The Univeristy of Western Australia
School of Electrical and Electronic Engineering

GENG5512 Engineering Research Project
Final Report

Implementation of Visual SLAM for Autonomous Shuttle Bus

Thomas Copcutt
22248715

Supervised by
Prof. Dr Thomas Bräunl

24 July 2022

Words in text: 6845

In this research a Visual SLAM solution with GPU utilisation is proposed for the nUWAy shuttle bus to replace the currently implemented LiDAR based SLAM solution. This follows from an underutilised GPU and issues of bottlenecking from the CPU experienced during execution of the bus's autonomous stack. Visual SLAM was chosen as it can make better use of the underutilised GPU than the LiDAR SLAM solution, which runs on CPU. Such work has promise for application in many other contexts on systems using heterogeneous CPU-GPU and System on Chip architectures.

Proven and mature Visual SLAM algorithms exist in prior research and shows potential for parallelisation of computation on the GPU. Adoption and modification of these implementations ensures an efficient base solution with minimal initial computational cost. Key areas of the SLAM algorithm are identified which have both high parallelisation potential and a significant compute expense. These tasks will be focused on for execution on the GPU.

The success of this project will ultimately be determined by the ability to lower the load on the CPU, as this result will successfully reduce the extent of bottlenecking of the system. It's unknown whether computation on the GPU in these scenarios will improve compute times so such metrics will also be assessed.

The performance will be reported based on:

a) Known to work ORB-SLAM examples, processed on a high-powered graphics workstation.

b) Pre-recorded data of the bus in a typical environment, processed on a high-powered graphics workstation.

c) Pre-recorded data of the bus in a typical environment, processed on the bus's internal systems.

d) Live data from bus cameras processed on the bus's internal systems during operation.

These datasets will be compared where possible to both an unmodified visual SLAM solution with which processing is done on CPU, as well as the current LiDAR based SLAM solution to determine a final feasibility of implementing the change.

Contents

Nomenclature

Table 1: Table of Nomenclature

Assignment	**Definition**
SLAM	Simultaneous Localisation and Mapping
vSLAM	Visual Simultaneous Localisation and Mapping
CPU	Central Processing Unit
GPU	Graphics Processing Unit
ROS	Robot Operating System
CAN	Controller Area Network
Base system	The ORB-SLAM2 implementation of vSLAM created by Mur-Artal and Tardos
Onboard computer	The Jetson Xavier computer on the nUWAy shuttle bus
Offboard computer	The workstation computer in the UWA Robotics Lab used for testing the design.

Chapter 1

Introduction

1.1 Background

The adoption of autonomous vehicles is predicted to be one of the next major transitions not only in the automotive industry but in human society as a whole. Driving in urban off road environments introduces many difficulties not present in on road driving. Today's road worthy autonomous cars rely heavily on road markings as well as accurate and detailed online road maps in conjunction with GPS to localise themselves and stay centred in a lane [2], [3]. The lack of predefined map or markings results in the requirement of a much higher level of mapping, localisation and path planning. As cars level of autonomy increases a more robust solution will be needed [4].

Currently autonomous vehicle companies are using a combination of both camera based technology and LiDAR [5]. However, there's great interest in the idea of foregoing LiDAR due to the reduced cost of cameras and the expected continued development in the computer vision field [5]. It's expected that vehicles without LiDAR systems will be able to reach full autonomy in the future [6].

Visual SLAM has interest from many industries including robotics, augmented reality, and 3D modelling [7], [8]. The challenge with the use of cameras, particularly monocular cameras, is that a 3D map of the environment must be determined from a set of 2D data. A combination of mathematical techniques will be discussed which handle this reconstruction.

1.1.1 nUWAy Shuttle Bus

Figure 1.1: The nUWAy Shuttle Bus

The nUWAy shuttle bus is a modified EasyMile EZ10 bus with an array of perception sensors to aid in autonomous driving. The bus currently runs using two computers. The first is an industrial PC which was fitted by the original manufacturer and largely serves to interface with the CAN bus and sensors. An additional Nvidia Jetson AGX Xavier was added to improve the processing power of the bus overall. It is planned that the bus will make routine trips along a main thoroughfare between the UWA Reid Library and the UWA business school. The bus will stop at these locations and allow students to embark and disembark to avoid the long walking journey between these two popular locations. It is now in the final stages before it can be run with the public, however there are still many areas of improvement that can be made once the bus is in operation.

1.2 Problem Statement

The nUWAy shuttle bus currently uses a LiDAR based SLAM implemented using SLAM Toolbox on ROS2. This method largely uses CPU for computation and competes with resources against many other systems run on the bus. As a result, the SLAM process was observed to not be able to operate in conjunction with the full autonomous stack. Additionally, new EZ10 shuttle buses may be aquired without front and back facing velodyne LiDAR units which may impact the current LiDAR SLAM solution. It has been proposed that a visual based SLAM implementation could make better use of the currently underutilised graphics processing unit on board one of the buses PC's and lower CPU load. This would ultimately allow for operation of SLAM in a lifelong mapping mode which has benefits mentioned in Section 2.1 for the long term successful operation in the dynamic campus environment.

1.3 Intended Contribution

This project intends to improve the overall performance of the bus by implementing tasks which can run on the GPU. Assessment of the feasibility of vSLAM for use on the nUWAy shuttle bus will be conducted and its performance will be compared to the existing SLAM Toolbox implementation both from a reliability and computational performance perspective. The SLAM system should still provide accurate results, be equally responsive and use less CPU than the existing method.

Chapter 2

Literature Review

2.1 Simultaneous Localisation and Mapping

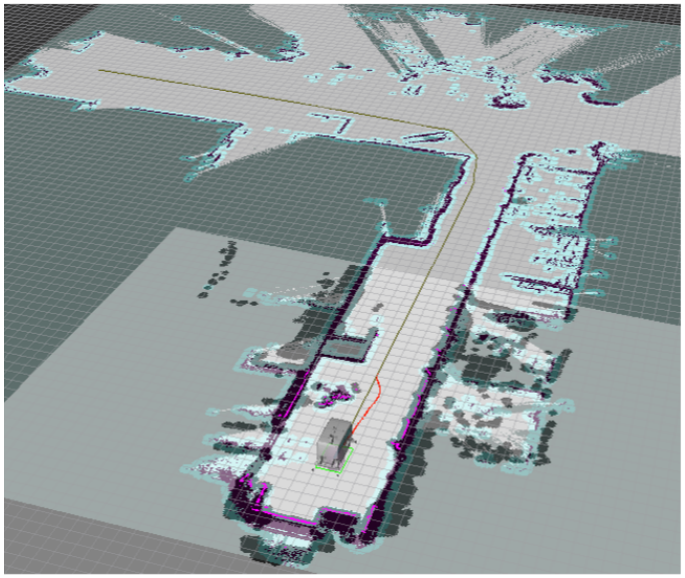


Figure 2.1: RViz display of the nUWAy shuttle bus on a map created using a SLAM algorithm

SLAM is a technique which uses current and past perception sensor data in new environments to position itself and its surroundings in a self generated map. The mapping and localisation elements are co-dependent, and so probabilistic methods are used to update the position and orientation as the map is generated and adjusted.

In robotics, SLAM is primarily a solution to navigate accurately and account for the uncertainty produced by sensors and motors. Without these errors a robot could successfully navigate in a dead reckoning fashion.

Many algorithms exist with varying speed, robustness and use cases. Choosing a SLAM implementation suitable to the available sensors and the environment a robot will be deployed in is key to the robot being able to accurately generate a map and localise within it.

While LiDAR SLAM relies on distinct 3D object shapes to operate, visual SLAM relies on identifiable visual features. This gives each method a natural edge in different circumstances.

For operation in a previously mapped environment most SLAM implementations can be run in a choice of two modes, lifelong mapping or localisation only. Lifelong mapping mode allows for the system to continually update its map and encorporate changes to the environment as they occur which is beneficial for relocalisation in dynamic environments. Localisation only mode foregos this ability to reduce the computation load when navigating in a known environment,

but risks the long term accuracy of the map if the real environment changes.

2.1.1 FAST Corners

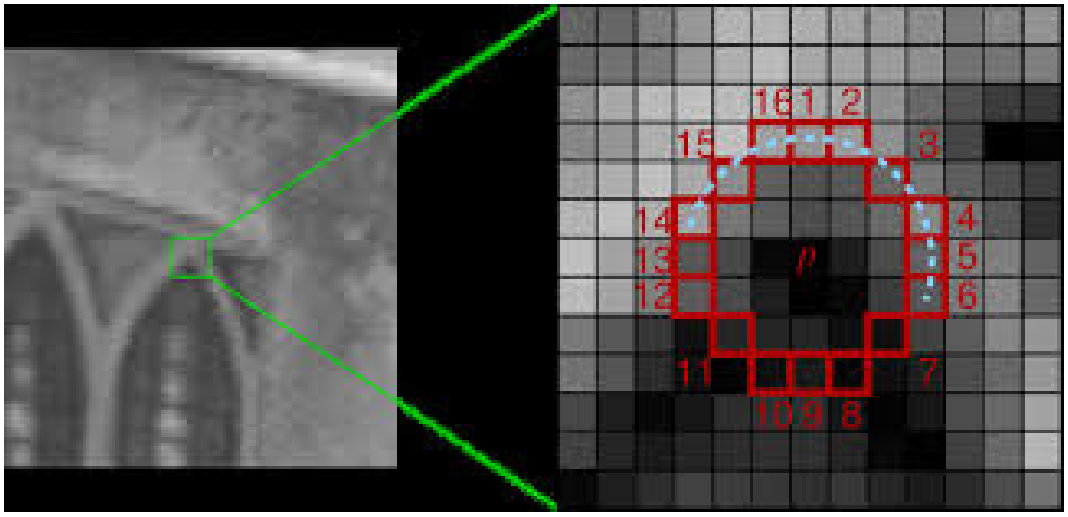


Figure 2.2: FAST Corner detection showing consecutive lighter pixels around a Bresenham circle [9]

FAST Corners is the corner detection technique used by ORB-SLAM. This technique is catered towards real-time applications like SLAM due to it's low computational requirement in comparison to other corner detection algorithms [9]. It also better recognises slightly softer corners such as those which would be obtained from camera data as opposed to pixel perfect computer generated imagery [10].

