Development of a Visual Inertial Odometry Positioning System for Autonomous Interplanetary Drone State Estimation

A project presented to
The Faculty of the Department of Aerospace Engineering
San Jose State University

In partial fulfillment of the requirements for the degree
Master of Science in Aerospace Engineering

by

Steven Rispoli

May 2021

Approved by

Dr. Periklis Papadopoulous
Faculty Advisor

Table of Contents

Contents

Abstract

Autonomous vehicles such as interplanetary drones are an expanding topic of interest. Autonomous vehicles such as the Ingenuity Helicopter are making history and inspiring a new era of space exploration. Traditional autonomous vehicles such as the Mars Exploration Rovers used Visual Odometry to know their own position. However, visual Odometry is very computationally intensive producing motion estimates every few minutes. Interplanetary drones need Visual Odometry systems that produce motion estimates at a much faster rate. This paper will develop a Visual Odometry system for an autonomous drone to extrapolate the performance of these systems. These estimates will be able to help predict the functionality of similar systems on Ingenuity and other interplanetary drones in development.

Acknowledgements

I would like to thank my parents for supporting me from the beginnings of my college journey to the end of Graduate school. Thank you for the sacrifices that you made as I had to prioritize my schooling over spending time with both of you. I would also like to thank my friends who have shared in my journey in and out of school. I would also like to thank my Advisor Dr. Periklis Papadopoulos for believing in me and pushing me to go the extra mile to succeed. Without all of you I wouldn't be where I am today.

Nomenclature

Symbol	Definition	Units (SI)
A	Intrinsic Camera Matrix	--------
A'	Intrinsic Camera Matrix for Camera 2 in a Stereoscopic Pair	--------
B	Stereoscopic Camera Baseline	cm
c	Principal Point Coordinate	Pixels
D	Pixel Depth	--------
f	Focal Length	--------
F	Feature	--------
j	Homogenous Camera Coordinate	Pixels
k_n	Radial Distortion Coefficient	--------
$\breve{m}$	Image Projection of P	--------
O	Camera Center	--------
p	2D Point	--------
p_n	Tangential Distortion Coefficient	--------
P	3D Point	--------
Q	Matched Feature Inlier Clique	--------

r	Distance Between Pixels	--------
r_{xy}	Rotation Transformation	--------
R	Rotation Matrix	--------
R'	Rotation Matrix of Camera 2	--------
R_s	Rotation Transformation from Camera 2 to Camera 1 in a Stereoscopic Pair	--------
s	Arbitrary Projective Transformation Scaling	--------
s_n	Thin Prism Distortion Coefficient	--------
$t_{x,y,z}$	Translation Transformation	--------
t	Translation Matrix	--------
t'	Translation Matrix of Camera 2	--------
t_s	Translation Transformation from Camera 2 to Camera 1 in a Stereoscopic Pair	--------
u	Image Width	Pixels
v	Image Height	Pixels
x'	Normalized Camera X Coordinate	Pixels
x''	Distortion Corrected Camera X Coordinate	Pixels
x^O	2D Stereoscopic Camera Coordinate	Pixels
X	3D Point X Coordinate	--------

y'	Normalized Camera Y Coordinate	Pixels
y''	Distortion Corrected Camera Y Coordinate	Pixels
Y	3D Point Y Coordinate	--------
Z	3D Point Z Coordinate	--------
Greek Symbols		
ε	Reprojection Error	--------
ρ	Camera Projection Matrix	--------
ω	Homogenous World Coordinate	Pixels
Subscripts		
$(\,)_a$	Image Frame a	--------
$(\,)_b$	Image Frame b	--------
$(\,)_c$	Camera Frame	--------
$(\,)_w$	World Frame	--------
$(\,)_x$	X-Axis	--------
$(\,)_y$	Y-Axis	--------
Acronyms		
DAPRA	Defense Advanced Research Projects Agency	--------

EKF	Extended Kalman Filter	--------
EOM	Equations of Motion	--------
FAA	Federal Aviation Administration	--------
FPGA	Field-Programmable Gate Array	--------
IMU	Inertial Measurement Unit	--------
FPS	Frames Per Second	
GPS	Global Positioning System	--------
JPL	Jet Propulsion Laboratory	--------
KITTI	Karlsruhe Institute of Technology and Toyota Technological Institute	--------
LET	Linear Electron Transfer	--------
OKVIS	Open Keyframe-based Visual-Inertial SLAM	--------
OpenCV	Open Computer Vision	--------
MER	Mars Exploration Rovers	--------
MSL	Mars Science Laboratory	--------
RAM	Random Access Memory	--------
RGB	Red Green Blue Image Format	--------
RGBD	Red Green Blue + Depth Image Format	--------
RLS	Recursive Least Squares	--------

SBC	Single Board Computer	--------
SGBM	Semi-Global Block Matching	--------
SLAM	Simultaneous Localization and Mapping	--------
VIO	Visual Inertial Odometry	--------
VO	Visual Odometry	--------

List of Figures

1 Introduction

1.1 Motivation

Interplanetary drones are an exciting field of research. With the successful flights of Ingenuity, JPL's Mars helicopter, a new era of aviation is emerging. However, this new era of aviation brings new challenges with it. These drones cannot be flown by a human pilot due to the delay in the commands reaching the drone on these far away planets. Flight on other planets will require these new aircraft to be autonomous.

A large challenge for autonomous drones is the measurement of accurate state data such as its position and velocity. Earthbound autonomous drones face the same problem as interplanetary drones in that they both do not have access to GPS. Autonomous drones on Earth must fly indoors where GPS signals are weak, and interplanetary drones will not have access to GPS at all. Linear acceleration sensors from IMU's can be used to extrapolate velocity and position of the drone, but they are not very reliable. Error in measurement causes sensor drift to accumulate quickly over time, rendering the data useless. Traditionally, GPS would correct the IMU drift for the drone's control system, but without GPS other methods must be used to correct the IMU data.

Visual odometry, or VO, is one such way to correct IMU drift. VO uses an optical sensor like a camera, to measure the egomotion of the camera. A VO system when combined with a traditional IMU is called a Visual Inertial Odometry system or VIO system. VIO systems can be comprised of 1 or multiple cameras. A single camera VIO system is less complex; however, scale estimation is difficult. Multiple camera setups, such as a stereoscopic camera allow VIO

systems to accurately measure the 3D motion that a singular camera setup cannot. VIO will give position and velocity data that has much less drift then the IMU alone.

1.2 Literature Review

One the first times VO was done was back in 1976 by PhD students at Stanford. Hans Moravec and Donald Gennery developed a visual system to track the movement of an electric vehicle. The system worked by sending the camera images wirelessly to a server which did the VO calculations. There are 4 steps to their VO system. First an interest operator determines areas in the image that will be easily identifiable from image to image to track. Then the binary search correlator attempts to find the features from the interest operator. Matches of the features in consecutive frames are sent to the high-resolution correlator to improve upon the matches accuracy. Finally, the camera solver uses the improved matches from the high-resolution correlator to calculate the camera position in both frames to find how far the vehicle has travelled (Moravec, 1976).

1.2.1 Improvement and Applications of VO

Jiawei Mo and Junaed Sattar point out a downfall of monocular VO, such as that mentioned above. Monocular VO can track the movement in only 2 directions. It lacks the depth perception necessary for the tracking of full 3D motion. To gain movement information in the third dimension, they proposed a stereoscopic approach. They use one camera for traditional VO while the other allows the system to fuse the third dimensions motion with that of the monocular VO. In previous work the fusion of monocular VO with a second camera is done by stereoscopic matching. Stereoscopic matching is computationally expensive however, so Mo and Sattar replaced stereoscopic matching with a scale optimizer which performs calculations on a single

pixel location instead of the region around the projected pixel. This allows scale optimization to be more computationally efficient. The scale improvement saw a decrease in computational time versus the stereoscopic matching implementation during testing with the KITTI computer vision dataset. It also saw lower rotational error, but higher translational error. However, the translational error is still below 3.2% compared with the ground truth data. Overall, Mo and Sattar proposed a novel approach to extend monocular VO to work in 3D. It is also less computationally intensive compared to other 3D VO implementations (Mo, 2019).

