are scaled and aligned with the ground truth trajectory (dashed line), the tracking accuracy can be seen to be degraded in the two regions marked with transparent Red. These two regions roughly correspond to frame intervals [900,980] and [1460,1550] in Figure 5.4. The profile shows, from the top, the normalised metrics (\( T' \), \( R' \) and \( \text{EMD}' \)) which are the product of the adaptation policy as described in Section 4.3, along with the combination metric, \( \sigma \).

The values of the two tuned parameters, DVFS and Skipped Frames, shown in the bottom two subplots of the figure are generated based on the use of the combination metric, \( \sigma \), but it is possible to infer how these two parameters will behave when using any of the three metrics. The areas highlighted in Grey indicate where each of the three metrics encounters new, previously unseen peaks the occurrence of which corresponds to the algorithm behaviour contained in lines 4 to 14 of Algorithm 1.

As can be seen from the profile, translational motion is most dominant in the two previously identified intervals; this is clearly captured with both \( T' \) and \( \text{EMD}' \) metrics but is also seen to be slightly registered with the \( R' \) metric. This behaviour in the \( R' \) metric is what leads to both skipping frames and reduction
of the frequency in what is a critical location, and this causes the degradation in the tracking accuracy.

This example supports the observation that difficult tracking regions can arise from any type of motion, such as the fast translation motion in this case, or from a rapid scene change. Adapting based any one of the three metrics individually is thus seen to be not portable across the different types of motion present in each sequence, let alone across different environments and algorithms. However, as can be seen in Figure 5.1 and 5.2, by combining the three metrics into the metric \( \sigma \) we find that we can deal with these challenges due to the increased reliability of detecting the changes, and, thus, the overall accuracy of the characterisation increases, resulting in robustness and accuracy on a par with the Base performance.

Figure 5.3: Resulting trajectories of MH04 from rotation-based adaptations which are applied to DSO running on the i7. Two critical regions are marked with transparent Red.
Figure 5.4: Per-frame profile of (DSO-i7-MH04) showing the behaviour of the adapted metrics along with their combination ($\sigma$). The highlighted regions indicate where the individual metrics adapt to unseen change in the environment which is prominent at the start of the sequence. The two parameters (DVFS) and (frame-skipping) are based on the combination ($\sigma$) of both DSO and ORBSLAM2.

In Figure 5.1 and 5.2, we see power reductions when using $\sigma$, ranging between 43% to 69% on the two different formulations while meeting the fixed frame rate constraint. We argue that the improvement in the robustness when using the $\sigma$ metric is mainly due to the skipping of redundant frames which enables both algorithms to have useful frames within their tracking window, supporting their keyframe selection criterion, in a portable fashion.

### 5.2.2 Performance on Different Datasets

The data in Figures 5.1 and 5.2, shows that for each algorithm power reduction is very similar across sequences from the two different datasets used (ICL-NUIM and EuRoC MAV) even though the datasets run at different frame rates (30 FPS and 20 FPS for ICL-NUIM and EuRoC MAV, respectively). This is to be expected since the same framework settings are used in both cases and the fixed frame rate is not violated on either dataset (i.e. non of the frames are skipped...
for violating the real-time QoS requirement). However, the large impact on the accuracy and robustness is most visible when running the difficult sequences: MH03-05 for DSO and MH04-05 for ORBSLAM2. The drone in these sequences experiences local, fast non-linear motion or dark scenes. In either of these cases, the combination metric $\sigma$ performs similar to or better than the Base. For example, when using the combination $\sigma$ as the metric for DSO running on the i7 and operating on MH03, the accuracy has improved relative to DSO Base case, as can be seen in Figure 5.1a. Also, from Figure 5.1b, it can be seen that the robustness has improved on MH04 relative to the ORBSLAM2 Base. The corresponding trajectories of these two examples, aligned and scaled to the ground truth (dashed line), are shown in Figure 5.5. This figure shows a side-by-side view comparing the Base trajectories and the trajectories resulting from adapting based on the $\sigma$ metric. This shows that with accurate characterisation, the framework is capable of coping and operating in different deployment environments.

5.2.3 Performance on the Two Platforms

Both the DSO and ORBSLAM2 algorithms have been developed and evaluated on high-end machines [35, 1], thus running on a low-end platform using the same (default) settings for the framework and the cross-compiled version of each algorithm with hard deadline constraint would be expected to result in only small power savings. The results in Figures 5.6 and 5.7 show that this is indeed the case when running on the low-end Jetson platform.

Looking at these figures, it can be seen that the combination $\sigma$ metric performs similar to the Base version of each algorithm in terms of both accuracy and robustness, resulting in power reductions of up to 40%.

On ICL-NUIM sequence LR0-3, Figure 5.7b, power reduction is small when running ORBSLAM2, especially so when using EMD$'$ and $\sigma$-based adaptation; this is because they are both more conservative with regard to skipping frames compared to adaptation using either $T'$ and $R'$. The framework aims to maintain a QoS of 30 FPS and to avoid forced frame skipping, as discussed in Section 4.5.1, which could lead to the accuracy and robustness being compromised.

Overall, the results from both platforms (i7 and Jetson) show that SLAM-Dunk can improve power consumption with minimal impact on the accuracy or robustness when using the same algorithm settings. Further, these savings are portable across different algorithms and datasets.
Figure 5.5: Side by side views of trajectories resulting from the Base and $\sigma$-based adaptations.