The technique works by comparing the 16 pixels in a Bresenham circle of radius 3 centred around a point of interest shown in Figure 2.2. If a number n of consecutive pixels are sufficiently brighter or darker than the point of interest then the point is considered a corner. For additional speed, the pixels are not compared consecutively but across diagonals to sooner discern whether a consecutive n would not be possible. Further machine learning optimisation has also been made to the FAST algorithm.

2.1.2 Keyframe

Keyframes are a subset of the image frames received from the camera. Keyframes are selected by a set of criteria which attempts to filter out at an early stage any frames which will not provide useful data for the SLAM result. Formally, the definition from Mur-Artal, Montiel and Tardós is

Criterion 1 Ensures keyframes are created frequently enough to be able to re-localise effectively [11].

Criterion 2 Maximises the compute resources of the mapping thread and prevents the processing of stale data [11].

Criterion 3 Ensures a frame has enough visual features to provide useful information. This will also filter frames which are out of focus or motion blurred [11].

Criterion 4 Ensures each keyframe is "distinct" and will provide new information from other existing keyframes to minimise superfluous data [11].

[10]

2.1.3 Covisibility

Covisibility relates to a mapping of all points which can be viewed in the same frame as a point of interest. Use of a covisibility graph allows for the tracking thread to focus on a local area with reduced, vastly reducing computation time for localisation [11], [12].

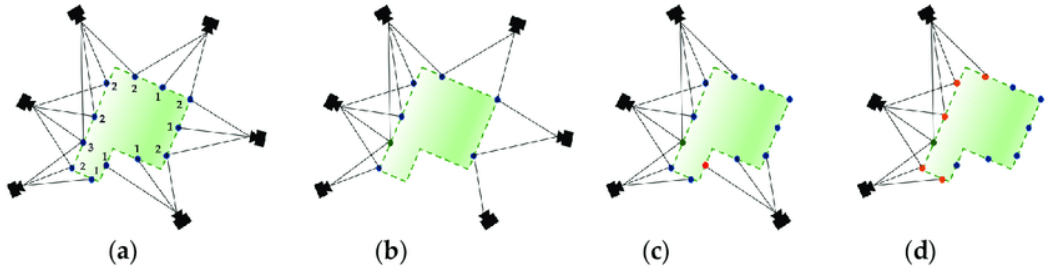


Figure 2.3: Creation of covisiblity graph from keyframe data [13]

The first image shows a set of six keyframes indicated by the cameras, and the keypoints which they can view. The numbers next to each point indicates how many keyframes they are visible in. Points visible in more keyframes are considered to have higher visibility [13].

The second image shows a reduced set of points which are visible in a minimum of two keyframes. This reduces the computation time as points with low visibility will have a lower probability of appearing in newer queried frames [12], [13].

The third image shows that despite two points being physically close they may not be observable together in any keyframe [13].

The fourth image shows in red all the points that are co-visible to the point in green, and the three keyframes associated with the green point [13].

2.1.4 Bundle adjustment

Bundle adjustment is a method of minimising the reprojection error between the real distance value and the reconstructed value when a 3D landscape is reconstructed from 2D image data [14]. It therefore becomes a sum of squares problem formalised in Equation (2.1)

$$\min_{\mathbf{a}_j,\mathbf{b}_i} \sum_{i=1}^{n} \sum_{j=1}^{m} v_{ij} d\left(\mathbf{Q}\left(\mathbf{a}_j, \mathbf{b}_i\right), \mathbf{x}_{ij}\right)^2 \tag{2.1}$$

The 3D points shown as X_1 through X_4 in Figure 2.4 are estimated as 3D map points by the bundle adjustment algorithm, through observations u in the keyframes C_1 through C_3. The estimated 3D X and C points are adjusted to minimise the error between the projection points u and the reprojection points u'.

2.2 Graphics Acceleration

Most modern computer systems have two main processing units, the Central Processing Unit (CPU) and a Graphics Processing Unit (GPU).

The CPU is optimised for sequential computation with a small number of extremely fast cores and larger caching and control flow allocation as shown in Figure 2.5. This means they can quickly execute chains of instructions where inputs may be reliant on previous computations.

GPU's have many more cores than a standard CPU. This means if a task can be broken into smaller independent tasks, they can be solved concurrently using the large number of cores to save time. Computer vision is an area which is easy to parallelise and see large performance

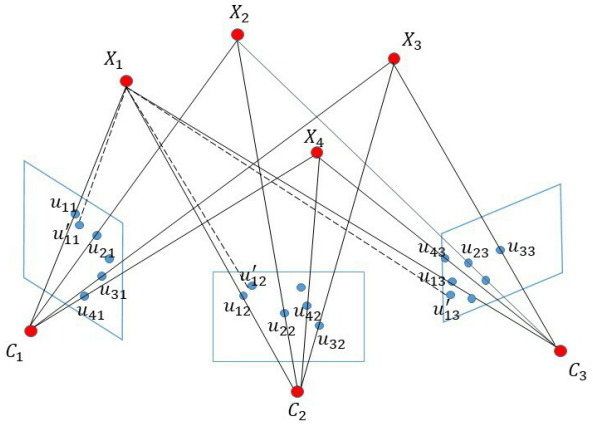


Figure 2.4: A visual display of 3D points captured in mulitple 2D frames

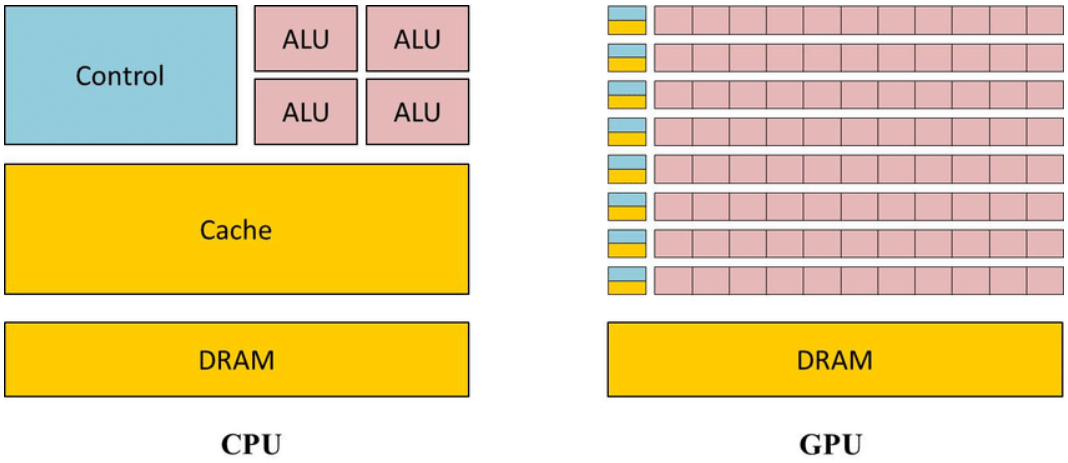


Figure 2.5: A simple diagram showing the physical differences between a CPU and GPU. [15]

increases utilising a GPU as similar subtasks are often run on large sets of pixels or points to convert the image into useful information [16].

Generally for software with GPU utilisation, the overall system structure will be run on the CPU with key computationally intensive and parallelisable blocks of the logic being computed on the GPU. Due to the CPU and GPU having separate memory storage this introduces some overheads to copy memory between these storages before and after GPU computation.

Pulli, Baksheev, Kornyakov *et al.* [16] shows performance increases of up to 700 % using a GPU for stereo vision applications which is similar to the time offset images a monocular visual SLAM implementation will be comparing.

2.2.1 Graphics Processing in SLAM

In vSLAM the tracking thread is the prioritised thread to boost performance due to its time requirement. It's been shown by Aldegheri, Bombieri, Bloisi *et al.* [17] that CUDA accelerated OpenCV can provide performance increases in ORB-SLAM on a NVIDIA Jetson. Additionally, parallelised bundle adjustment has been tested in [18] and [19] with considerable performance increases. In contrast, filtering based methods require an essential stage of resampling [20] which require a sum of importance weights [21]. This results in filtering based methods not being able to be parallelised as efficiently and less capable of GPU utilisation.

Chapter 3

Design Process

3.1 Design Tools

3.1.1 ORB-SLAM

ORB-SLAM is a visual SLAM solution which can be used with monocular, stereo and RGB-D cameras. This will be considered the base system in the report. The basis of ORB-SLAM2 lies in Bundle Adjustment. However, this is a computationally expensive task and cannot be performed for every frame that the camera produces. Because of this, keyframes, discussed in 2.1.2 are utilised, and the SLAM problem is split into three separate asynchronous tasks of tracking, mapping and loop closing [11] as shown in Figure 3.1. This allows the tracking thread to update localisation at an acceptable rate while bundle adjustment can be handled in mapping. It has been shown in [22] that keyframe based analysis presents better results for similar computational cost when compared to filtering methods for visual SLAM [11].