Stereoscopic VO has been investigated as well. In 2008, Andrew Howard developed a stereoscopic VO system at JPL for DARPA and NASA. His algorithm took rectified disparity maps from a stereoscopic camera as inputs. It used a feature detector and feature matcher to reduce the amount of data being processed before reducing the features even farther by processing only the matching inliers. To calculate the egomotion of the camera a transformation was found that minimizes the reprojection error from stereoscopic frame to frame using the Levenberg-Marquardt least-squares algorithm. Transformations that produce an error below a threshold value are verified and transformations that don't meet the threshold are ignored and discarded. Two tests were done. The first was on a Boston Dynamics BigDog platform. Tests were performed at three different resolutions, at two different frame rates, across three different trajectories for a total of 24 runs. The resolutions were 160x120, 329x249, and 640x480, framerates were 30Hz and 15Hz and there were 2 ovular trajectories and one out and back. The 2D and 3D root mean square error decreases as resolution increases and framerate decreases. However, the increase in resolution also lead to a double in computation time from the lowest to highest frame rates. The improvement from resolution is intuitive but the increase in accuracy from the decrease in framerate is not. This is because the noise integrated from the

transformations in the VO algorithm does not add up as quickly with lower framerates. However, Howard notes that there is a lower bound for this decrease in framerate to improve accuracy due to the travel of the features from frame to frame. The second test was conducted on DARPA's LAGR vehicle. The LAGR vehicle is a platform to assess the slip during ground traversing. VO is used to check the wheel odometry where a discrepancy means that slip is occurring. This test had four trajectories with two different resolutions with two pairs of stereoscopic cameras. The data from the VO algorithm was compared to data from an onboard IMU performing IMU odometry. The 2D and 3D error from the VO system after about 400 meters of travel was about 1 meter or 0.25% as opposed to the IMU error which was about 17 meters or 4.4%. However, the VO system reported several failures where the VO system was not able to produce a transformation under the error threshold. This was due to the slow write speed of the data logger and failure of the stereoscopic disparity matching. Neither f these are faults of the VO algorithm, but of the other supporting systems. Overall, the VO system developed by Howard proved to be accurate in both the tests, and much better than IMU odometry alone.

A further improvement upon VO integrates an IMU into the system to perform VIO. Vladyslav Usenko, Jakob Engel, Jörg Stückler, and Daniel Cremers, use an IMU to correct for drift between the camera frames. This allows the faster IMU to give the VIO system information between camera frames. IMU drift in their approach was mitigated as well, as the motion measured by the IMU between frames integrated from frame to frame instead of IMU measurement. Additionally, Usenko et al. marginalize information between keyframes out. This reduces the amount of calculations that need to be done to estimate the pose in each frame. Then finally, their VO portion uses both the static stereoscopic from both cameras and the temporal stereoscopic images over time. This allows the depth map to be constructed and tracked as the

camera moves. During testing versus IMU coupling that is fused with a standalone VO system, Usenko et al.'s VIO system is superior. With a dataset collected by a drone, The VIO system outperformed the fused VIO, particularly when there was significant motion blur. This held true for both rotation and translation over a course of 40 meters, as well as over time on a course of over 1140 meters. Another test on a dataset taken from a car was also done to test the VIO systems qualitative results, as it was able to successfully track the movement of the car in a challenging optical environment (Usenko, 2016).

1.2.1.1 Drone Use Cases

An early use of VIO to feed a drone's flight controller was done by Farid Kendoul, Kenzo Nonami, Isabelle Fantoni, and Rogelio Lozano. In their work, a VO system was fused with an IMU to provide state data to a drone. A subsequent adaptive control system was also developed for autonomous flight for the drone. As their work was done early, their VO system is older and less advanced than some of the newer VIO systems in the papers mentioned above. It is a monocular system that uses a RLS algorithm that is fused with the IMU and barometer data through a linear Kalman Filter. That data is then fed into a hierarchical control system that separates translational and rotational control. The drone itself is based on an Ascending Technologies GmbH quadrotor drone kit. It uses a Gumstix micro-controller with the console-st and wifistix expansion boards, running a Linux environment. It is responsible for the bulk of the computing, including the VIO fusion and flight controller. Additionally, an AVR micro-controller is used to turn the Gumstix commands into PWM output for the motors, as well as manager emergency tasks such as landing. The drone also has a MNAV100CA combination IMU GPS and barometric sensor. The MNAV100CA interfaces with the Gumstix micro-controller to feed it the sensor data through an RS-232 serial link. As for the camera, it is a

RangeVision KX-171 analog camera with transmitters and receivers for wireless interface with a laptop. The laptop acts as a ground control station and is used to give commands to the drone. It also does some VIO image processing from the video feed sent from the drone. The drone itself does additional VIO calculations and fusions and runs the flight controller. Testing the drone in autonomous flight saw a 2 meter max error in the 3D position estimate. However, Kendoul et al. attribute this error to windy flight conditions. That being said, the drone was successfully able to fly autonomously and track reference angles. An additional test to fly a pre-determined trajectory was also done. In that test GPS data was unavailable, but the drone was still able to fly the trajectory due to the VO algorithm. However, they do note that over time they still see drift in the VO data. (Kendoul, 2009).

Building inspection is another area where VIO on drones is a key system. Duarte Dornellas, Filipe Rosa, Alexandre Bernardino, Ricardo Ribeiro and Jose Santos-Victor developed a drone system to autonomously take pictures of buildings in areas where GPS signal is inconsistent. They used the open source Kalibr and OKVIS libraries to calibrate and design their VIO system. The drone used a PixHawk as the flight computer, and a Nvidia Jetson TX2 as the VIO processor. Additionally, an Arduino Nano was used to trigger the IMU and stereoscopic camera readings. The MAVLink 2.0 protocol was used for communication between the PixHawk and the TX2. Additionally, ArduPilot's Mission Planner was used as the ground station basis. The stereoscopic setup was composed of 2 FLIR/Point Grey BFLY-U3-05S2M-CS cameras with Fujinon YF2.8A-2 lenses mounted to each. The IMU used was an XSens MTx-28A33G35. Both had API's but the IMU does not have Linux support. For their VIO system, the cameras were operating at 20 Hz while the IMU operated at 200 Hz to maintain synchronization. During tests of flight, the drone flew 2 pre-planned trajectories. The first flight had ground truth data and it

was observed that the translation error averaged 0.021 meters with a peak under 0.05 meters, while the orientation error averaged 2.183 degrees with a peak below 5 degrees. The second flight showed similar results, however there was a large spike in the orientation error due to an error in the ground truth capture. Two additional flights were conducted outside, which compared the VIO position data to GPS data taken during the flight. Both flights showed drift in the VIO data compared to the GPS data. However, the GPS data was not reliable in the second outdoor flight and aid by the VIO system gave better results than the GPS alone. Overall, Dornellas et al. quantitatively showed that VIO could be used on a drone to fill in the gaps of poor GPS signal (Dornellas, 2019).

1.2.1.2 Interplanetary VO

Visual Odometry does have some precedence in interplanetary use as well. Both MER utilized VO for portions of the autonomous driving that was instructing. Wheel encoders were used successfully on level rigid surfaces but were found to exceed the 10% per 100 meters drift required. A VO system had been developed for use on the MER but had not gone through the operational readiness tests. However, it was able to work successfully to track the rover's motion. However, the computation time was so long it increased the autonomous drive speed by an order of magnitude. Comparing the VO with the traditional wheel odometry on an identical test rover the VO system for the MER was much lower than wheel odometry. During a test over slip inducing terrain, wheel odometry exceeded the 10% error after only 1.4 of the 2.45 meter distance. VO, however, saw error around 1% at the end of the 2.45 meters. To achieve these results there were some constraints that needed to be met. Due to the rover's slow CPU, the VO calculations took up to 3 minutes for each image pair. This dramatically increased the time it took to traverse land. It also put more stress on the VO algorithm to be as accurate as possible.