5.3 Tuning Application-Specific Parameters

In this section, we explore tuning an application-specific parameter, namely the Maximum number of Feature (MF) allowed to be used for tracking in ORBSLAM2
5.3. TUNING APPLICATION-SPECIFIC PARAMETERS

along with the two portable parameters DVFS and frame skipping. Our goal is to explore its performance using the $\sigma$ characterisation metric on extra two datasets with different motion patterns, sensors and scale. We further, compare the performance of SLAM-dunk against the recent framework proposed in [2].

5.3.1 Performance on RGB-D TUM

In this section, we explore the performance of SLAM-dunk on the RGB-D TUM dataset [58] using the RGB-D sensor and operating in the linearised mode, described in Section 2.2.2, with three objectives in mind. The first is to compare performance of SLAM-dunk with the framework proposed in [2], using the same range for the ORBSLAM2 MF parameter (MF range = \{MF | 510 \leq MF \leq 1000\})

Figure 5.6: Violin plots of benchmarking the framework on the i7 and using EuRoC MAV Machine Hall sequences.
and the same sequences evaluated from the RGB-D TUM dataset. The second objective is to combine and adapt the application-specific parameter (MF) with the two general parameters DVFS and frame skipping, to show that further improvements can be made with SLAM-dunk less impact on the accuracy compared to [2]. Finally, the improvements made on memory usage is evaluated.

We perform a finer tuning of the parameters using the rounded values resulting from equation 3.2, instead of the coarse steps used in [2]. We compare our results with the results obtained using the Jetson TX2, shown in Table A.1 in the Appendix A. Since we use a newer version of the Jetson (the AGX Xavier), we report the improvement or impact relative to the base algorithm (where the MF value is fixed to 1000 features) on the used AGX Xavier for our results. Unlike [2], we account for any impact on the robustness by performing ten consecutive runs.
The results are shown as violin plots in Figure A.3 and condensed by reporting the mean values in Table A.2. A summary of our results and those reported in [2], is shown in Table 5.1, where we report the geometric mean values of per-frame time and energy, and the translational RMSE, similar to [2].

<table>
<thead>
<tr>
<th></th>
<th>Result from [2]</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Geometric Mean</td>
<td>Impact (+/-)</td>
</tr>
<tr>
<td>MF1000 (Base)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time (s)</td>
<td>0.257</td>
<td>—</td>
</tr>
<tr>
<td>Energy (J)</td>
<td>1099.598</td>
<td>—</td>
</tr>
<tr>
<td>RMSE (m)</td>
<td>0.025</td>
<td>—</td>
</tr>
<tr>
<td>MF510-1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time (s)</td>
<td>0.191</td>
<td>25.69%</td>
</tr>
<tr>
<td>Energy (J)</td>
<td>836.655</td>
<td>23.91%</td>
</tr>
<tr>
<td>RMSE (m)</td>
<td>0.027</td>
<td>-7.16%</td>
</tr>
<tr>
<td>Combined</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time (s)</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Energy (J)</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>RMSE (m)</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 5.1: Results summary on RGB-D TUM.

The results show that for the same adaptation range of MF (MF510-1000), SLAM-dunk is more conservative in terms of time and energy savings, achieving about 18.33% and 19.63% respectively compared to 25.69% and 23.91%. However, in terms of the overall accuracy it achieves a slight improvement of 2.88% compared to an impact of -7.16%. This implies that SLAM-dunk favours less impact on the accuracy compared to [2]. When all three parameters are combined, SLAM-dunk achieves about 41.67% improvement on energy and still has less impact on the overall accuracy. Sample trajectories of the ten runs on different sequences are presented in Figure A.1 in Appendix A.

In terms of memory usage, the ORBSLAM2 factor graph grows over time as it incorporates observations and trajectory poses. This incremental growth consumes the main memory and eventually leads to a degradation in the overall performance as the memory becomes full. Figure 5.8 shows an example case on the sequence Fr3-long where the adaptations can be used to slow down this growth. Adapting only the MF parameter results in about 2% reduction in the trend growth relative to the base (MF1000). On the other hand when MF is combined with frame skipping, up to 21.6% growth reduction is achievable.

\[1\] The adaptation range of each parameter is tuned so that we maintain the similar improvement on time achieved in [2] and maximise the improvement on energy with minimal impact on the accuracy.
Figure 5.8: The incremental memory usage of ORBSLAM2, running on the sequence Fr3-long from TUM RGB-D, as a result the growing factor graph. The slow rise in memory using when adapting all three parameters (combined) compared to the base case (MF1000) is mainly due to skipping frames which filters redundant poses and subsequently reducing the factor graph size.

the end of the sequence. This implies that it is possible to prolong the tracking operation by up to 21.6% before running out of main memory, if the same trend continues at the same pace.

5.3.2 Performance on the Stereo KITTI

In this section, we evaluate the scalability of SLAM-dunk on the large-scale driverless car dataset KIITI [12]. The sequences in this dataset span large urban neighbourhoods and highway segments, some with a travelled distance more than 5 Km. For the evaluation, we use the stereo version of ORBSLAM2 which is suitable, given its ability to tackle large scale maps and being a full SLAM algorithm. Our goal is to evaluate the behaviour of SLAM-dunk in such dataset in terms of it is ability to cope with large scale distances. We adapt the MF parameters which has a default value of 2000 on this dataset, thus we define the range as \( \{MF|510 \leq MF \leq 2000\} \). Similar to the previous section, a total of ten consecutive runs are performed for each sequence of the dataset. The results are
Table 5.2: A summary of the results from the large-scale KITTI dataset (sequence 00 to 10) shown as violin plots in Figure A.4. The results show the improvements made when adapting the MF value (MF510-2000) relative to the base case where MF is fixed at 2000 (MF2000). (*) sequences ended prematurely due main memory limit on the Jetson.