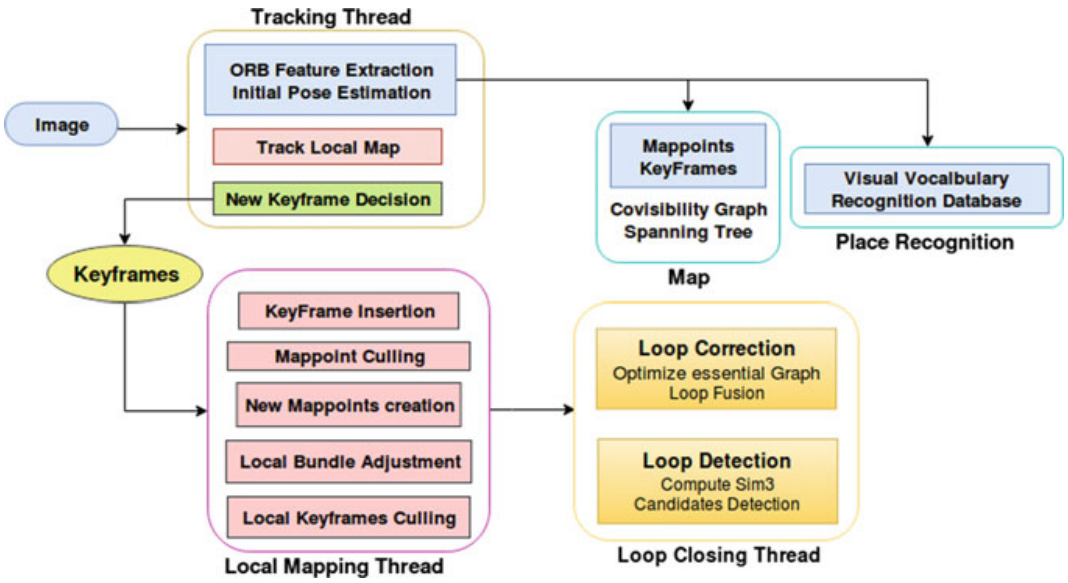


Figure 3.1: A flowchart of the ORB-SLAM2 process [23]

Tracking

The tracking process takes place every camera frame. It firstly extracts features using FAST corners and estimates a pose comparing it to the previous frame [11]. If this estimation fails, the tracking attempts to re-localise using the global map [11]. This estimate is then made more

accurate using a local map before a decision is made as to whether the frame should be used as a keyframe [11].

Being able to run this task every frame makes the localisation as robust and agile as possible.

Mapping

Once it's decided a frame should be added to the keyframes, the covisibility graph explained in 2.1.3 is updated. Map points are culled if they are not visible in the first three keyframes since creation or they no longer appear in enough keyframes due to the local keyframe culling process [11]. New map points are then added from the new keyframe and bundle adjustment is applied. Finally, the aforementioned local keyframe culling removes keyframes considered redundant [11].

Loop closure

The keyframe is compared with all its covisible keyframes through a bag of words model to select frames similar to the current keyframe without being already connected [11]. Transformations accumulated through the error around the loop including translation, rotation and scale are accounted for [11]. If a keyframe is found to be similar enough the covisibility graph is adjusted to essentially merge the two frames into one. Then correction and optimisation are done on previous keyframes [11].

3.1.2 OpenCV

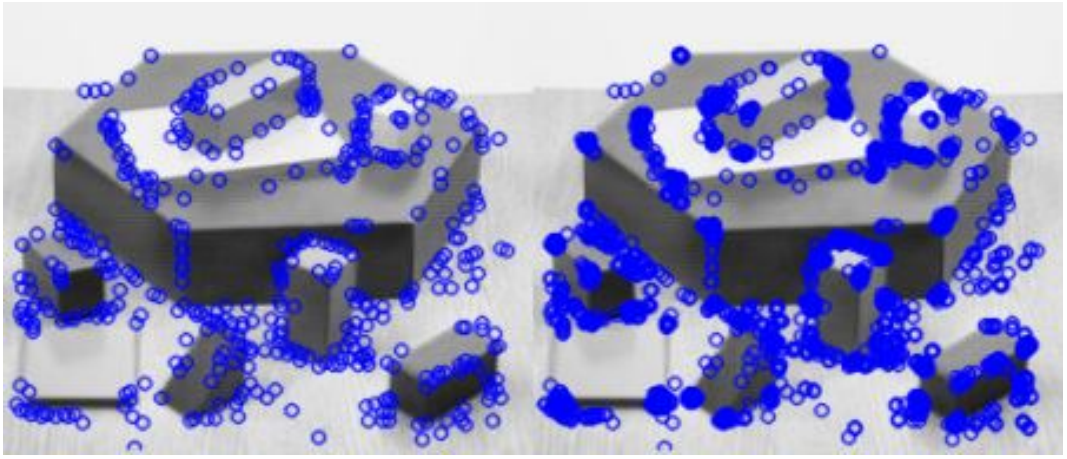


Figure 3.2: OpenCV implementation of FAST Corner detection [24]

OpenCV is an open source software library with over 2500 algorithms, providing a comprehensive set of tools for computer vision [25]. It provides a stable framework for real time vision applications. With its large and active community, as well as compatibility for many languages and platforms, it can be expected OpenCV will be supported well into the future [25]. Open CV tools will be used for various image conversions and processing within the design. Beyond the standard OpenCV libraries is the OpenCV contrib modules, a set of extra in-development libraries which includes libraries which utilise CUDA discussed in Subsection 3.1.3. These libraries will provide vital functionality to the ORB-SLAM2 system for processing input images.

3.1.3 CUDA

CUDA is a general purpose processing platform for NVIDIA GPU's. It's this toolkit which converts tasks into parallel instructions the GPU can understand and execute, as well as ports

the results back to the CPU. Despite being restricted to use on NVIDIA processors, it has better performance than competitors such as OpenCL due to being designed for its specific architecture [26].

CUDA is a general purpose processing platform for NVIDIA GPU's. It's this toolkit which extends the C/C++ language to handle compilation and running on both host (CPU) and device (GPU). CUDA provides functions to handle memory allocation on the GPU memory space, calling of a device executed kernel task with thread management and asynchronous computation, and various libraries to assist with development of device code. CUDA abstracts device code away from the hardware level in a similar way that C++ does, allowing development for any Nvidia hardware without the need to alter code for a different target device. This is beneficial for the split offboard and onboard testing scheme proposed in Section 3.3.

3.1.4 ROS

ROS provides an open-source framework that allows various robotics processes to pass information between one another through a graph network. It's language and platform independence and maintained API's have resulted in adoption from many third party package implementations which are largely open source and available for use. ROS focuses on processes (known as nodes) being largely independent of one another and being coupled at runtime through the use of messages and topics which form the graph network.

3.1.5 Nvidia Jetson AGX Xavier

The Jetson Xavier computer is a development board with an ARM CPU architecture and dedicated Nvidia on board GPU with CUDA support. The computer currently runs an Ubuntu 20.04 operating system and runs much of the buses current autonomous implementation. This will be known as the onboard computer in the report.

3.1.6 Graphics workstation

The graphics workstation is an offboard system and will be the system used for initial development and testing. It's system architecture differs from the Xavier computer as it has an Intel x86 CPU and two Nvidia Titan X graphics cards. Key compatibility criteria are met however as these graphics cards have CUDA support which will allow testing of the design utilising graphics processing. Due to CUDA's hardware abstraction philosophy the implementation of the design will not differ despite these differences in physical hardware. This computer will be referred to as the offboard computer in the report.

3.1.7 Cameras

Two camera solutions were considered for this project, firstly being the Pointgrey GS3-PGE-23S6M-C monochrome cameras already installed in the bus, and secondly being a colour variant also produced by Pointgrey. Ultimately for this project images will be converted to greyscale for ORB extraction so the benefits of colour images were dismissed in favour of the existing architecture. The cameras communicate over the network to a computer with the Pointgrey Spinnaker drivers installed.

3.2 Evaluation Criteria

The evaluation criteria were designed to link the testing conducted to the aims of the project. The success of this project will ultimately be determined by the ability to lower the load on

the CPU, as this result will successfully reduce the extent of bottlenecking of the system. Furthermore, the design must produce comparable or improved mapping and localisation accuracy. Finally, any performance improvements generated from the design over the base ORB-SLAM2 system wish to be highlighted. With these requirements in mind the following evaluation criteria were devised.

3.2.1 Computation consumption

Monitoring and logging of process ID's during testing will reveal the load on both CPU and GPU due to the design. This will be accomplished by recording the output of the `top` and `nvidia-smi` commands for CPU and GPU usage respectively at 1 s intervals. The CPU data is represented as a percentage usage of a single core. As the CPU is comprised of multiple cores this can exceed 100 %.

3.2.2 Pointcloud evaluation

Pointclouds between base system and the design will be compared qualitatively to ensure that the design produces a pointcloud map that is of equal or better quality then the base system. These pointclouds will be visualised in RViz software and manually analysed.

3.2.3 Time

The computation time of the affected sub processes in the ORB-SLAM2 algorithm will be measured and compared between CPU and GPU executed variants The time taken for the bundle adjustment and FAST corners sections of the ORB-SLAM2 algorithm to compute will be measured using the `chrono` library will measure the total real time taken for computation to complete, and the output will be written to a file for analysis. While improvements to the timing of the algorithm at this point is not the primary objective of the project, the metric is important in ensuring the real time operation of the algorithm for its use as a SLAM implementation on the bus.

3.3 Method

Testing will follow a structured, stage based approach, with defined gateway criteria to warrant progression to the next stage. Trialling will be tracked in a spreadsheet, with results files named in adherence to a strict convention to ensure integrity of the data.

The design will initially be installed, debugged and tested on the offboard computer. This provides a more stable and controllable environment, free from the rest of the code run on the bus which may cause unpredictable adversities during development. Additionally, its remote availability allows for uninterrupted access which will be beneficial throughout the early phases of the project.