As well, the long computation time necessitated that the human driver decides what to image for the VO calculation, so that the feature detector has enough data to track. Due to these challenges VO was only used in very specific situations consisting of short steep slippery terrain. Regardless of these limitations, VO proved to be an effective tool for the MER to safely and accurately traverse the Martian landscape (Maimone, 2007).

Building upon the success of VO on the MER, VO was implemented on the MSL Curiosity rover as well. Curiosity saw a large improvement on its VO computation time over the MER. This was not only due to a newer generation CPU, but the introduction of a FPGA as a vision co-processor. During tests in Curiosity's development, Howard et al. used a Xylinx Virtex 5 FPGA to assess its performance over MER's VO implementation. The results of the tests showed a vast improvement over MER's implementation. The stereoscopic calculations took about 24 to 30 seconds on the RAD6000 CPU found on the MER compared to 0.005 seconds on the FPGA accelerated setup. The performance gains continued to be stable as increasing the resolution of the images from a width of 256 to 1024 pixels only saw the FPGA take 0.082 seconds to do the stereoscopic calculation. Additionally, a test of the whole VO system was done. The MER analog completed the visual odometry calculations in about 160 seconds while the FPGA as a co-processor took 0.016. However, only the feature detection and matching were run on the FPGA and following calculations would have to be done on a separate CPU (Howard, 2012).

The Mars 2020 Perseverance rover see's additional improvements over Curiosity's VO hardware and programming. Perseverance adds an additional RAD750 CPU and Vertex 5 FPGA. As well as this, the calculations have been parallelized. This results in the VO that took 65 seconds on Curiosity to take a total of 9.8 seconds on Perseverance. This increase in VO

calculation, among other penalizations, increased Perseverance speed to about 100 meters per hour compared to Curiosity's roughly 15 meters per hour (Rieber, 2018).

1.2.2 Interplanetary Drones

Perseverance is not the only vehicle that just landed on Mars. The Ingenuity helicopter is travelling to Mars alongside Perseverance. Ingenuity is a technology demonstrator used to assess the viability of drone operations on Mars. It will be the first vehicle to attempt powered flight on Mars. Ingenuity is planned to fly daily during a period of 30 Sols. The planned flight duration will be up to 90 seconds and cover up to 300 meters at altitudes between 3 and 10 meters. At the time of writing this paper, Ingenuity has made 4 successful flights with additional flights scheduled.

The drone is built around a central structural tube. The electronics from the avionics bay sits at the base of the tube. The wires from the electronic bay are fed up to the top of the tube. The top of the drone is where the co-axial counter rotating blades are as well as the solar panel and antenna. Figure 1 shows an illustration of the drone with some labels.

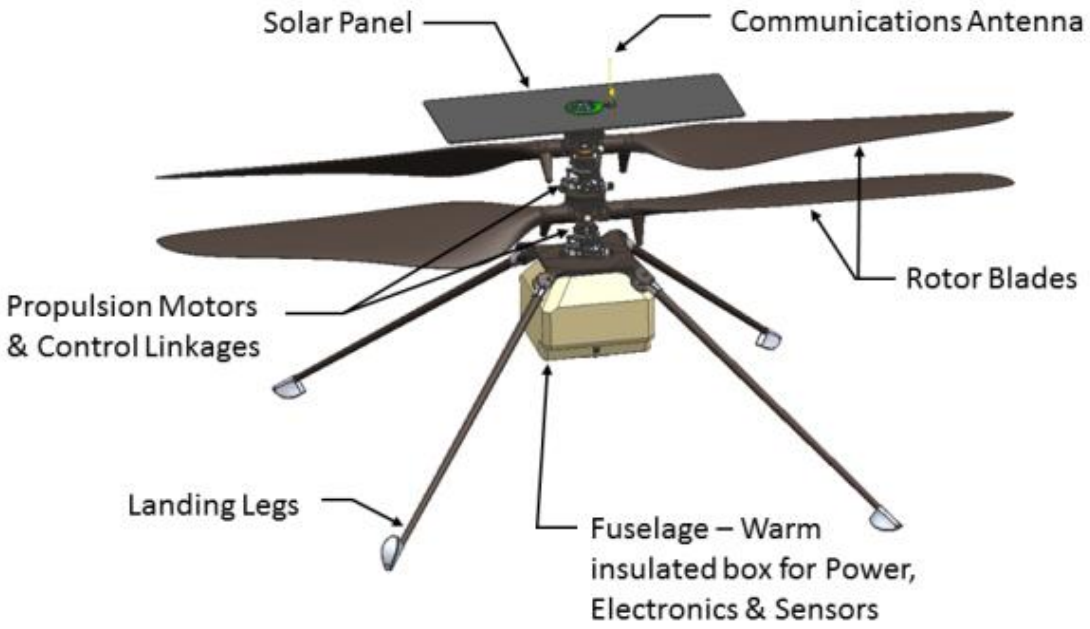

Figure 1: Ingenuity Engineering Model Design

It has a complex avionics system that ensures safe flight for the helicopter. Its primary processor is a Qualcomm Snapdragon™ 801 processor at 2.26 GHz with 2 GB of RAM and 32 GB of Flash memory. It has 2 identical Texas Instruments Hercules TMS570LC43x automotive processors at 300 MHz with their own 512 KB of RAM and 4 MB of flash memory. These are redundant processors that receive and process identical. These perform flight control functions that are critical to the operation of the drone. During flight one of the two will be the primary processor while the other is waiting to be hot-swapped in case a fault causes the primary to restart.

To perform the mission critical tasks, Ingenuity utilizes a military grade radiation hardened ProASIC3L FPGA from MicroSemi. It performs all the I/O to the sensors, actuators,

and fault management functions. It also does the hot-swap of the Hercules microcontrollers. Specifically, it runs the flight controller including an altitude control loop, the rotor system controller, waypoint guidance, sensor I/O from the IMU altimeter and inclinometer. It also manages the current and temperature telemetry. In addition, it does the time system management, power management, and thermal controls. All communication of the various systems are done via the FPGA which also stores and restores data to the processors when they have to recover after a fault. Additionally, triple module redundancy has been implemented to critical flip-flops to further protect against data faults.

Ingenuity has a variety of sensors to keep the drone safe during flight. Unlike the processors and FPGA the sensors are commercial off the shelf. Ingenuity has 2 Bosch Sensortec BMI-160 IMUS, a MuRata SCA 100T-D02 inclinometer, and a Garmin Lidar-Lite-V3 with a range of 10 meters. The camera used for navigation is an Omnivision OV7251 camera with a resolution of 640x480, a global shutter, and a field of view of 133x100 degrees. Ingenuity also has a Return to Earth camera that is used to collect high resolution images to send back to Earth. Its is a Sony IMX214 with a resolution of 4208x3120, a rolling shutter, and a field of view of 47x47 degrees. There is a slight overlap between the navigation camera and the Return to Earth camera as well that allows for feature registration post processing on Earth (Balaram, 2018).

1.2.3 Supporting Hardware for VIO

There are 3 types of processors that can be used on an interplanetary drone. These are the CPU, GPU, and FPGA. Each of them have strengths and weaknesses that need to be considered when deciding which one to use. CPUs are the easiest of the 3 to work with. They are easily reprogrammable and are the most versatile of the 3. They have an extensive flight heritage,

including the MER, which used the radiation hardened RAD6600 (Cheng, 2006). The RAD series of CPUs is the premiere line of radiation hardened CPUs. Currently the most capable RAD series CPU is the RAD750. Even though it is the cutting edge for space processors it is an order of magnitude behind cutting edge of contemporary processors in terms of processor speed.