SLAM-Dunk is shown to improve the overall per-frame computation time by around 22%, and energy consumption by around 26%. This is with an overall negative impact on the accuracy of 1.54% or about 1.6 cm drift from the trajectories that resulted with the base settings (MF2000). This shows that SLAM-Dunk can also be used to improve VSLAM performance operating in large-scale environments. One interesting observation from the results of sequence 04, is that the improvement in energy made by the adaptation is small (about 1.1%). This sequence starts with the car moving in a straight line at a fixed speed which implies lack of significant motion variations (in both rotation and translation) even though there are changes in the scene (resulting in a $\sigma$ value close to or equal to 1). In such a case, SLAM-dunk behaves in a safe or conservative manner in terms of saving computation time or energy, despite the accurate result. These objectives could be improved, in such a case, by introducing weights on the three
metrics, or relying on previous history records of the maximum variation in the characterisation metrics, if available, and if a non-monocular sensor is used where the scale is fixed.

5.4 Summary

In this chapter, we explored the performance of SLAM-Dunk on sparse VSLAM formulations using both general and portable tuning parameters. In Section 5.2, we have shown that full portability is achievable under challenging settings and different sparse VSLAM algorithms using pure monocular-based datasets and different platforms. While in Section 5.3, application-specific parameters are explored along with portable parameters on two additional datasets with different sensors, further demonstrating the portability and versatility of the framework.

In the following chapter, we show that SLAM-Dunk can also be tuned for a specific algorithm and platform by exploring different predefined range settings for the two parameters, DVFS and Skipping Frames.
Chapter 6

Exploring performance on dense VSLAM

In this chapter, we perform a design-time exploration of the adaptation range of the two portable parameters, DVFS and frame skipping, to improve Elastic Fusion (EF) performance objectives at run time. We aim to establish two aspects: the first is to explore the extent of performance portability across dense and sparse VSLAM. That is, the possibility of using the same parameter ranges used for the two sparse VSLAM algorithm, DSO and ORBSLAM, to gain similar improvements on the dense VSLAM (Elastic Fusion). The second aspect is to find out the potential of the framework to improve the performance objectives of a specific VSLAM operating in the same environment and using the same platform. Section 6.1 describes the setup used for the evaluation in this chapter. In Section 6.2, we describe the space explored and present the results, where the focus is on the improvements made with a portable configuration. We then evaluate the impact on the map accuracy and density.

6.1 Experimental Setup

In this chapter, we target the ICL-NUIM living room dataset since it has a ground truth 3D model map which is used for evaluating the quality of the constructed map. The RGB-D frames are used with EF, but SLAM-Dunk relies upon the converted grey-scale to measure the pyramid-based EMD metric. EF is evaluated on the Jetson using SLAMbench3 [15] which reports the mean Absolute Trajectory Error (ATE) [58]. The default settings are used except for a depth
cutoff distance of 5m, which we found improves the accuracy for LR0 and LR2 (the default is 3m) [25]. EF implementation is CUDA-based, relying heavily on the GPU to perform computations and a single thread running on the CPU. However, we control DVFS on both the GPU and CPU processing cores to observe any additional benefit. EF does not originally meet the QoS (that is 30 FPS) on the Jetson; thus, the linearised mode is used with EF and the framework to improve its performance.

Unlike DSO and ORBSLAM2, EF has a deterministic behaviour, which means that there is negligible variability in the resulting map or trajectory over multiple runs and fewer outliers in the sensor trajectory. Given such deterministic behaviour, the outlier rejection window is set to 1 (described in Section 4.3). We also focus on using $\sigma$ (defined in Section 4.4) as a characterisation metric because it performed well on the two non-deterministic algorithms, as shown in Section 5.2, so we anticipate that it will also result in accurate tracking on a deterministic formulation such as EF.

6.2 Exploring Parameters Adaptation Range

Since a predefined parameter range is used on each parameter, we aim to find parameter range configurations that achieve good performance across the different scene trajectories while having minimal impact on the trajectory accuracy and map density compared to the base EF. The goal here is to target a single VSLAM running in a particular environment while exploring the adaptation range, unlike what has been done in Section 5.2, where the same adaptation range was chosen to target multiple sparse algorithms, environments and platforms.

We explore the impact of varying the (predefined) ranges of the two parameters (DVFS and redundant frame skipping) on the ability to trade-off Elastic Fusion (EF) performance metrics: per-frame execution time and power consumption, and the accuracy of trajectory and 3D map reconstruction quality.

As mentioned above, the base version of EF does not meet the real-time constraint on all of the trajectories used; it runs on average in 0.067 seconds per frame or about 15 FPS. To improve its performance, we explore increasing the maximum value of skipped frames (i.e. redundant frames) in batches of size $Y_{max}$, while adjusting the minimum value of the DVFS range, $X_{min}$, on each processor and maintaining $X_{max}$ at the maximum frequency available on the
target processor. Our primary objective is reducing the power used by the Jetson with minimal impact on the baseline trajectory accuracy and map density; that is, we aim for an ATE below 5cm [17] while achieving a real-time performance of 30 FPS (Average per-frame time of 0.033 seconds).
Figure 6.1: Pareto fronts of exploring DVFS and frame skipping range with respect to power consumption on both CPU and GPU of the Jetson. The per-frame average execution time (a) and the ATE (b). The coloured points are runs based on different configurations of the two explored parameters, where different colours on the colour bar indicate the number of skipped frames. While the variations in the intensity of each colour represent DVFS steps on both processor where the most intense represents a configuration with the highest frequency on the GPU and on the CPU. The base EF is marked with (◦) while (×) marks the first portable configuration, across sequences, that meets the performance and accuracy objectives which are marked with dashed lines.
Figure 6.1a and 6.1b show the Pareto fronts resulting from the exploration of the design space or the above parameters configurations. The Pareto fronts of the per-frame average computation time with respect to power consumption are shown in Figure 6.1a and the mean ATE in Figure 6.1b. On the Jetson, the base EF violates the 5 cm ATE constraint on LR3, which is consistent with the results obtained in [25] using a high-end workstation; thus, it is excluded from both figures. We also exclude configurations that result in an average computation time larger than the base EF (circled and coloured with the darkest red in the figure) or does not meet the error constraint (> 5 cm). The figure has four distinct colours representing the total number of redundant frames skipped in batches, where skipping zero frames is represented by red, and so on up to skipping three frames in batches (in blue). Although on some trajectories, it is possible to skip a larger number of frames before violating the accuracy constraint, such configurations are not applicable to all of the trajectories. This is why the maximum number of frames skipped in a batch is three.