The end goal is to have the design installed on the onboard Jetson computer. This would go further to proving the projects viability but comes with some constraints involved with the handling of the other critical live code on this system.

3.3.1 Stage 1: Pretesting

The first stage will consist of running a set of tests from publicly available odometry benchmarks. These large datasets of high quality test scenarios are aimed to encompass the large majority of typical scenarios and should provide a rigorous criterion which can easily be compared to the results of the base ORB-SLAM2 system.

The EuRoC dataset [27] will be used to test the installation on a scenario with which the base system is known to work from previous studies [1]. This test will confirm that the system is fully installed and operational, and successfully produces a pointcloud output from the given input. This dataset was selected due to the provision of direct rosbag download of the data, further reducing sources of error from external conversion means. This dataset also provides the camera calibration values required to correct for the images intrinsic distortions.

Pretesting will then be run on the three campus trials outlined in Section 3.4 in order to assure that the system can be successfully applied in the scenarios the nUWAy shuttle bus will be faced with.

In order to complete this stage of testing and move to stage 2, the system must be fully operational, accepting incoming image data and producing a viewable pointcloud output which is representative of each test environment.

3.3.2 Stage 2: Offboard trials

The offboard trials will test the base ORB-SLAM2 implementation against the design on a graphics workstation. Timing and computation consumption data will be recorded for analysis, and the pointclouds produced by these trials will be compared to affirm that the design can produce results of equal quality to the base system.

In order to complete this stage of testing and move to stage 3, the system must display comparable or improved pointcloud output to the pretesting results and display a reduced CPU load in comparison to the base system.

3.3.3 Stage 3: Onboard trials

Onboard trials will run on the nUWAy shuttle buses Jetson Xavier computer and test the performance and operability of the base system and the design. This will ultimately show the percentage reduction in CPU demand in the final target computing environment. Testing will progress to stage 4 following the design demonstrating its ability to run on the bus hardware at an acceptable CPU demand.

3.3.4 Stage 4: Live trials

The final stage of trials will test the design using a live video stream from the buses onboard cameras while in motion, with other elements of the autonomous stack running. This will gauge real performance and reliability on the bus in operation and be the basis of comparison to the existing solution.

3.4 Trial selection

The campus environment was considered to consist of areas of urban environment, including buildings, posts and pathways, and more organic environments, consisting mostly of vegetation. For the SLAM algorithm to be adequate for use on campus it must be able to successfully maintain its mapping ability in both of these environments. Each trial was assigned a short form letter code denoted in brackets which will be used to identify the trials throughout the report.

3.4.1 Oceans building trial (OC)

The first trial will take place on a stretch of road near the UWA Oceans Institute shown in Figure 3.3. This area with wide, unobstructed road and prominent buildings on both sides provides an outdoor urban environment scenario. It's expected that the algorithm will be able to handle this scenario well due to the geometric and static corners present.

Figure 3.3: Path of oceans building trial

3.4.2 Social science to mechanical building trial (SSM)

The social science building provides an area with an urban environment on one side of the thoroughfare and an open area on the other side. The social science area is one identified to be of high importance. It is both an area where we have seen the most reliable localisation on the buses current implementation and an area which is used for the buses operational drives. The route then takes a 90 degree turn and heads toward the campuses mechanical engineering building. This tests a section of route with predominantly vegetation and a lack of defined corners provided from buildings. The route plan can be seen in Figure 3.4

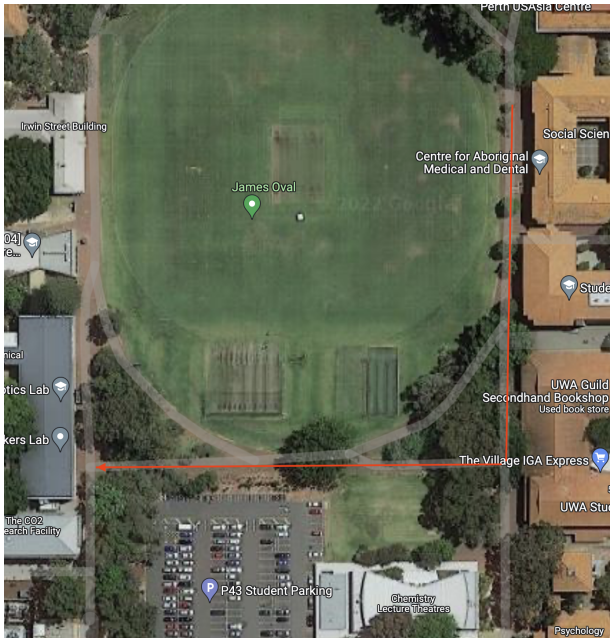


Figure 3.4: Path of social science to mechanical building trial

3.4.3 Reid to childcare building trial (CC)

Finally, a longer testing route from Reid library to the childcare centre was recorded shown by Figure 3.5. This route forms the majority of the route intended for the buses operational drives to transport students from the Reid library to the UWA business school. It provides a mixture of urban and organic environments and will test the ability of the algorithm over a longer distance. In the current bus implementation, mapping and localisation is still not something that can be considered completely robust over this long distance. For this reason, promising results in this trial will be a key driver to warrant replacement of the current implementation.

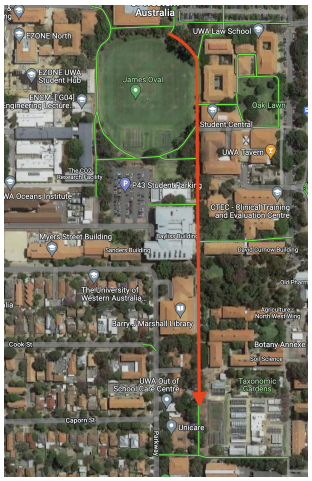


Figure 3.5: Path of Reid library to childcare building trial

Chapter 4

Final Design

The final design combines the techniques of CUDA ORB extraction with bundle adjustment optimisation also using CUDA, and runs as a node on the ROS system. By combining these techniques it will achieve the highest proportion of the ORB-SLAM2 algorithm run on GPU publicly available. Supporting framework was designed to allow this system to function on the nUWAy shuttle bus.

4.1 System Architecture

Figure 4.1 displays the full ROS system architecture including the image acquisition methods for both testing and execution and the sub components with CUDA acceleration displayed in green.

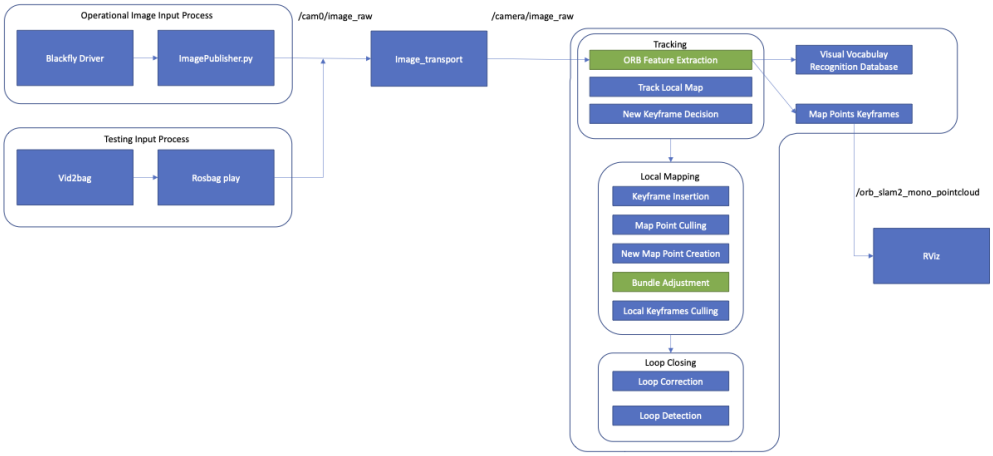


Figure 4.1: Modified system architecture diagram

4.1.1 ORB extraction

The design will consist of the works of Nguyen [28] utilising OpenCV with CUDA acceleration techniques to improve the speed of FAST corner detection and ORB extraction in the tracking thread of the ORB-SLAM2 algorithm. Due to the ORB extraction process being executed in the tracking thread, it is therefore run at a high frequency and therefore is a key process to be implemented on the GPU in order to reduce usage on the CPU.

4.1.2 Bundle adjustment

The modified bundle adjustment stage removes the usage of the g2o sparse optimiser class and replaces it with the CUDA bundle adjustment class from the works of Adaskit-Team [29]. This class is designed to function in a similar way to the g2o sparse optimiser such that the process flow through addition of keyframe and mappoint data to the optimiser before the optimisation algorithm is the same, with familiar naming convention. This data is however eventually being copied across to the graphics card memory where the optimisation algorithm will run.

Attempts were made to further extend this process fully through to the local bundle adjustment algorithm with only limited success. Ultimately a product which computes the full process on the GPU was not able to be delivered, and further development would be required to complete this.

4.1.3 VideoToRosbag.py converter

For ease of acquisition and reproducibility of the tests, recorded footage on the bus was done within the SpinView camera driver software and saved as a video file. In order to convert this file into a image stream on a ROS topic to be read in by the design, a python script was made utilising OpenCV to create a bagfile from a video file. This bag file can then be replayed multiple times to conduct trials and testing. The produced bagfile would publish the Image messages to the same topic as the cameras on the bus, essentially mimicking their behaviour to the eye of the ROS system. This method allowed easier viewing of recorded data immediately after recording to ensure its of good quality for trial purposes over directly recording a rosbag on the bus.