GPUs are a processor that is starting to gain popularity for heavy computational tasks. Generally its individual clock speed is slower than a CPU, however, they have many more cores than CPUs. This large amount of cores is the main draw for GPUs. This allows them to parallelize many small processes, and run them concurrently instead of serially like with CPUs. GPUs are generally more difficult to program, but the performance benefits justify the extra effort. Unlike the RAD series of CPUs, there are no commercially available radiation hardened GPUs. This could become a problem for interplanetary use as they will be much more susceptible to faults.

FPGAs are a unique type of processor that can have its hardware reprogrammed to perform a specific task at the hardware level. Like CPUs, FPGAs also have an extensive flight heritage with several lines of radiation hardened FPGAs from Xilinx. Unlike CPUs, once the FPGA has been programmed to do its specific task, it can only perform that task until it is reprogrammed to do another. Additionally, it is much more difficult to program FPGAs compared to GPUs and CPUs. Fortunately, the increase in computational time minimizes these drawbacks.

Paulo Ricardo Possa et al (2013) compared the performance of a CPU, GPU, and FPGA of edge and corner detection both of which are important for feature detection. They compared an Intel Core2 Duo E6600, CPU at 2.4 GHz, a GeForce GTX 580 at1 54 GHz, and a Cyclone IV EP4CE115 FPGA configured at 242 MHz for the Canny edge detector and 232 MHz for the

Harris corner detector. For the edge detector, the FPGA ran the algorithm 1.92 – 0.93 times faster than the GPU and 27.3 – 23.3 times faster than the CPU, as the image size increased from 512x512 to 3936x3936 pixels. Additionally, the energy consumption of the FPGA was 154 – 167 times more energy efficient than the GPU, and 94 – 102 times more energy efficient than the CPU, as the image size increased from 512x512 to 3936x3936 pixels. For the corner detector, the FPGA ran the algorithm 2.02 – 0.96 times faster than the GPU and 17.4 – 20.9 times faster than the CPU, as the image size increased from 512x512 to 3936x3936 pixels. Additionally, the energy consumption of the FPGA was 154 – 166 times more energy efficient than the GPU, and 94 – 101 times more energy efficient than the CPU, as the image size increased from 512x512 to 3936x3936 pixels. However, none of these processors tested were radiation hardened so the comparison isn't a direct comparison for an interplanetary drone use case. That said, Ingenuity has a non-radiation hardened CPUs and was able to fly multiple successful flights.

1.3 Proposal

The goal of this project is to extrapolate the performance of Earth based VIO systems to an interplanetary drone. A VIO system will be developed on a more conventional embedded system rather than with an FPGA. Additionally, this project does not look to improve on existing VIO but to implement an existing VIO system for benchmarking. Both performance and efficiency of the implemented VIO will be conducted, then based on a variety of factors, will be expanded to make an approximation of the performance that can be expected on an interplanetary drone.

There are three success criteria for this project as follows.

1. Complete Success Criteria:

a. A VIO system will be implemented and flown on the Theia drone and in the virtual simulator. Tests will be performed to assess the execution time of the VIO system on the actual drone during flight. Groundtruth from the simulator will be used to assess the accuracy of the algorithm. From these tests a feasibility study will extrapolate these results to hardware that is likely to appear on other interplanetary drones.

2. Partial Success Criteria:

    a. A VIO system will be implemented and tested in the virtual simulator. Tests will be performed to assess the execution time of the VIO system on the actual drone during flight. Groundtruth from the simulator will be used to assess the accuracy of the algorithm. From these tests a feasibility study will extrapolate these results to hardware that is likely to appear on other interplanetary drones.

3. Minimum Success Criteria:

    a. A VO system will be implemented and tested with a stereoscopic camera setup. Tests will be performed to assess the execution time of the VO system. A feasibility study will extrapolate these results to hardware that is likely to appear on other interplanetary drones.

1.4 Methodology

There are two major parts to the completion of this project. The first part is to implement a VIO system that will allow a testing drone to achieve autonomous flight without receiving state

data from a GPS. The second part is to take the results from the first part to approximate expected VIO results.

Part I: Validation of VIO on an Autonomous Drone

1. Specify test drone embedded flight computer.
2. Identify VO/VIO algorithms that are best suited for autonomous flight.
3. Implement the identified VO/VIO algorithm in C++. Add IMU fusion in the case the identified algorithm is VO only.
4. Deploy the C++ VIO code to the test drone/simulator.
5. Test the VIO under manual control to compare with GPS data in flight, or groundtruth data from the virtual simulator.
6. Make adjustments to the VIO system if it does not achieve a reasonable error.
7. Test the VIO on the test drone for a predetermined autonomous flight.

Part II: Extrapolate Autonomous VIO Performance to Interplanetary Performance

1. Determine hardware that will likely be used for interplanetary drones.
2. Compare performance criteria of selected hardware with the hardware used on the test drone.
3. Investigate performance tradeoffs between different processor architectures.
4. Use the performance tradeoffs study to scale test drone VIO performance to an interplanetary drone.
5. Discuss the viability of VIO as a real time system on interplanetary drones

2 Drone Systems

For this paper, a VIO system will be built and deployed to demonstrate the effectiveness of a VIO system as a form of state estimation for an autonomous drone. The drone test platform is a racing drone that was developed to emulate the drones in the Lockheed Martin-Drone Racing League AlphaPilot RacerAI drone. Theia, the drone used is the same one in built by Walter Harper Steven Rispoli and Brayan Mendez (2020). The design and hardware of the drone will be briefly explained in the following sections as well as the underlying software.

2.1 Drone Hardware

Like the AlphaPilot RacerAI, Theia has a forward biased quadcopter design rather than the X quadcopter design. The drone features 1380KV motors with 60 Amp 5S electronic speed controllers fitted to 7x4x3 propellers. It features a BNO055 IMU and ultrasonic sensor for orientation and altitude determination. In addition, it uses two Arducam BO200 cameras mounted to form a stereoscopic pair. Its flight controller is a distributed system comprised of 2 SBC's. The SBC's used are a Raspberry Pi 4 and an Nvidia Jetson Nano. They communicate through a peer to peer ethernet connection. The Pi was chosen for its fast processor speed of 1.5 GHz and its large 4 GB of RAM. The Jetson Nano was chosen to process more intensive applications. It features a 64-bit 1.43 GHz processor 4 GB of RAM and a 128-core Maxwell GPU.

2.2 Drone Software

The Raspberry Pi 4 runs a headless Ubuntu 16 operating system. It runs the internal messaging system that allows the other software packages to communicate with each other. It

utilizes TCP/IP sockets to transmit byte data in a multithreaded Client-Server model. It runs faster than any of the other software packages at 1000Hz and can be adapted to handle fixed and varying data package sizes. The control system is a set of sub-controllers for translational and rotational control. It features cascade P, PD, and PID controllers, as well as anti-windup clamping, output saturation, and filtered derivative methods to improve stability. Currently, the control system is linearized about the hover condition so fast movement may create less stable conditions. However, for autonomous flight, the speed will likely be low enough to mitigate the unwanted behavior. The control system uses the messaging client to communicate with a motor mixing algorithm, that interfaces with the drone's motors. Currently, the only state feedback to the controller is the IMU. The IMU alone builds up too much error to be useful as the only form of state estimation. GPS is also ineffective as there is not FAA licensing for autonomous drones. That is why VIO is so important for autonomous flight.