For a given batch which is colour coded, we vary the minimum DVFS range on the two processors; this variation is represented in the figures by the colour intensity, where the most intense shade means that the minimum frequency is equal to the highest frequency available on each processor (so there is no range available) and the lightest shade means that the full frequency range is used for the tuning.

Looking at Figure 6.1a, it can be seen that increasing the number of skipped (redundant) frames improves the execution time and the power consumption on both processors as a result of avoiding the need to track and reconstruct similar surfaces. This resulted in some configurations meeting the QoS constraint (horizontal dashed line at 0.033 s). Further, redundant frame skipping is shown to improves power consumption and execution time but does not violate the trajectory accuracy constraint of 5 cm in Figure 6.1b. On the other hand, adjusting the tuning range for DVFS improves power consumption mainly on the GPU, but impacts only EF execution time, rather than the accuracy as evident in Figure 6.1b, which imply that the impact of DVFS tuning on the ATE is negligible.

In terms of portability, using the same adaptations range used for the two sparse VSLAM algorithms is possible on dense EF. However, it will not deliver the desired QoS (that is 30 FPS) given the gap between the computations intensity of the two classes. The framework adaptation policy and characterisation
metrics are still portable, however, implying that portability is achievable to an extent between sparse and dense formulations. In the following section, we discuss the portability of parameter adaptation range configuration at the level of the environment.

6.2.1 Portable Configuration for Adaptations Range

In terms of power reduction, the Pareto Fronts in the top subplot of Figure 6.1a shows that the power consumed by the GPU can be reduced by approximately 77%, 60% and 62% on LR0, LR1, and LR2 respectively. This is achieved while maintaining an execution time equal to, or less than, the base EF. However, since we want to meet the QoS requirement, we are limited in terms of the available configurations. Mainly we are interested in a configuration that achieves the real-time constraint with the least power consumed, and which is portable across the trajectories, i.e. portable, here, is used in the sense that by using this specific configuration, the majority of trajectories meet the performance objectives. The figure highlights such configuration by ×. In Figure 6.1a, there are configurations particular to each trajectory where further power savings can be made while meeting the real-time constraint compared to the portable configuration (×), especially on the dominant processing unite, the GPU. This implies that further power savings can be gained by adapting the predefined range online. In this case, however, we can observe that the portable configuration is already performing well since it is not that far from achieving these power savings.

<table>
<thead>
<tr>
<th>Base EF</th>
<th>Per-frame Avg. Time (s)</th>
<th>LR0</th>
<th>LR1</th>
<th>LR2</th>
<th>LR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATE (m)</td>
<td>0.0146</td>
<td>0.0078</td>
<td>0.0058</td>
<td>0.2248</td>
<td></td>
</tr>
<tr>
<td>CPU Power (W)</td>
<td>6.5225</td>
<td>6.3971</td>
<td>6.7659</td>
<td>6.1595</td>
<td></td>
</tr>
<tr>
<td>GPU Power (W)</td>
<td>1.8468</td>
<td>1.4539</td>
<td>1.2721</td>
<td>1.5928</td>
<td></td>
</tr>
<tr>
<td>EF @ 30 FPS</td>
<td>Per-frame Avg. Time (s)</td>
<td>0.0324</td>
<td>0.0316</td>
<td>0.0330</td>
<td>0.0297</td>
</tr>
<tr>
<td>ATE (m)</td>
<td>0.0146</td>
<td>0.0265</td>
<td>0.0163</td>
<td>0.4408</td>
<td></td>
</tr>
<tr>
<td>CPU Power (W)</td>
<td>2.7638</td>
<td>4.2079</td>
<td>4.0153</td>
<td>3.1593</td>
<td></td>
</tr>
<tr>
<td>GPU Power (W)</td>
<td>1.4925</td>
<td>1.2460</td>
<td>1.1958</td>
<td>1.2528</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: The performance of EF base along with a real-time portable range configuration using the framework.

Table 6.1 shows a comparison between the base EF and the real-time adapted
version using this portable configuration. As can be seen, the GPU power savings range between 34% and 58%, and between 6% and 21% on the CPU. These savings are mainly due to redundant frame skipping. The sequence LR0 has the highest power saving achieved with our adaptation framework, which implies that it has a large amount of redundant scenery compared to the other sequences. Given that EF operates in the linearised mode where both power savings and time are both improved, improvements on the per-frame energy of both processing units can also be quantified, which ranges between 65% and 77%.

### 6.2.2 Map Accuracy and Quality

A key aspect of dense VSLAM is to construct an accurate and dense 3D map. Thus, it may be expected that there will be an impact on the quality of the constructed map due to frame skipping. The impact, however, is minimised in our framework since only highly redundant frames are skipped.