4.1.4 Image publisher

The image publisher acts as an interface between the blackfly camera drivers and ROS. It creates a node in the ROS environment and begins querying the camera for images before converting them into a ROS message and publishing them to a topic. This topic is either directly subscribed to by the design or is remapped using an `image_transport` node such that the image data gets delivered to the SLAM system.

4.2 Camera Calibration

Each lens has unique optical characteristics which are imperfect. These create subtle distortions in the image which must be corrected before processing which predominantly include barrel or pincushion distortion, an effective change of magnification dependent on the distance from the centre of the lens focus point. These distortions can be corrected with with a set of parameters obtained through a calibration process using a checkerboard pattern shown in Figure 4.2. This process was carried out using the ROS `image_pipeline` package. The final calibration results used in testing are as in Figure 4.1

4.3 Package Installation

Throughout the installation and testing process, issues were experienced with competing versions of OpenCV, Eigen, Python and importantly CUDA. These issues are discussed further in Section 6.2. Ultimately the final versions of these packages that were found to work in conjunction with each other and used in testing are shown in Table 4.2

It was revealed that the installation of the CUDA software on the Jetson Xavier was different to that on desktop PC's and required a complete system wipe which was deemed unfeasible due to the significant impact to the buses operability it would have. Despite this, ROS 1 was

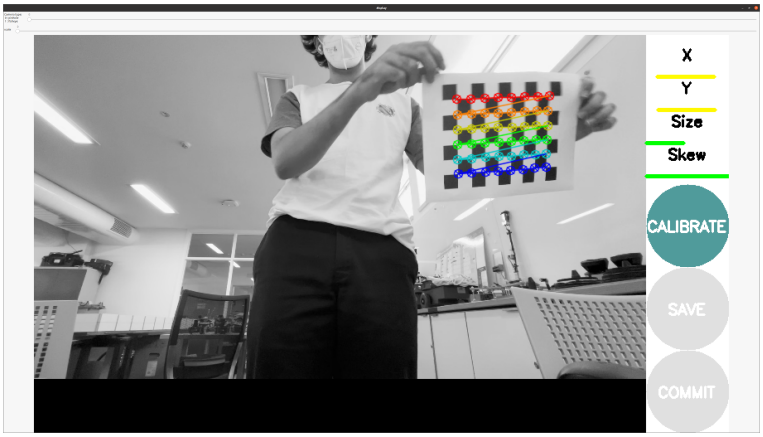


Figure 4.2: Calibration process using a checkerboard pattern

Table 4.1: Camera calibration parameters

Parameter	Result
fx	896.893463
fy	886.578382
cx	908.242571
cy	503.797799
k1	-0.013504
k2	-0.002980
p1	-0.008882
p2	-0.016219
k3	0

Table 4.2: Package versions for workstation

Package Name	Version
CUDA	11.6
CUDA Samples	11.6
Nvidia Driver	515.48.07
Eigen	3.4.0
OpenCV	4.5.5
OpenCV contrib	4.5.5
Pangolin	0.8

successfully reinstalled onto the Jetson Xavier and following this the base system was installed to test the operation of the ORB-SLAM2 implementation on the final hardware.

Both computers had severe storage space limitations on their respective primary drives. Due to this, some packages had to be installed on secondary drives with the utilisation of symlinks to meet the space requirements of the system and its dependent packages. This process was identified as a risk for long term reliability of the system as it creates additional vulnerabilities to files being moved and no longer accessible.

4.4 Bandwidth limitations

During development of the image publisher it was found that the image stream from the camera required a very large amount of bandwidth and was choking the buses network when image

acquisition was in progress at high frame rates and resolutions. This then caused issues for other devices trying to communicate with each other which rendered the bus unreliable to drive autonomously.

An immediate fix was pursued to rectify this issue by lowering the frame rate and resolution of images acquired from the camera. This proved effective, however was not desirable as a long term solution as both these compromises cause issues for the design and any future camera based processing projects. A lowered resolution results in poorer quality ORB extraction leading to longer initialising times and a higher chance to lose tracking while in motion. A lowered frame rate contradicts the aim of this project to improve to performance of the tracking thread and achieve processing of a higher frame rate in this thread.

To create a more practical long term solution the cameras were tested on a separate network, by utilising the USB-C port on the Jetson Xavier computer and a USB-C to Ethernet adapter. This allowed a direct and isolated secondary network such that the cameras could stream images at higher resolution and frame rates without impacting the main network on the bus. After assignment of IP addresses and adjustment of IP settings this technique ultimately saw success. In its current form this will only allow the use of a single camera, which is adequate for the purposes of this project, but a network switch can be implemented to solve this problem and run both cameras simultaneously if funding is acquired for its purchase.

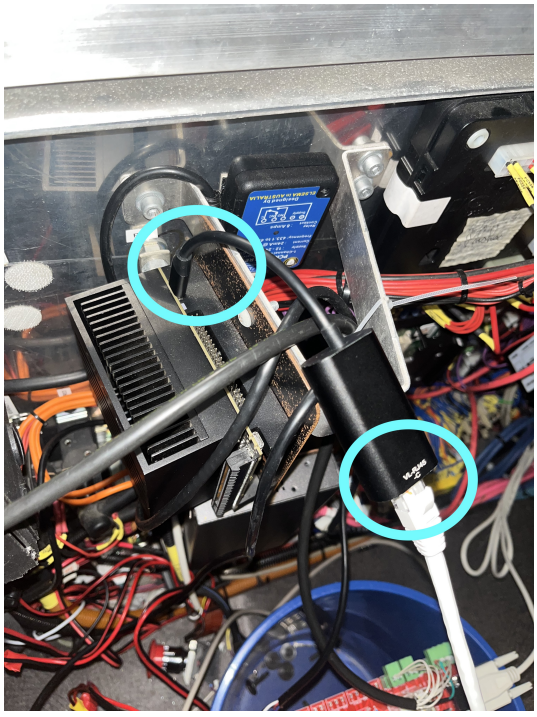


Figure 4.3: USB-C Ethernet converter utilised as a second network connection for the Jetson Xavier direct to Pointgrey camera

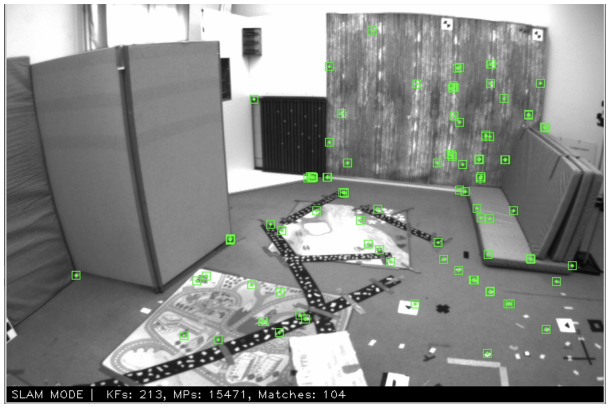
Chapter 5

Results

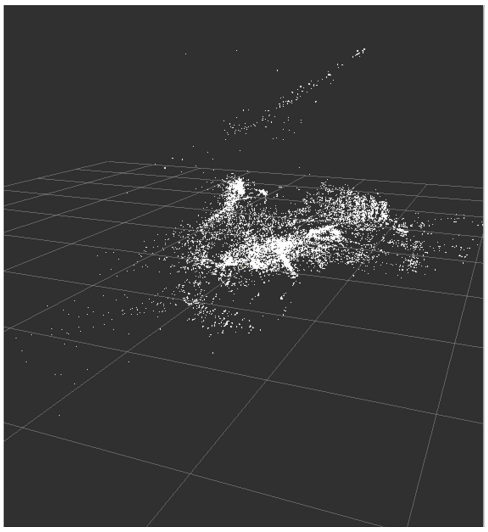
5.1 Stage 1: Pretesting

Pretesting was conducted firstly on EuRoC dataset examples followed by the three campus tests. This will confirm proper operation and suitability of the ORB-SLAM2 algorithm in the UWA campus environment.

5.1.1 EuRoC V1 trial



(a) Successful ORB extraction

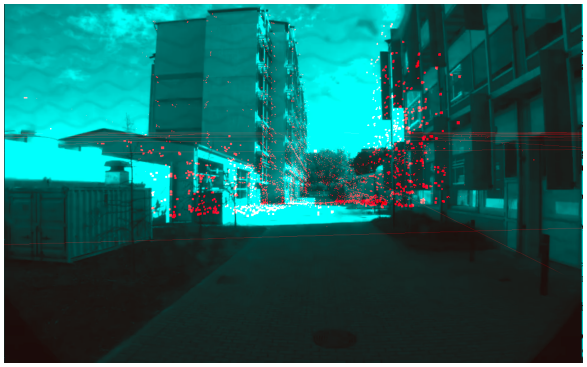


(b) Generated Pointcloud

Figure 5.1: Pretesting run on the EuRoC V1 dataset

The key purpose of the EuRoC test was to initially test the ORB-SLAM2 algorithm on a dataset it was known to be effective on, in order to confirm that the installation process was complete and that there is full operation of the system. From the display window shown in Figure 5.1a we can see the image stream going through to the system, as well as display of the extracted ORB keypoints and other information about the current state of the system. It also confirms that we can retrieve a generated pointcloud and display this in the RViz software as shown in Figure 5.1b. This confirms correct operation of the ORB-SLAM2 system and allow progression to verification of its applicability to the campus's environment.

(a) Generated Pointcloud



(b) Overlaid pointcloud on raw trial image

Figure 5.2: Pretesting run on the oceans trial

5.1.2 OC trial

The system quickly initialised and produced a high quality point cloud, with the prominent building faces to each side of the camera easily distinguishable. This is shown in Figure 5.2a, with the same pointcloud from a differing perspective overlaid on a frame in Figure 5.2b. The foreground buildings on either side are detected and the system performed particularly accurate point placement of the background building. Slight differences between the pointcloud and underlaid image can be explained due to the underlying image not being adjusted for distortion and perspective difference between the camera and RViz.