Beyond the Raspberry Pi, the Jetson Nano runs a Linux for Tegra version of Ubuntu that is tailored for Nvidia Jetson SBCs. The Jetson Nano handles the stereoscopic camera calculations and was also used for object detection, however for this paper the object detection module will be ignored. The stereoscopic camera utilizes the OpenCV C++ library to produce a depth map of the environment in front of the drone. Coupled with the object detection module, the camera system would generate waypoints for the control system to fly to. However, the coupling has yet to be implemented before fully autonomous flight can be achieved. The VIO system would be primarily run on the Jetson Nano to leverage the parallelization capability of the Maxwell GPU. Using OpenCV's CUDA library the VIO algorithm will be deployed on the GPU.

2.3 Supporting Drone Systems

Beyond the onboard hardware and software, there is a Linux server that runs a virtual simulator for the drone. The server has a first generation 12-core Threadripper 1920x CPU with a maximum speed of 4 GHz. It has two GPUs, a GTX 1080 and a P102-100 and 64 GB of 3000 MHz DDR4 RAM. The server runs a flight simulator built in Unity 3D. It utilized the Nvidia PhysX engine to do the physics modelling. The EOMs were derived for Theia and implemented in the flight simulator. The flight controller can be validated on the server through the flight simulator. The internal messaging system communicates with the flight controller and the simulator and graphs the controller results on the server. Additionally, the cameras are modelled in the simulator and the object detection and depth modules work in the simulator as well. The simulator is also capable of operating with a hardware in the loop capacity, where the controller commands are sent to the drone, as if it were flying through the simulator. This capability provided additional validation that Theia would be flight capable with a satisfactory state estimation system.

3 Computer Vision Algorithms

OpenCV is the basis for much of the code for the VIO algorithm. It has implementations for numerous computer vision algorithms. The important models that need to be understood to put the VIO algorithm into context are the camera model, distortion model, and stereoscopic camera model.

3.1 Camera Model

OpenCV uses a simplified version of Zhang's (Zhang, 2000) pinhole camera model

$$s\,p = A[R|t]P_w \tag{3.1}$$

Where, p is a specific pixel in the image, A is the intrinsic camera matrix, R is the rotation matrix from the camera to world frame, t is the translation vector from the camera to world frame, P_w is the 3D world coordinate of the specific pixel, s is the arbitrary scaling of the projective transformation. Zhang's model projects the 3D world coordinates in an image to their respective location in the camera image coordinates. A is comprised of the focal lengths f_x and f_y and the coordinates of the principle point $(c_x,\ c_y)$ in pixels

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

This matrix is specific to the optics of the camera meaning that it can be used as a constant for every image if the camera is the same It is also able to project 3D camera coordinates to 2D camera coordinates

$$p = AP_c \tag{3.3}$$

Expanding equation 3.3 gives

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{3.4}$$

The transformation between the camera frame and the world frame in 3D is done with the joint $[R|t]$ extrinsic rotation-translation matrix

$$P_c = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} P_w \tag{3.5}$$

Expanding equation 3.5 gives

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{3.6}$$

Furthermore equation 3.6 can be rewritten in a normalized manner where $x' = \frac{X_c}{Z_c}$ and $y' = \frac{Y_c}{Z_c}$

$$Z_c \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{3.7}$$

Equation 3.1 can be fully expanded to be

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{3.8}$$

Examining equation 3.8 we can make the following simplifications

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x \frac{X_c}{Z_c} + c_x \\ f_y \frac{Y_c}{Z_c} + c_y \end{bmatrix} \tag{3.9}$$

Where

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = [R|t] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{3.10}$$

Figure 2 helps to visualize equations 3.1 through 3.10. It helps to explain the pinhole model that is used by OpenCV. The red ray traces a line through the pinhole at the origin of the camera frame, through the 2D point (u, v) on the image, to the 3D point $P = (X_w, Y_w, Z_w)$. The boxes on the plane uv represent the pixels in the image to the point (u, v). The point (u, v) is the only dimensional quantity in the model, the other variables all derive their dimensions from the

projective transformation scaling s. However, finding s can be difficult so another method to find the image scale must be used.

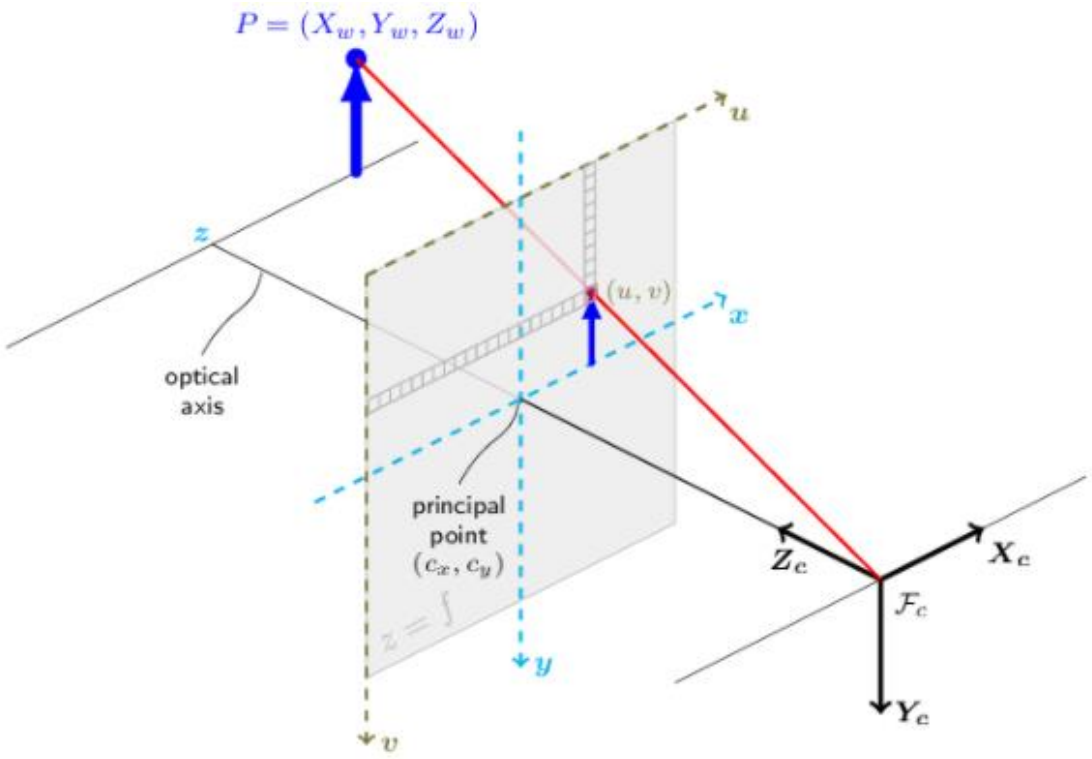

Figure 2 – Pinhole Camera Model Visualization

3.1.1 Extension to Stereoscopic Calibration

Zhang (Zhang, 2009) extended his model for camera calibration to stereoscopic cameras to support other projects he was working on at the time. In his stereo model the second camera would be denoted with a $'$. The transform between cameras is (R_s, t_s) such that $(R', t') = (R, t) \cdot (R_s, t_s)$. This relationship is shown more precisely in equations 3.11 and 3.12.

$$R' = RR_s \tag{3.11}$$

$$t' = Rt_s + t \tag{3.12}$$

To find (R_s, t_s) the cost function in equation 3.13 is minimized.

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\left[\delta_{ij}\left\|p_{ij}-\breve{m}\left(A,k_1,k_2,R_i,t_i,P_j\right)\right\|^2+\delta'_{ij}\left\|p'_{ij}-\breve{m}\left(A',k'_1,k'_2,R'_i,t'_i,P_j\right)\right\|^2\right] \quad (3.13)$$

In equation 3.13 δ_{ij} and δ'_{ij} are Booleans denoting if the point j is visible to the camera.