![Image of map accuracy and quality](image.png)

Figure 6.2: The impact of skipping redundant frames on the map quality and its reconstruction error using ICL-NUIM living room sequences with respect to the Base EF. The colour on the map represents the reconstruction error in meters.

To evaluate the quality of the map constructed when using our adaptation...
with that of the base EF, we measure the reconstruction error of both maps with respect to the ground truth 3D model. Since the main source of impact on the map is expected to be redundant frame skipping, we evaluate the reconstruction when each of one, two and three frames are skipped in batches. We follow an approach similar to that of [59] for registering and evaluating the construction quality. That is, the constructed maps are first coarsely aligned with the ground truth mesh (manually), as described in Section 2.2.1.3, and then a fine alignment is performed using Iterative Closest Point (ICP) [71]. Finally, the perpendicular distance between each vertex on the constructed map and the closest triangle on the ground truth mesh is measured. We report both the quantitative results along with the visual construction of the estimated maps next.

<table>
<thead>
<tr>
<th>Error (m)</th>
<th>LR0</th>
<th>LR1</th>
<th>LR2</th>
<th>LR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skip 0</td>
<td>mean</td>
<td>0.0038</td>
<td>0.0048</td>
<td>0.0027</td>
</tr>
<tr>
<td></td>
<td>median</td>
<td>0.0029</td>
<td>0.0034</td>
<td>0.0025</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>0.0034</td>
<td>0.0047</td>
<td>0.0017</td>
</tr>
<tr>
<td></td>
<td>min</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>max</td>
<td>0.0580</td>
<td>0.0322</td>
<td>0.0187</td>
</tr>
<tr>
<td>Skip 1</td>
<td>mean</td>
<td>0.0049</td>
<td>0.0041</td>
<td>0.0033</td>
</tr>
<tr>
<td></td>
<td>median</td>
<td>0.0045</td>
<td>0.0035</td>
<td>0.0028</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>0.0033</td>
<td>0.0027</td>
<td>0.0025</td>
</tr>
<tr>
<td></td>
<td>min</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>max</td>
<td>0.0264</td>
<td>0.0204</td>
<td>0.0181</td>
</tr>
<tr>
<td>Skip 2</td>
<td>mean</td>
<td>0.0037</td>
<td>0.0045</td>
<td>0.0037</td>
</tr>
<tr>
<td></td>
<td>median</td>
<td>0.0034</td>
<td>0.0040</td>
<td>0.0029</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>0.0026</td>
<td>0.0031</td>
<td>0.0030</td>
</tr>
<tr>
<td></td>
<td>min</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>max</td>
<td>0.0278</td>
<td>0.0251</td>
<td>0.0203</td>
</tr>
<tr>
<td>Skip 3</td>
<td>mean</td>
<td>0.0037</td>
<td>0.0048</td>
<td>0.0062</td>
</tr>
<tr>
<td></td>
<td>median</td>
<td>0.0035</td>
<td>0.0043</td>
<td>0.0046</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>0.0025</td>
<td>0.0034</td>
<td>0.0054</td>
</tr>
<tr>
<td></td>
<td>min</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>max</td>
<td>0.0311</td>
<td>0.0314</td>
<td>0.0437</td>
</tr>
</tbody>
</table>

Table 6.2: The impact of skipping redundant frames on the reconstruction error with respect to the ground truth map.

Table 6.2 shows EF reconstruction error statistics for the four ICL-NUIM trajectories (LR0 to LR3) for the base EF with zero to three frames being skipped. A visual representation of the reconstructions for these cases showing the error in the maps is shown in Figure 6.2.
As can be seen from the table, the reconstruction error where redundant frames are skipped is on a par with that of the base EF, and, in the worst case, is less than 5 cm except for LR3, where the base EF does not perform well either.

When inspecting the scene in the LR3 sequence, it would appear that the impact on the map reconstruction is caused by a low textured surface in the scene, which results in large drift. The problem is exacerbated by the inability of the base EF to perform full loop closure on the Jetson. Given that the base does not perform well on this sequence, and we do not employ any metric for smooth surface detection, it is to be expected that the adapted version will perform similarly.

<table>
<thead>
<tr>
<th></th>
<th>LR0</th>
<th>LR1</th>
<th>LR2</th>
<th>LR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skip 1</td>
<td>14.90%</td>
<td>8.53%</td>
<td>6.00%</td>
<td>13.23%</td>
</tr>
<tr>
<td>Skip 2</td>
<td>22.80%</td>
<td>17.03%</td>
<td>7.28%</td>
<td>24.63%</td>
</tr>
<tr>
<td>Skip 3</td>
<td>31.58%</td>
<td>26.77%</td>
<td>14.32%</td>
<td>30.85%</td>
</tr>
</tbody>
</table>

Table 6.3: The impact of skipping redundant frames which manifests itself as a reduction (%) in the constructed map surface area relative to that of the EF base maps.

In terms of the quality of the reconstruction, we use the constructed surface area to measure the impact of skipping frames relative to the base EF map of each trajectory. This impact manifests itself as a reduction in the constructed area, which is shown in Table 6.3. The problem is most prominent when a batch of three redundant frames are being skipped with a reduction in the constructed surface area ranging between 14.3% to 31.6% relative to the base map of each trajectory. Looking at the visualised reconstructions for LR0 in Figure 6.2, for example, we can detect that regions that are visited only once with a shift in the camera motion direction are those most affected. In LR0, this occurs for the right-hand side wall of the room. Such motion usually implies that this region is of the least interest in the scene. The majority of the map is still dense, however, with power-efficient, accurate tracking and mapping and real-time performance being achieved using only the two portable parameters being tuned.