5.1.3 SSM trial

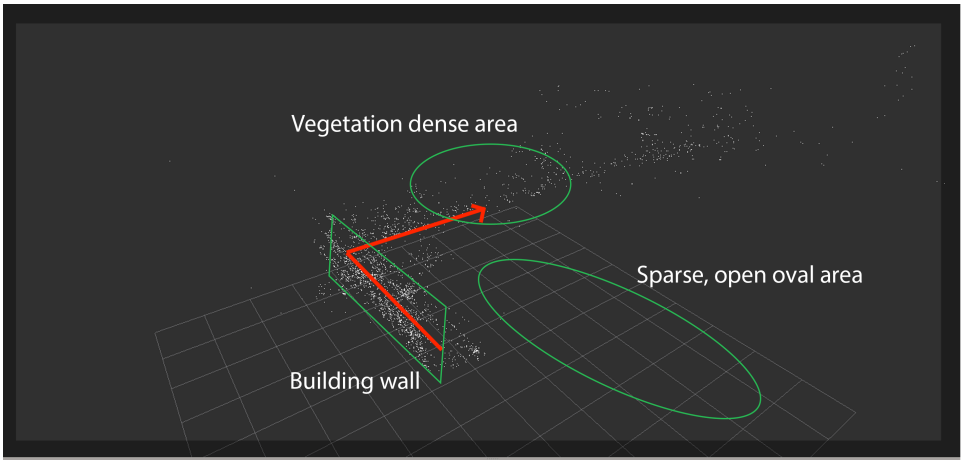


Figure 5.3: Annotated RViz output of SSM trial testing

The system dealt well with the challenges of the SSM trial, successfully initialising despite a lack of considerable geometry on one side of the camera. It also adequately displayed the right angle corner taken and continued to be able to map and localise in a more vegetation based environment. These characteristics are captured in the annotation in Figure 5.3.

5.1.4 CC trial

The system continued to produce pleasing results even in a longer length trial with key landmarks on the journey adequately depictable from the RViz pointcloud output. Figure 5.4

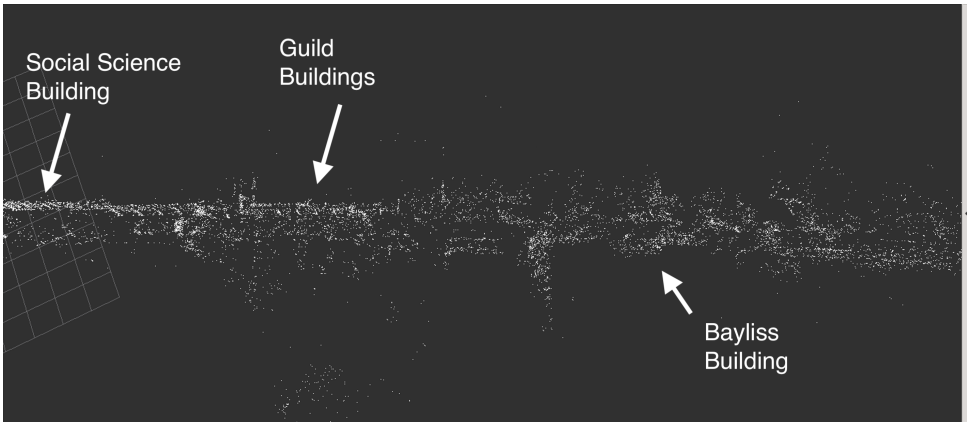


Figure 5.4: Annotated RViz output of CC trial testing

displays some of the various buildings visible from the pointcloud. The pointcloud maintains it's expected straight line tracking all the way through to the end of the path.

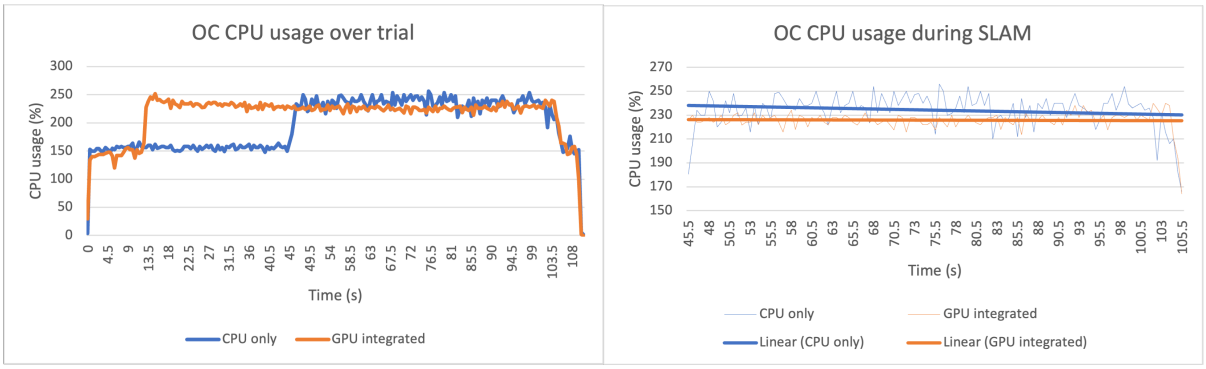
5.1.5 Pretesting summary

These situation based results give confidence to proceed with the GPU integration and testing. All three campus trials displayed results that indicated ORB-SLAM2 as a suitable SLAM solution for the campus environment. This meets the gateway criteria outlined in 3.3.1 to continue into GPU development and offboard testing.

5.2 Stage 2: Offboard Testing

The offboard testing is designed to observe the benefit of the CUDA integrated processes of the ORB-SLAM system, reducing the CPU usage as well as time benefits. This testing will provide important indicative data of these benefits on real campus situations which can be used to make the case for implementation on the operational Xavier computer.

5.2.1 OC trial



(a) CPU usage of offboard OC trial (b) CPU usage of offboard OC trial during SLAM

Figure 5.5: Offboard CPU usage of the oceans trial

In the oceans building test we see in Figure 5.5a that the design trial initialised significantly faster indicated by the earlier rise in CPU usage as the full computational process of the system becomes operational. This is also reflected in the pointcloud in Figure 5.10, which displays

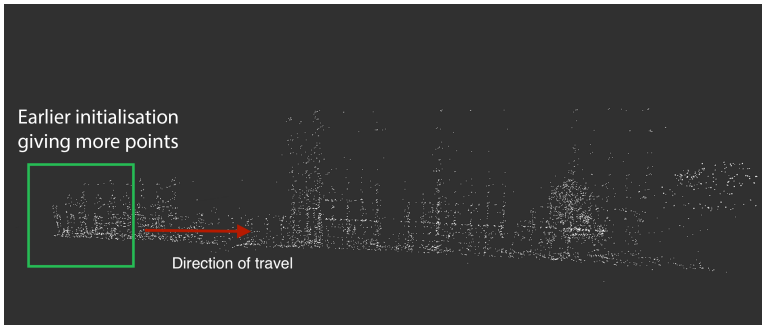
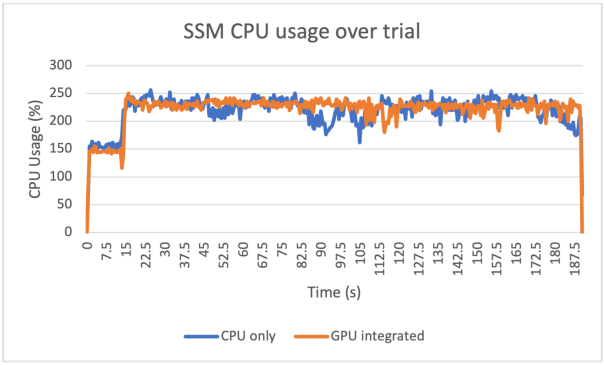


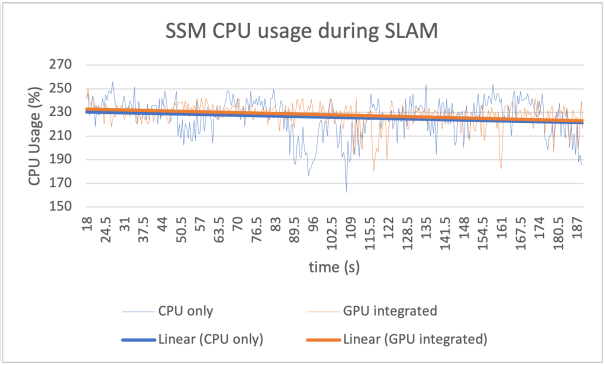
Figure 5.6: Pointcloud output of design run on OC trial

points in an earlier period of the trial than the CPU test shown in Figure 5.2a. The period where both systems are successfully mapping is shown in 5.5b. The displayed average values clearly show the reduction in the CPU usage by the design over the base ORB-SLAM2 system. This was calculated to be a 8.44 % raw reduction resulting in a 3.60 % decrease in CPU usage.