3.2 Distortion Model

The distortion model used in OpenCV is a combined version of the Brown-Conrady model (Brown, 1966) and the Fitzgibbon division model (Fitzgibbon, 2001). The combined model includes the radial and tangential distortion terms as well as the thin prism model as seen in Wang (Wang, 1992). The full OpenCV model is as follows

$$\begin{bmatrix}u\\v\end{bmatrix}=\begin{bmatrix}f_x x''+c_x\\f_y y''+c_y\end{bmatrix} \quad (3.14)$$

Where

$$\begin{bmatrix}x''\\y''\end{bmatrix}=\begin{bmatrix}x'\frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6}+2p_1x'y'+p_2(r^2+2x'^2)+s_1r^2+s_2r^4\\y'\frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6}+2p_2x'y'+p_1(r^2+2y'^2)+s_3r^2+s_4r^4\end{bmatrix} \quad (3.15)$$

And

$$r^2=x^2+y^2 \quad (3.16)$$

These models ignore the higher order terms past r^6. OpenCV's implementation also uses a simplified version when k_3 is 0. As noted by Zhang (Zhang, 2000) the distortion is predominantly dominated by the first two terms in the radial distortion model. Thus equation 3.12 simplifies to

$$\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} x'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{bmatrix} \quad (3.17)$$

Figure 3 shows the effect of the first distortion coefficient on an image. As can be seen, negative values indicate that there is barrel distortion and positive values give pincushion distortion. An undistorted image will have a value of 0 for all the distortion terms.

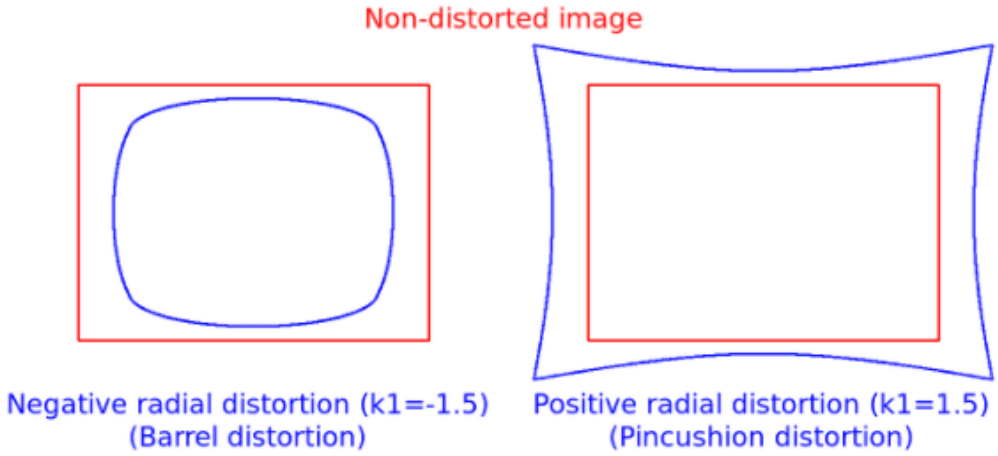

Figure 3 – The Effect of Distortion k_1 on an Undistorted Image

3.3 Stereoscopic Camera Model

The stereoscopic camera model allows for the measurement of depth within an image. Every pixel in the image with some exceptions will be abled to be paired with a depth reading, allowing for a 3D point cloud to be constructed that represents the scene in the image. This depth is inversely proportional to the disparity between the pixel of the object in one camera image compared to the other camera. Figure 4 gives a visualization of this geometry. Using similar triangles gives us the following relationship (Fidler, 2015)

$$\frac{B}{D} = \frac{B + x^O - x^{O'}}{D - f} \quad (3.18)$$

After some simplification we get

$$D = \frac{Bf}{x^O - x^{O\prime}} \tag{3.19}$$

The disparity in the images is given by $x^O - x^{O\prime}$. However, since depth is tied to one pixel, the stereoscopic model projects the depth to only one of the camera views. Additionally, This model assumes that the images from the two camera are rectified so that there is no vertical translation or any rotations between the images.

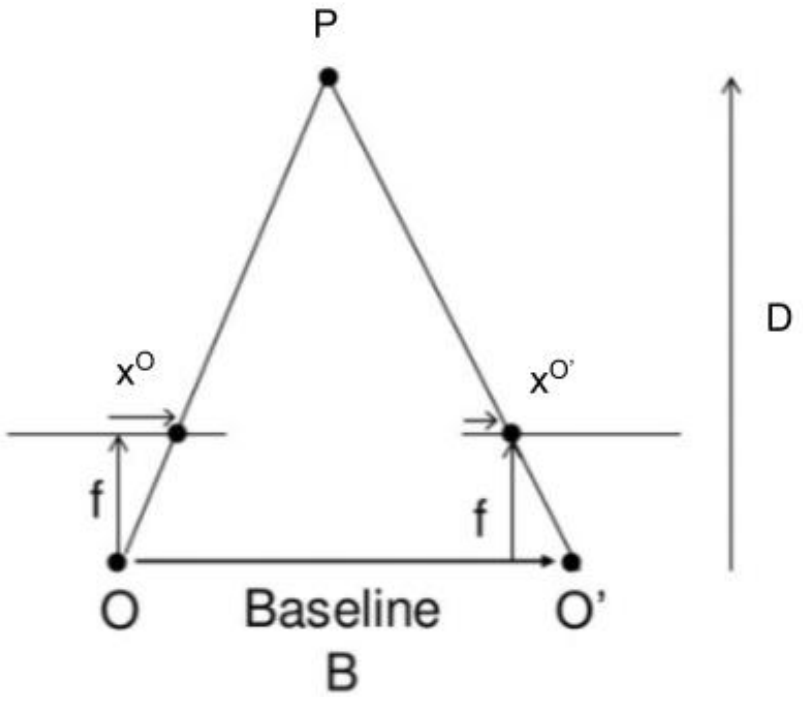

Figure 4 – Stereoscopic Camera Model Visualization

3.4 Stereoscopic Correspondence

The stereoscopic correspondence algorithm used in OpenCV is based on Heiko Hirschmuller's (2008) paper Stereo Processing by Semiglobal Matching and Mutual Information. His algorithm for stereo correspondence is based on a pixelwise matching of Mutual Information between images. It would use a variety of 1D constraints to approximate a global 2D smoothness constraint.

The first step is to match pixels using a cost function. Images are assumed to have been rectified so that their epipolar lines match but it is not required. The cost of the calculation is based on Mutual information MI, defined by the entropy H, of the 2 images shown in 3.20:

$$MI_{I_1,I_2} = \sum_p mi_{I_1,I_2}\left(I_{1_p} I_{2_p}\right) \tag{3.20}$$

Where mi_{I_1,I_2}:

$$mi_{I_1,I_2}(j,k) = h_{I_1}(i) + h_{I_2}(k) - h_{I_1,I_2}(j,k) \tag{3.21}$$

This leads to the cost function with pixel intensity I_{bp} and correspondence I_{mq}:

$$C_{MI}(p,d) = -mi_{I_{b,f_D(I_m)}}\left(I_{bp} I_{mq}\right) \tag{3.22}$$

Where q is the base image with disparity d:

$$q = e_{bm}(p,d) \tag{3.23}$$

The second step is cost aggregation. This prevents wrong pixelwise matches that have lower costa than the correct matches. The matching costs are aggregated in 1D from all directions equally. The cost $S(p,d)$ at pixel p with disparity d is given by (3.24):

$$S(p,d) = \sum_r L_r\,(p,d) \tag{3.24}$$

Where $L_r(p,d)$ is the cost along each path calculated by where $r$is the direction of the pixel and $P_1$and $P_2$are constant penalties:

$$L_r(p,d) = C(p,d) + min\big(L_r(p-r,d), L_r(p-r,d-1) + P_1, L_r(p-r,d+1) + P_1, {}^{min}_{i} L_r(p-r,i) + P_2\big) - {}^{min}_{k} L_r(p-r,k) \quad (3.25)$$

Step 3 is the computation of disparities. $D_b$is the disparity of the image while D_m is the disparity of the match. Disparity is the disparity that minimizes (3.24). If the disparities differ they are marked as invalid.