To obtain more dense maps, one may run EF at the lower rate of 20 FPS (0.05 ms), which is an acceptable rate according to [25], while skipping a batch of one redundant frame, and still save power, as can be observed from Figure 6.1a. Such trade-offs can easily be achieved with our framework.
6.3 Summary

In this chapter, we employed SLAM-Dunk to improve the performance of a dense VSLAM algorithm by exploring the space of the parameters’ adaptation range configuration. We showed that portability is achievable to an extent across sparse and dense formulations, with the main limiting factor being the difference in computation performed by the two formulations. However, by exploring the range of parameters, it is possible to achieve a real-time frame rate of 30 FPS and power reduction of up to 58%. This improvement comes at the expense of a reduction in the reconstructed map surface area; however, this reduction is mainly in areas of least interest and with a shift in motion direction. Further, the exploration gave insights into the potential benefits of adapting the parameter range online, which we aim to explore in the future.
Chapter 7

Conclusion and Outlooks

This chapter concludes the thesis by first summarising its result, then reviewing its contributions and finally presents possible research routes for future work.

7.1 Summary

This thesis explored portable runtime adaptation on three VSLAM algorithms, representing different classes, where the goal is to improve performance objectives with minimal impact on the accuracy and robustness. In doing so, we answer the three research questions introduced in Chapter 1. To achieve this, a framework that is efficient, maintainable and portable across VSLAM formulations and platforms has been developed.

We first studied frame skipping and DVFS - both of which are examples of general parameters that target VSLAM and the platform, respectively - in a simulated environment with limited motion patterns and a rich scene. We introduced a basic adaptation model and showed that adapting each parameter based on knowledge about the VSLAM tracking difficulty achieves the desired performance objective compared to methods that lack such knowledge. Applying frame skipping using our basic adaptation model, which utilises selected characterisation metrics achieves up to a 50% power reduction with minimal impact on the accuracy, relative to the ground truth. However, similar results cannot be achieved by using an approximate computing method known as loop perforation, since frame skipping is then performed in a way which is agnostic to the tracking difficulty. Similarly, when exploring the use of DVFS, default DVFS governors, such as on-demand or powersave, do not deliver the desired power and accuracy balance,
when compared to the tracking difficulty-based adaptation which achieves a 40% power reduction with almost no impact on the accuracy. The combination of the two parameters was found to not lead to a negative impact on the accuracy and achieves up to a 79% power reduction.

In realistic environments, however, the motion patterns are typically diverse, and the scene may not be adequate in terms of tracking information. This dictated improving the framework of the basic adaptation model and seeking improved characterisation metrics of tracking difficulty. In the extended framework, first, motion sources (rotation and translation) are decoupled, and a novel pyramid-based EMD metric was introduced to quantify scene changes. Second, a novel adaptation policy that normalises the characterisation metrics was presented and shown to be able to adapt to unknown variations in the metrics and to reject outliers in their values. The improved framework, SLAM-dunk, was then used to study the performance portability of the characterisation metrics on two sparse formulations, DSO and ORBSLAM2, using two monocular-based datasets, ICL-NUIM and EuRoC MAV, with diverse motion patterns and scenes, during execution with a hard deadline constraint on two platforms with different architectures, x86 and ARMv8.

Experiments using the same framework settings and the same adaptation ranges of the two parameters, DVFS and frame skipping demonstrate that portable performance is generally achievable when the characterisation metrics are combined, resulting in power reductions of between 43% and 69%, with a marginal impact on the accuracy and robustness. This confirms that relying on a single characterisation metric, as is currently common in related work on runtime adaptation, is limited when considering the portability and robustness, and leads to a less reliable adaptation framework.

We further study the performance of SLAM-dunk on ORBSLAM2 using two additional datasets with different sensors, motion patterns and scales operating under the linearised mode described in 2.2.2. We show that SLAM-dunk achieves marginal improvements to the accuracy by adapting ORBSLAM2-specific parameter using the RGB-D TUM dataset with slightly conservative improvements to ORB-SLAM2 per-frame time and energy compared to a recent adaptation framework which shows an impact on the accuracy. However, when both general and application-specific parameters are adapted simultaneously, SLAM-dunk still achieves less impact on the accuracy and outperforms the recent framework.
in terms of per-frame energy (42%) while achieving the similar improvements in computation time. We also used the stereo KITTI, a large scale driving dataset to show the scalability of SLAM-dunk which is shown to improve the overall per-frame computation time by around 22%, and energy consumption by around 26%.

The SLAM-Dunk framework has also been used to explore the potential of the two portable tuning parameters, DVFS and redundant frame skipping, to improve the performance of a Dense VSLAM, Elastic Fusion, on a mobile Nvidia Jetson robotics platform. The goal was to explore the trade-offs involved in achieving real-time performance with a computationally demanding dense formulation while minimising the impact on the accuracy. This exploration resulted in up to 56% reductions in the execution time and up to 77% increase in energy savings while maintaining both trajectory and map accuracy at an acceptable level.

Finally, it was found that using the same portable parameter adaptation ranges (DVFS and frame skipping) used with the sparse formulation would not deliver the desired improvements to the performance of the dense Elastic-Fusion VSLAM formulation. Even though the framework is seen to be portable, in terms of the selected characterisation metrics and the adaptation policy (with the exception of the choice of window size of 1 instead of 5 for the management of outlier values), the ranges specified for the adaptation parameters need to be relaxed in order to meet real-time performance on the dense, Elastic Fusion, VSLAM algorithm. Portability on all levels was seen to be achieved across different sparse formulations; however, portability is only achievable to an extent across sparse and dense formulations.