5.2.2 SSM trial



(a) CPU usage of offboard SSM trial



(b) CPU usage of offboard SSM trial during SLAM

Figure 5.7: Offboard CPU usage of the social science to mechanical building trial

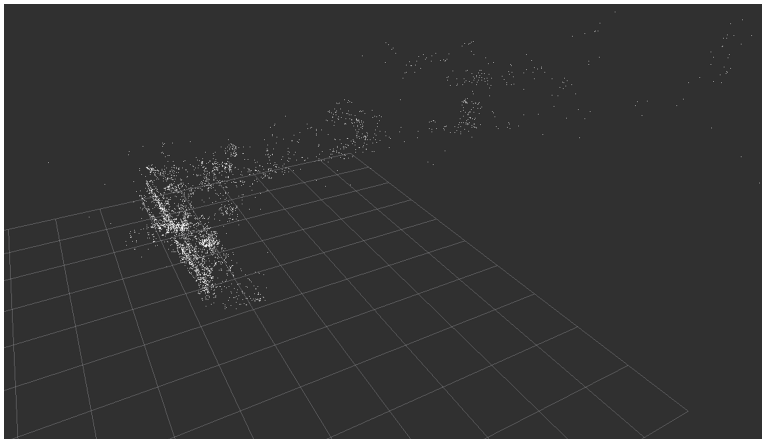
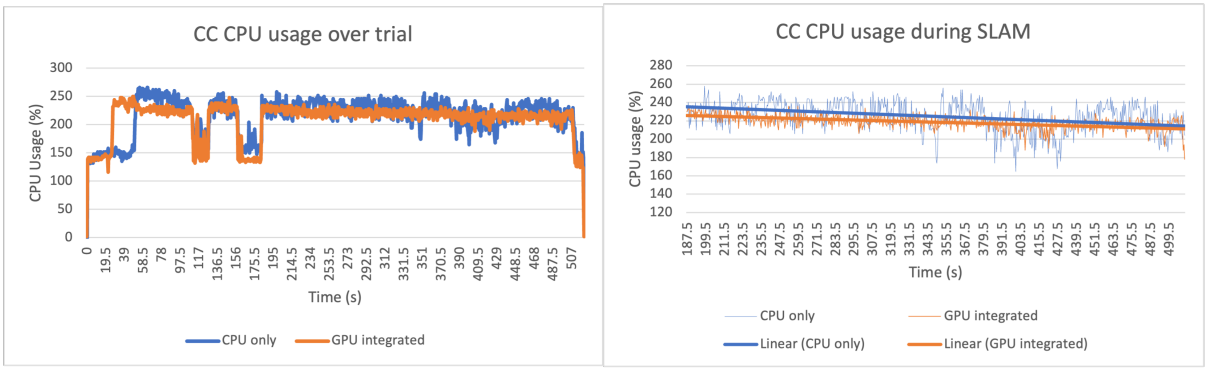


Figure 5.8: Pointcloud output of design run on SSM trial

The results of the social science to mechanical building trial were far more correlated to the base system. There is no additional delay initialising that was seen in Section 5.2.1 and the CPU usage is far closer to each other at 2.06 % raw difference resulting in 0.91 % increase. The

slightly less CPU usage in the base system is suspected to be caused by an unexplained dip in usage through the middle of the test seen in Figure 5.7, which is not echoed in the test of the design. Once again pointcloud output is highly comparable, its output in Figure 5.10 having the same key characteristics outlined in Section 5.1.3.

5.2.3 CC trial



(a) CPU usage of offboard CC trial

(b) CPU usage of offboard CC trial during SLAM

Figure 5.9: Offboard CPU usage of the Reid to childcare building trial

Figure 5.10: Pointcloud output of design run on CC trial

Show characteristics similar to those found in Section 5.2.1 with faster initialisation by the design and a lower CPU usage at 6.22 % raw difference or 2.77 % decrease. The two dips in CPU usage at the 200 and 325 second marks were identified as being due to two emergency stop procedures taken during the route. Using this extended journey and observing the flat to decreasing trend of the CPU usage in 5.13c we can determine that the CPU usage from the system will not increase as journey length and map size increases and hence these factors should not be a concern when operating on the bus.

5.2.4 Timing

The timing data will assess the performance value of the design over the base ORB-SLAM2 system, isolating the ORB extraction and bundle adjustment algorithms which have been modified.

Its expected that ORB extraction time is mostly reliant on the number of features to extract per frame and therefore proportional to the number of features parameter set. As this parameter was kept constant throughout all tests we don't expect drastic change in the ORB extraction time between the different trials. Tests on the design shown in Figure 5.11 reflect this theory with averages only ranging by 9 % while the base system tests show a larger variation of up to

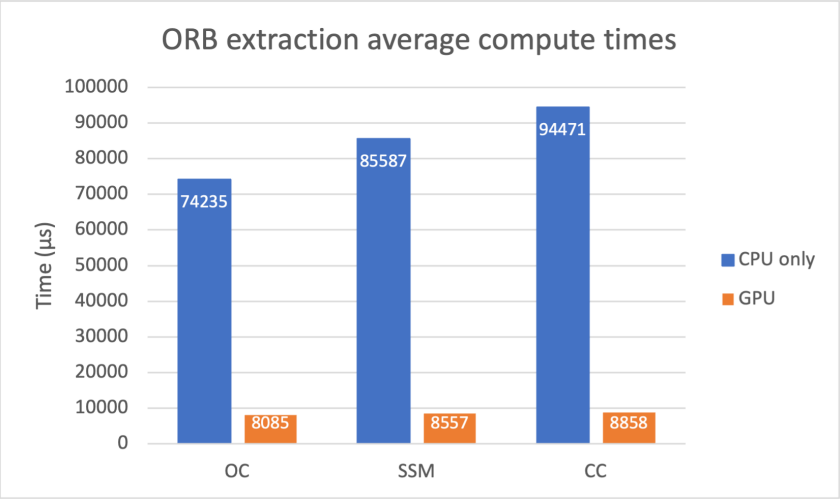


Figure 5.11: ORB Extraction timing data for base system and design across trials

27 %. We see a reduction in the compute time of the ORB Extraction algorithm by a factor of about 10x across the three trials of the design which is an impressive speed difference compared to the base system.

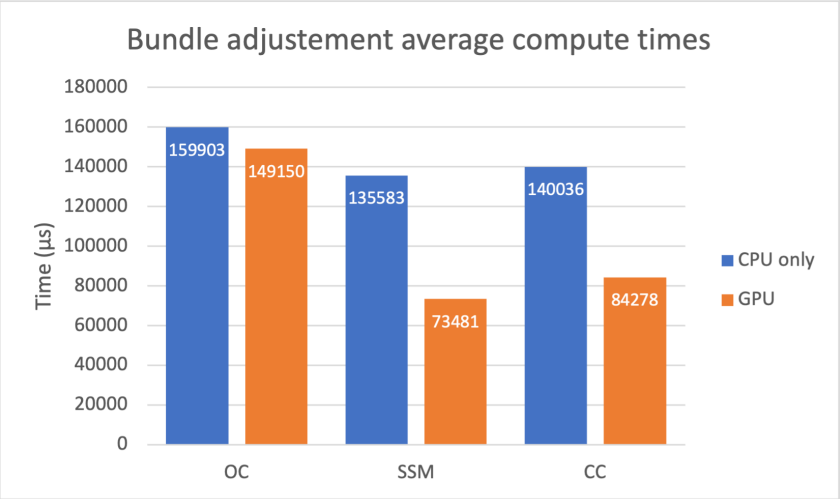


Figure 5.12: Bundle Adjustment timing data for base system and design across trials

Timing results of bundle adjustment shown in Figure 5.12 display substantial reduction by 40 % to 45 % in the SSM and CC trials and a marginal reduction in the OC trial. Variation in these trials is much more expected as the bundle adjustment calculations change dependent on the unique map point and keyframe locations in each trial. This indicates that the design, with partial utilisation of the GPU, is able to reduce the time spent on the bundle adjustment algorithm. Further discussion is posed in 6.1.

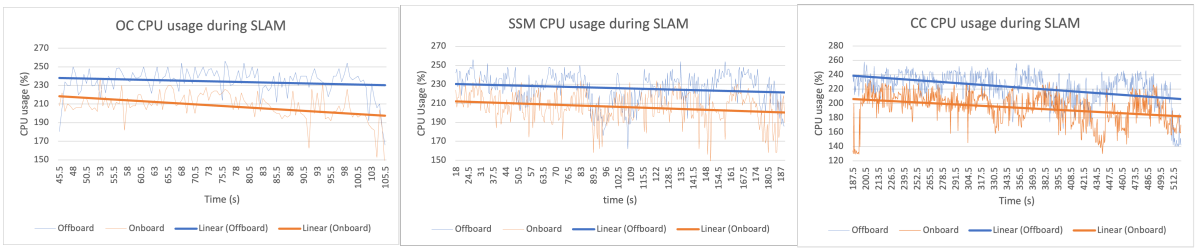
5.2.5 Offboard testing summary

Offboard testing confirmed that the design meets the pointcloud quality and timing criteria outlined in Section 3.3.2. While the CPU load was reduced in two out of three results, the difference was not large enough to be considered a success and formally move to stage 3 testing. The design will need further refinement and expansion to handle the full local bundle adjustment algorithm on the GPU to see considerable decrease in CPU usage and complete these

requirements.

5.3 Onboard Testing

The onboard testing was only able to be conducted using the base system but gives a basis for comparison between the performance of the two computer systems such that some predictions may be able to be drawn of the performance of the design if implemented on this system.



(a) OC trial during SLAM (b) SSM trial during SLAM (c) CC trial during SLAM

Figure 5.13: Comparison of offboard and onboard computers running base ORB-SLAM2 across campus trials

The CPU consumption data shows that across the three trials the Xavier Jetson uses on average 11 % less than the offboard computer. Initialisation time matched that of the offboard base system trials, indicating that there was no difference in the algorithms ability to process the image data and operate all components of the system.

5.3.1 Onboard testing summary

The preliminary onboard testing that was able to be conducted provides evidence that the Xavier Jetson has a somewhat more capable CPU and as such should provide better performance then the offboard system for all the components of the design which do not utilise the GPU. This is in practice offset by the other systems that will be running in the autonomous stack when the bus is in operation which is yet to be tested until the gateway criteria of Stage 2 testing is met.