Step 4 is a refinement of the disparity image as errors can still be present. Peaks in the disparity image are filtered out by segmenting the image and setting the disparity to invalid where variance within the segment are above a certain threshold. Areas with low texture often lead to areas of inconsistent disparity. These areas are assumed to have some correct disparities and the low texture areas are segmented as well. A hypothesis of best fitting planes is proposed that best matches the segments. The cost aggregation function is used to find the plane that best matches the segments. This plane sets the disparity values in the segments of low texture. Steps 2 and 3 as well as the peak filtering may accidentally create invalid disparities where there should not be. To undo these invalidations, the depth image is interpolated depending on whether the invalidation is caused by a mismatch or an occlusion. Mismatches are determined by the consistency check that matches the epipolar lines of the images. For occluded images the interpolation is done from the occlude. For mismatched images, neighboring occluded pixels are found and the mismatched pixel is treated as occluded.

3.5 VIO Algorithm

The VIO algorithm will be a blend of the VO algorithm in Howard (2008) and Usenko (2016). It will be heavily based on the algorithm by Howard, with keyframe and IMU integration from Usenko's algorithm. Both algorithms find the position of the drone by minimizing the reprojection error of a transformation between camera frame to camera frame. However, there are a lot of steps that need to be taken before this transformation can be found.

Howards formula requires inputs of the raw rectified and disparity images. Luckily, the depth module deployed on Theia already calculates these images from the stereoscopic camera. Using the prefiltered and disparity images a set of features is found that contains the pixel location and the world location. This is done for consecutive images so that the translation can be found between the two times the images were taken. After the initial set of features is found the successive set becomes the first feature set. This means that for every image after the first two only one feature set needs to be calculated for the current image. Next a score matrix is calculated from the feature sets using a sum of absolute differences calculation where lower scores are desirable. The score matrix then matches features from frame to frame and puts the pair into a matching matrix. After the matching matrix is created some of the matches are filtered out by comparing the world locations of the features and thresholding them out. Matches that meet the threshold are put into a binary consistence matrix represented as ones. After the consistence matrix is found the maximum inlier set is found. The maximal clique of the consistence matrix is found, and compatible matches of sets in the maximal clique are found and added to the inlier set. At this point the motion of the camera can be estimated by using the matches in the clique. A translation that minimizes the reprojection error of a transformation

between frames is found using the Levenberg-Marquardt least-squares algorithm. The Error equation is given as

$$\varepsilon = \sum_{(F_a,F_b)\in Q}(j_a - \rho\Delta\omega_b)^2 + (j_b - \rho\Delta^{-1}\omega_a)^2 \qquad (4.1)$$

Where

ε is the reprojection error

F is a feature in the image

Q is the clique of the matched feature inliers

j is the homogenous image coordinates

ω is the homogenous world coordinates

ρ is the camera projection matrix

Δ is the transformation between image a and b.

If there are enough points in the clique, the co-linearity is close to one, and the reprojection error ε is below a certain threshold the transformation Δ is valid. The transformation Δ is calculating the egomotion of the stereoscopic camera, which in turn measures the movement of the drone.

4 Results

Overall, successful trajectory estimates were not able to be achieved. There are bugs in the code that are preventing correct estimates from being calculated. The first suspected cause of the problem is the input to the VO function in OpenCV. The RGBD Odometry object is meant to use input from a Microsoft Kinect camera. The Microsoft Kinect captures RGBD images with a depth assigned to each pixel. This implementation however the depth map is calculated from a stereoscopic pair of images. Then the depth map is packaged with its corresponding image so

that it can be used by the RGBD Odometry object. It is possible that the assignment of the depth image to the RGB image from the stereoscopic camera has not been implemented correctly and is causing the RGBD Odometry object to give undefined behavior.

The second suspected cause is that the cameras were not calibrated correctly. At the and of the calibration cycle the accuracy of the calibration is given by the RMSE of the calculations. The general guideline for camera calibration is to get an RMSE below 1, or below 2 if not possible. However, the calibration process is an iterative process that differs with every set of calibration images. This can lead to false positive in the calibration where low RMSE values below 1 are calculated, but the resulting calibration is visibly incorrect with more distortion than the initial images. After tests were done it was found that the depth map coming from the SGBM object was giving depth results that were not consistent for the image. A new calibration was performed and the depth map was much more consistent, producing a depth image that was human readable.

Nevertheless, the complete pipeline from image input to odometry output is complete and runs. Even though correct estimates were not able to be found, an analysis of the execution time of the VO algorithm can be done. Execution times on 4 different devices were logged and averaged to give performance estimates for similar drone systems. The 4 devices are:

Nvidia Jetson Nano @1.43GHz

Nvidia Jetson Nano HITL @1.43GHz

Nvidia Jetson Xavier @2.26GHz

Nvidia Jetson Xavier HITL @2.26GHz

Laptop based Intel i7 5700HQ @2.7GHz

Desktop based AMD Ryzen 9 3950X @ 3.5GHz.

The devices included are both desktop level and embedded system level. Additionally, the 2 Jetson devices were run distributed in a HITL mode with another device handling the supporting systems, and as a standalone device running the VO as well as the communication server.

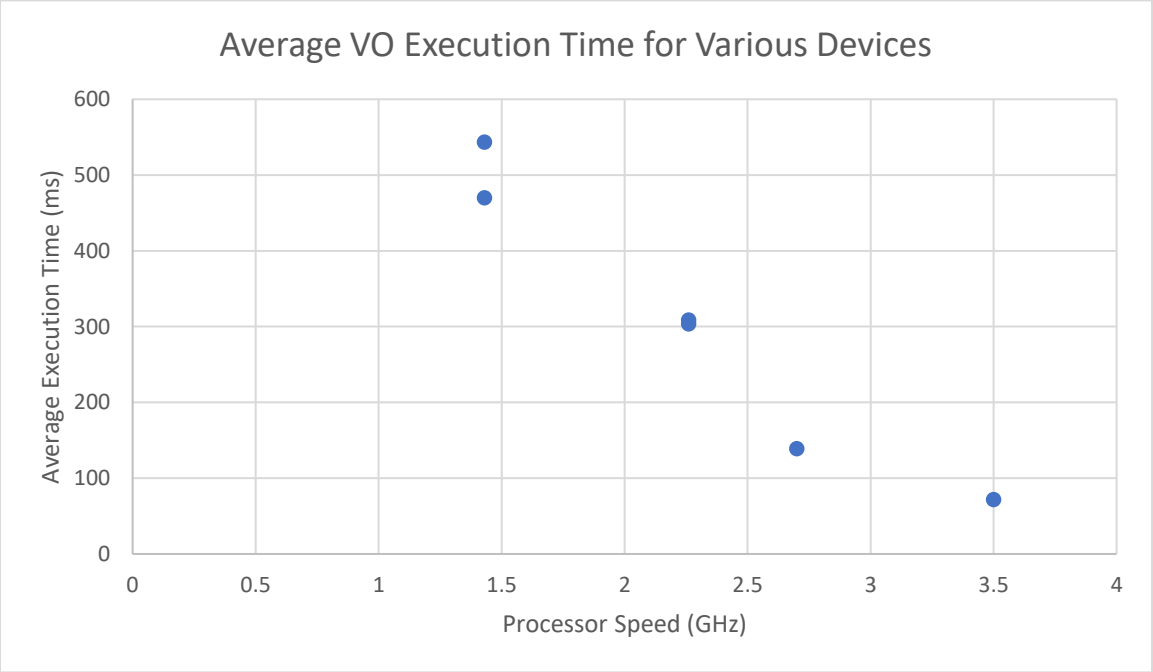

Figure 5 – Average Execution Time of the Tested Devices. There are 2 data points at 1.43GHz and 2.26GHz for standalone and HITL.