Overall, the framework is conservative in how adaptation is applied, with accuracy the highest priority. This conservatism is achieved by using linear functions on the delta of the normalised metrics for parameter tuning, but with a pessimistic approach for frame skipping, i.e. making it convergent towards the minimum value. Whereas, for DVFS or ORBSLAM2’s maximum feature (MF) parameters, the approach is an optimistic one becoming convergent towards the maximum value as the tracking difficulty increases.
7.2 Review of Contributions

In this thesis, we have shown that VSLAM performance requirements can be achieved in a portable fashion in a diverse problem space. In the following, we summarise the contributions that resulted from this research.

**SLAM-Dunk** a novel lightweight tuning framework for improving VSLAM performance at runtime. The framework monitors a range of metrics which characterise the motion and scene information content and adapts these to provide an estimated approximation of the current tracking difficulty. Approximating the tracking difficulty allows SLAM-Dunk to alter parameters related to both the VSLAM system and the platform on which it is executing. These parameters allow the framework to trade-off performance objectives, e.g. power use and accuracy, dynamically at runtime. The framework is modular, allowing both easy extension to new metrics that may become available and the inclusion of new control policies, with minimal changes to the targeted VSLAM implementation. Such a framework is key to ensuring accurate and robust tracking on various VSLAM systems operating in environments with diverse scenes and motion patterns which change in unknown ways as the sensor moves through the environment.

**A novel adaptation policy** based on metric normalisation. The policy enables the framework to operate across the different VSLAM systems and datasets while delivering the desired performance objectives. Performance portability is difficult to achieve with a control framework that requires a large number of thresholds (hyperparameters) to be specified, as is typical in approaches that aim to improve certain VSLAM systems currently. The adaptation policy requires only a single hyperparameter to be tuned in the presence of outliers in the measured metrics.

**A novel efficient scene characterisation metric** based on measuring the distance between a pair of coarse Gaussian pyramids which is used as a metric characterising the change in the scene, along with the readily available motion metrics for rotation and translation. The new metric accounts for both spatial and intensity changes across the frames.

**A performance analysis and benchmarking** of the presented framework components with adapted tuning parameters over different VSLAM algorithms operating in diverse sets of environments and with a range of sensors and under different constraints. This shows the versatility of the presented framework.
7.3 Limitations and Future Work

This thesis showed that the framework could achieve power reduction with minimal impact or improved accuracy relative to the base VSLAM algorithm. However, it is challenging to know the optimal power reduction, for example, that can be achieved without comparing it with an oracle solution. We plan to investigate the possibility of providing such an oracle in the context of the used parameters. Such an oracle should result in an accuracy similar to the ground truth, maximise power reduction and minimise per-frame computation time.

The current SLAM-Dunk framework can be extended, given the potential for further research into the runtime performance and portability of characterisation metrics and tuning parameters across the diverse application space of VSLAM. In terms of characterisation metrics, there are two possible paths to exploring improved portability and effectiveness: information theoretic-based metrics and learning-based metrics trained based on both the scene and motion patterns in diverse problem settings. Besides improving VSLAM performance objectives, additional metrics can also be used to assist in tasks associated with VSLAM; an example would be when to perform semantic segmentation to ensure accurate labelling while not impacting VSLAM performance objectives, especially on low-end platforms.

Currently, the parameters adjustment is based on the linear relationship assumption; the tuning may be improved by exploring the relationship type with the desired performance objectives. Moreover, the predefined ranges to be used with the tuned parameters need to be explored and set at design time. In future, we plan to examine the feasibility of changing the ranges dynamically at runtime since this can be expected to provide further improvements in VSLAM performance.
Bibliography


[22] A. Billy, S. Pouteau, P. Desbarats, S. Chaumette, and J. P. Domenger, “Adaptive slam with synthetic stereo dataset generation for real-time dense


[31] N. Yang, R. Wang, X. Gao, and D. Cremers, “Challenges in Monocular Visual Odometry: Photometric Calibration, Motion Bias, and Rolling Shutter


Appendix A

Supplementary Materials

This Appendix presents supplementary materials for the results presented in Section 5.3. It contains violin figures of the ten runs performed on the RGB-D TUM and KITTI dataset as results of exploring runtime adaptation on ORBSLAM2 using application-specific and general parameters. It also includes summary tables of these results along with a sample trajectories.
Figure A.1: Sample sequence trajectories from RGB-D TUM where each subplot row represent the ten trajectories corresponding to the RMSE violin plots in Figure A.3
Figure A.2: Sample trajectories from the Stereo KITTI.
Table A.1: Results from [2] on a TX2 Nvidia Jetson. These results are used for comparison in Section 5.3.1.