Chapter 6

Discussion and Suggestions

6.1 Knock on effects of recording real timing data

The timing data acquired is representative of the real time between the start of the process and the completion. This differs from the actual time spent processing as each CPU core is constantly multitasking between processes in real time. Understanding this, every effort was made to ensure the computer was in a uniform state for each test, closing down any remote desktop connections and other non essential tasks during testing. Some of the implications were unavoidable, however, and it's predicted that a portion of the time reductions in the bundle adjustment data will be as a result of the reduced CPU load from the ORB Extraction, and hence less time spent processing the ORB extraction while the local bundle adjustment process is in progress. The real time had to be measured in order to get constant data when tasks were split between the CPU and GPU.

6.2 Testing on unclean systems

Both computer systems were considered to be unclean environments with which the project was to be implemented on. This goes against available advice which is to install and test code on a freshly installed operating system, to ensure the latest versions of software and remove the possibility of duplicate or modified packages, or remaining remnants of uninstalled packages disrupting the compilation of newly installed software. Unfortunately due to the operational nature of the Xavier PC and the shared use and configuration of the testing workstation, a clean environment was not achievable for either of these systems. Installation on an unclean environment can cause an array of issues as various other packages and dependencies are out of date or installed in unlikely and problematic locations. While on the graphics workstation is not frequently used by other personnel, installation on the Jetson Xavier computer will need to be carefully assessed to ensure that modification is not made which affects external code on the computer used for the current operation of the bus. Due to the many interdependent packages on the pc this can happen quite easily without immediately being aware. Such small changes can never the less cause a great deal of unexpected downtime and debugging hours later on.

6.3 Separation of operational and development hardware

With the recent addition of a second shuttle bus and potential acquisition of new Nvidia hardware, had this project been run at a later date where such hardware was available with a clean environment, its expected that difficulties faced during the installation process would have been significantly reduced by testing on a new machine in a clean system environment. Such a reality would also mean there would be no need for stage two testing as easy and accessible testing and

debugging could be done directly on the intended target machine. With two shuttle buses the testing on the bus would not have had to have been foregoed in favour of keeping the bus in an operational state, as testing which would incur large downtime could be conducted on the second bus.

6.4 Extension to full local bundle adjustment on GPU

The extension of the techniques used by Adaskit-Team to compute local bundle adjustment on the GPU was partially produced but not able to be completed. Unrefined knowledge of the C++ programming language was the main element that prevented execution of this extension in a timely manner. The key hurdle lay in the robust kernel used in local bundle adjustment which could be overlooked during global bundle adjustment in Adaskit-Team's method [29], but was required to produce an accurate pointcloud while undertaking local bundle adjustment. It's suggested that such a task be engaged by an individual with stronger programming background and experience in C++.

Chapter 7

Conclusion

The operational uptime requirements of the bus and alternative developments ultimately hindered this projects ability to proceed to the execution phase but the design passed some the constructed milestones to determine its feasibility up until this point. With further development of the code to extend the graphics integration to the whole of the local bundle adjustment process it is expected we will see results which further reduce CPU demand of the system. This, in combination with the improved CPU results seen on the onboard system provide promise that a CUDA integrated ORB-SLAM implementation is a feasible solution which may provide simultaneous localisation and mapping results without overburdening the computation systems available on the nUWAy shuttle bus.

Bibliography

[1] R. Mur-Artal and J. D. Tardos, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017. DOI: 10.1109/TRO.2017.2705103.

[2] V. der Automobilindustrie, "Lane keeping assist systems," *Safety and Standards - German Association of the Automotive Industry*, [Online]. Available: https://www.vda.de/en/topics/safety-and-standards/lkas/lane-keeping-assist-systems.html.

[3] V. Demuynck, "How do hd maps extend the vision of autonomous vehicles?" *TomTom*, 2020. [Online]. Available: https://www.tomtom.com/blog/automated-driving/hd-maps-vision-autonomous-driving/.

[4] O.-R. A. D. (committee, *Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles*, Apr. 2021. DOI: https://doi.org/10.4271/J3016_202104. [Online]. Available: https://doi.org/10.4271/J3016_202104.

[5] G. Sharabok, "Why tesla won't use lidar," *Towards Data Science*, 2020. [Online]. Available: https://towardsdatascience.com/why-tesla-wont-use-lidar-57c325ae2ed5.

[6] E. Chan, "Lidar vs. camera only – what is the best sensor suite combination for full autonomous driving?" *Mirae Asset Global Investments (Hong Kong) Limited.*, 2021. [Online]. Available: https://www.am.miraeasset.com.hk/insight/lidar-vs-camera-only-what-is-the-best-sensor-suite-combination-for-full-autonomous-driving/.

[7] V. O. M. Team, "What is visual slam technology and what is it used for?" *Association for Advancing Automation (A3)*, 2018. [Online]. Available: https://www.automate.org/blogs/what-is-visual-slam-technology-and-what-is-it-used-for.

[8] F. Huang, H. Yang, X. Tan, S. Peng, J. Tao and S. Peng, "Fast reconstruction of 3d point cloud model using visual slam on embedded uav development platform," *Remote Sensing*, vol. 12, no. 20, 2020, ISSN: 2072-4292. DOI: 10.3390/rs12203308. [Online]. Available: https://www.mdpi.com/2072-4292/12/20/3308.

[9] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof and A. Pinz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443, ISBN: 978-3-540-33833-8.

[10] D. Tyagi, "Introduction to fast (features from accelerated segment test)," 2019. [Online]. Available: https://medium.com/data-breach/introduction-to-fast-features-from-accelerated-segment-test-4ed33dde6d65.

[11] R. Mur-Artal, J. M. M. Montiel and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015. DOI: 10.1109/TRO.2015.2463671.

[12] Y. Li, N. Snavely and D. P. Huttenlocher, "Location recognition using prioritized feature matching," in *Computer Vision – ECCV 2010*, K. Daniilidis, P. Maragos and N. Paragios, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 791–804, ISBN: 978-3-642-15552-9.

[13] Y. Zhou, X. Zheng, R. Chen, X. Hanjiang and S. Guo, "Image-based localization aided indoor pedestrian trajectory estimation using smartphones," *Sensors*, vol. 18, p. 258, Jan. 2018. DOI: 10.3390/s18010258.

[14] B. Asadi, "Bundle adjustment explained," 2019. [Online]. Available: https://ros-developer.com/2019/10/17/bundle-adjustment-explained/.

[15] N. Ben Amor, "Towards a dynamic deployment strategy of wheelchair command applications on heterogeneous architecture," *Journal of Information Assurance and Security*, vol. 11, pp. 117–125, Mar. 2016.

[16] K. Pulli, A. Baksheev, K. Kornyakov and V. Eruhimov, "Realtime computer vision with opencv: Mobile computer-vision technology will soon become as ubiquitous as touch interfaces.," *Queue*, vol. 10, no. 4, pp. 40–56, Apr. 2012, ISSN: 1542-7730. DOI: 10.1145/2181796.2206309. [Online]. Available: https://doi.org/10.1145/2181796.2206309.

[17] S. Aldegheri, N. Bombieri, D. D. Bloisi and A. Farinelli, "Data flow orb-slam for real-time performance on embedded gpu boards," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 5370–5375. DOI: 10.1109/IROS40897.2019.8967814.

[18] C. Wu, S. Agarwal, B. Curless and S. Seitz, "Multicore bundle adjustment," vol. 42, Jun. 2011, pp. 3057–3064. DOI: 10.1109/CVPR.2011.5995552.

[19] R. Hänsch, I. Drude and O. Hellwich, "Modern methods of bundle adjustment on the gpu," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. III-3, pp. 43–50, 2016. DOI: 10.5194/isprs-annals-III-3-43-2016. [Online]. Available: http://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/III-3/43/2016/.

[20] V. Jilkov and J. Wu, "Efficient gpu-accelerated implementation of particle and particle flow filters for target tracking," *Journal of Advances in Information Fusion*, vol. 10, pp. 73–88, Jun. 2015.

[21] L. Murray, "Gpu acceleration of the particle filter: The metropolis resampler," *Preprint arXiv:1202.6163*, Feb. 2012.

[22] H. Strasdat, J. Montiel and A. J. Davison, "Visual slam: Why filter?" *Image and Vision Computing*, vol. 30, no. 2, pp. 65–77, 2012, ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2012.02.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0262885612000248.

[23] S. Vithalani, S. Soni and P. Rajpura, "Autonomous navigation using monocular orb slam2," in *Recent Advances in Communication Infrastructure*, A. Mehta, A. Rawat and P. Chauhan, Eds., Singapore: Springer Singapore, 2020, pp. 59–68, ISBN: 978-981-15-0974-2.

[24] OpenCV, "Fast algorithm for corner detection," 2021. [Online]. Available: https://docs.opencv.org/3.4/df/d0c/tutorial_py_fast.html.

[25] ——, "Opencv: About," 2021. [Online]. Available: https://opencv.org/about/.

[26] K. Karimi, N. G. Dickson and F. Hamze, *A performance comparison of cuda and opencl*, 2011. arXiv: 1005.2581 [cs.PF].

[27] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016. DOI: 10.1177/0278364915620033. eprint: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.full.pdf+html. [Online]. Available: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract.

[28] T. Nguyen, "Orbslam2cuda," *Nanyang Technological University*, 2017. [Online]. Available: https://github.com/thien94/ORB_SLAM2_CUDA.

[29] Adaskit-Team, "Cuda-bundle-adjustment," *Fixstars Corporation*, 2021. [Online]. Available: `https://github.com/fixstars/cuda-bundle-adjustment`.