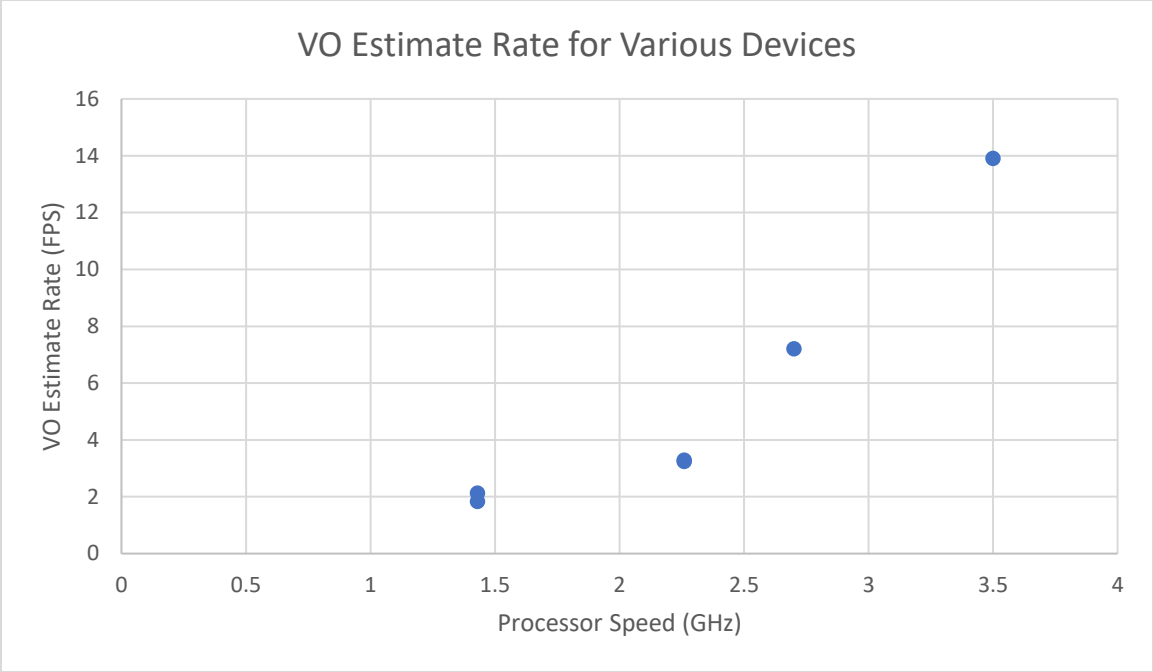

Figure 6 – VO Estimate Rate of the Tested Devices. There are 2 data points at 1.43GHz and 2.26GHz for standalone and HITL.

Figure 5 shows the results of the average VO execution times, while Figure 6 shows the rate at which the estimates are calculated. The 2 embedded devices the Nvidia Jetson Nano and the Nvidia Jetson Xavier have update rates of about 2 and 3 estimates per second. The laptop processor runs at about 7 FPS while the desktop processor runs at about 14 FPS. Using the results from Paulo Ricardo Possa et al (2013) this could increase by up to about 2 times with use of a GPU and up to about 27 times with an FPGA. For the embedded Jetson devices that would give update rates of 4-6 FPS with a GPU or 54-81 FPS with an FPGA. These are only estimates however and are not entirely representative of results that one would expect if actually implemented. Paulo Ricardo Possa et al (2013) only focused on corner and edge detectors in their paper, leaving out a multitude of other calculations in the VO algorithm that are not included in these performance gain estimates.

For an interplanetary drone, the update rates from Figure 6 for the HITL tests are too low to be used confidently in autonomous systems. Ingenuity flies at about 2 meters per second. Update rates on the low end mean that the drone would travel 1 meter between estimates. IMU integration for a VIO system can gap the downtime between estimates of the VO system. The drawback of IMU integration, however, is that drift accumulates very fast. An IMU running at 80 Hz would propagate drift through 39 estimations before being corrected by the VO update in this scenario.

References

Balaram, B., Canham, T., Duncan, C., Grip, H. F., Johnson, W., Maki, J., . . . Zhu, D. (2018). Mars Helicopter Technology Demonstrator. 2018 AIAA Atmospheric Flight Mechanics Conference. doi:10.2514/6.2018-0023

Brown, D. C. (1966). Decentering Distortion of Lenses.

Cheng, Y., Maimone, M. W., & Matthies, L. (2006). Visual odometry on the Mars exploration rovers - a tool to ensure accurate driving and science imaging. *IEEE Robotics & Automation Magazine*, *13*(2), 54–62. https://doi.org/10.1109/mra.2006.1638016

Dornellas, D., Rosa, F., Bernardino, A., Ribeiro, R., & Santos-Victor, J. (2019). GPS emulation via visual-inertial odometry for inspection drones. *2019 19th International Conference on Advanced Robotics (ICAR)*. doi:10.1109/icar46387.2019.8981597

Fidler, S. (2015). *Depth from Stereo*. Lecture.

Fitzgibbon, A. (2001). Simultaneous linear estimation of multiple view geometry and lens distortion. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, *1,* I.

Guertin, S. M. (2019). Radiation Effects on ARM Devices (pp. 18-21, Rep.). Pasadena, California: Jet Propulsion Laboratory California Institute of Technology.

Hirschmuller, H. (2008). Stereo processing By Semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *30*(2), 328–341. https://doi.org/10.1109/tpami.2007.1166

Howard, A. (2008). Real-Time Stereoscopic Visual Odometry for Autonomous Ground Vehicles. *IEEE International Conference on Intelligent Robots and Systems*.

Howard, T. M, Morfopoulos, A, Morrison, J, Kuwata, Y, Villalpando, C, Matthies, L, & McHenry, M. (2012). *Enabling continuous planetary rover navigation through FPGA stereoscopic and visual odometry*. 1–9. https://doi.org/10.1109/AERO.2012.6187041

Kendoul, Farid, Nonami, Kenzo, Fantoni, Isabelle, & Lozano, Rogelio. (2009). An adaptive vision-based autopilot for mini flying machines guidance, navigation and control. *Autonomous Robots*, *27*(3), 165–188. https://doi.org/10.1007/s10514-009-9135-x

LaBel, K. A. (2009). Proton Single Event Effects (SEE) Guideline (p. 2, Rep.). NASA Electronic Parts and Packaging Program.

Maimone, M., Cheng, Y., & Matthies, L. (2007). Two years of Visual Odometry on the Mars Exploration Rovers. *Journal of Field Robotics, 24*(3), 169-186. doi:10.1002/rob.20184

Mo, J., & Sattar, J. (2019). Extending Monocular Visual Odometry to Stereoscopic Camera Systems by Scale optimization. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. doi:10.1109/iros40897.2019.8968272

Moravec, H. (1976). *Cart Project Progress Report* (pp. 1-13, Rep.). Stanford, CA: Stanford University.

Rieber, R. (2018). AutoNavigation: Intuitive Autonomy on Mars and at Sea The mobility system for Mars-2020 (pp. 12-14, Rep.). Pasadena, California: Jet Propulsion Laboratory California Institute of Technology.

Usenko, V., Engel, J., Stuckler, J., & Cremers, D. (2016). Direct visual-inertial odometry with stereoscopic cameras. *2016 IEEE International Conference on Robotics and Automation (ICRA)*. doi:10.1109/icra.2016.7487335

Wang, J., Cohen, P., & Herniou, M. (1992). Camera Calibration with Distortion Models and Accuracy Evaluation. *IEEE Transactions On Pattern Analysis and Machine Intelligence, 14*(10).

Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22*(11), 1330-1334. doi:10.1109/34.888718

Zhang, Z. (2009). CAMERA CALIBRATION.