<table>
<thead>
<tr>
<th>Location</th>
<th>Time (s)</th>
<th>Energy (J)</th>
<th>RMSE (m)</th>
<th>Time (s)</th>
<th>Energy (J)</th>
<th>RMSE (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fr1-desk</td>
<td>0.394</td>
<td>1741</td>
<td>0.031</td>
<td>0.284</td>
<td>1254</td>
<td>0.035</td>
</tr>
<tr>
<td>Fr1-desk2</td>
<td>0.403</td>
<td>1776</td>
<td>0.047</td>
<td>0.352</td>
<td>1551</td>
<td>0.043</td>
</tr>
<tr>
<td>Fr1-floor</td>
<td>0.161</td>
<td>623</td>
<td>0.019</td>
<td>0.128</td>
<td>505</td>
<td>0.021</td>
</tr>
<tr>
<td>Fr1-plant</td>
<td>0.343</td>
<td>1503</td>
<td>0.049</td>
<td>0.26</td>
<td>1212</td>
<td>0.042</td>
</tr>
<tr>
<td>Fr1-room</td>
<td>0.278</td>
<td>1221</td>
<td>0.171</td>
<td>0.225</td>
<td>1079</td>
<td>0.219</td>
</tr>
<tr>
<td>Fr-xyz</td>
<td>0.281</td>
<td>1239</td>
<td>0.014</td>
<td>0.237</td>
<td>1041</td>
<td>0.013</td>
</tr>
<tr>
<td>Fr2-desk</td>
<td>0.267</td>
<td>1090</td>
<td>0.029</td>
<td>0.191</td>
<td>789</td>
<td>0.02</td>
</tr>
<tr>
<td>Fr2-dwp</td>
<td>0.296</td>
<td>1256</td>
<td>0.009</td>
<td>0.224</td>
<td>964</td>
<td>0.011</td>
</tr>
<tr>
<td>Fr2-xyz</td>
<td>0.171</td>
<td>726</td>
<td>0.006</td>
<td>0.138</td>
<td>609</td>
<td>0.005</td>
</tr>
<tr>
<td>Fr3-long</td>
<td>0.394</td>
<td>1688</td>
<td>0.025</td>
<td>0.225</td>
<td>971</td>
<td>0.052</td>
</tr>
<tr>
<td>Fr3-near</td>
<td>0.153</td>
<td>651</td>
<td>0.049</td>
<td>0.116</td>
<td>504</td>
<td>0.059</td>
</tr>
<tr>
<td>Fr3-near</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Fr3-st-near</td>
<td>0.196</td>
<td>842</td>
<td>0.015</td>
<td>0.12</td>
<td>528</td>
<td>0.018</td>
</tr>
<tr>
<td>Fr3-st-far</td>
<td>0.198</td>
<td>853</td>
<td>0.018</td>
<td>0.139</td>
<td>615</td>
<td>0.018</td>
</tr>
</tbody>
</table>

**Geo. Mean**

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Energy (J)</th>
<th>RMSE (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.257</td>
<td>1099.598</td>
<td>0.025</td>
</tr>
<tr>
<td>0.191</td>
<td>836.655</td>
<td>0.027</td>
</tr>
</tbody>
</table>

**Improvement(+) / Impact(-)**

25.69% 23.91% -7.16%
<table>
<thead>
<tr>
<th></th>
<th>Base (MF1000)</th>
<th>MF510-1000</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (s)</td>
<td>Energy (J)</td>
<td>RMSE (m)</td>
</tr>
<tr>
<td>Fr1-desk</td>
<td>0.056</td>
<td>0.430</td>
<td>0.022</td>
</tr>
<tr>
<td>Fr1-desk2</td>
<td>0.057</td>
<td>0.438</td>
<td>0.030</td>
</tr>
<tr>
<td>Fr1-floor</td>
<td>0.045</td>
<td>0.294</td>
<td>0.028</td>
</tr>
<tr>
<td>Fr1-plant</td>
<td>0.054</td>
<td>0.431</td>
<td>0.015</td>
</tr>
<tr>
<td>Fr1-room</td>
<td>0.053</td>
<td>0.401</td>
<td>0.052</td>
</tr>
<tr>
<td>Fr-xyz</td>
<td>0.056</td>
<td>0.415</td>
<td>0.012</td>
</tr>
<tr>
<td>Fr2-desk</td>
<td>0.060</td>
<td>0.437</td>
<td>0.008</td>
</tr>
<tr>
<td>Fr2-dwp</td>
<td>0.063</td>
<td>0.430</td>
<td>0.006</td>
</tr>
<tr>
<td>Fr2-xyz</td>
<td>0.053</td>
<td>0.342</td>
<td>0.003</td>
</tr>
<tr>
<td>Fr3-long</td>
<td>0.064</td>
<td>0.462</td>
<td>0.014</td>
</tr>
<tr>
<td>Fr3-nt-near</td>
<td>0.052</td>
<td>0.335</td>
<td>0.021</td>
</tr>
<tr>
<td>Fr3-sn-far*</td>
<td>0.018</td>
<td>0.106</td>
<td>0.007</td>
</tr>
<tr>
<td>Fr3-st-near</td>
<td>0.055</td>
<td>0.364</td>
<td>0.012</td>
</tr>
<tr>
<td>Fr3-st-far</td>
<td>0.057</td>
<td>0.368</td>
<td>0.013</td>
</tr>
<tr>
<td>GeoMean</td>
<td>0.0555</td>
<td>0.3938</td>
<td>0.0143</td>
</tr>
<tr>
<td>Improvement (+)</td>
<td>18.33%</td>
<td>19.63%</td>
<td>2.88%</td>
</tr>
</tbody>
</table>

Table A.2: Result on RGB-D TUM: the summary of mean values over ten runs shown as violins plot in Figure A.3, after adapting the maximum feature allowed for ORBSLAM2 between the range of 510 to 1000 (MF510-1000) relative to the default fixed value (MF1000). The combination of this parameter along both DVFS and frame skipping is seen to have extra improvements. (*) not included in the geometric mean computation as tracking terminated prematurely on all cases.
Figure A.3: Result on RGB-D TUM: Violin plots over ten runs when adapting the maximum feature allowed for ORBSLAM2 between the range of 510 to 1000 (MF510-1000) relative to the default fixed value (MF1000) and the combination of this parameter along both DVFS and frame skipping (Combined). (*) Tracking terminated prematurely on all cases.
Figure A.4: Violin plots of the ten runs on KITTI dataset. (*) sequences ended prematurely due main memory limit on the Jetson.