THE UNIVERSITY OF WESTERN AUSTRALIA

FINAL YEAR PROJECT

Computer Vision Assisted Unmanned Aerial Vehicles

Author:
Brett DOWNING

Supervisors:
Prof. Thomas BRAUNL
Chris CROFT

October 26, 2015

Abstract

Multirotors are here to stay, and may soon be expected to interact in an human environment. Commodity quadcopters are advertising capabilities to act as chase-cams and turn-key mapping solutions, but none of the current generation commodity UAV chase-cams offer computer vision driven or even assisted flight modes to improve tracking, image framing or obstacle avoidance. Such vision assisted routines would also apply to autonomous or semi-autonomous inspection tasks for fixtures in remote or hazardous environments.

In this project, we re-designed and re-built the hexacopter platform inherited from previous year groups and implemented turn-key waypoint navigation and failsafe methods using the Ardupilot software stack. Using this platform, we developed and tested computer-vision driven object tracking and navigation routines using limited computational resources. The aim being to integrate vision assisted behaviours into future low-cost, lightweight UAVs.

Contents

1 Introduction

Recent developments in power and control electronics has allowed small, consumer level Unmanned Aerial Vehicles (UAVs) to lift enough processing capacity to navigate at least partly by computer vision.

The University of Western Australia (UWA) hexacopter development programme commenced in 2013. It provides a UAV platform with which students can develop and test appropriate technologies in a structured format. The hexacopter was initially built by the 2013 project group.

The 2015 programme team-members were Jeremy Tan [1], Richard Allen [2] Manish Mohanty [3] and Brett Downing [4]. The team agreed to design and execute individual projects as interdependent modules within the overall programme.

Tan [1] developed a library of feature detection and extraction routines and assessed the optimisation potential for each on the single board computer of our platform.

Allen [2] investigated means of implementing collision avoidance on a UAV, developing a robust algorithm to find the shortest detour around an exclusion zone, and experimenting with low-confidence occupancy grids.

Mohanty [3] worked on Capture and Tagging of images to suit proprietary online 3D reconstruction algorithms, and the generation of paths with which to scan objects and fields.

I (Brett Downing) [4] developed the computer vision driven chase-cam behaviour described in this report, using the optimised Computer Vision routines under active development by Tan [1].

All Motion Control routines in the code base were designed keeping in mind that they must eventually be integrated with the collision avoidance and geo-fencing routines developed by Allen [2]. This included the mapping paths used by Mohanty [3].

The UAV inherited from the 2014 project team was tested and determined to be inadequate as a test-bed for the 2015 project. We re-designed and re-built the UAV, thereby greatly improving its overall capabilities and making it fit-for-purpose. The re-build involved replacement of both the flight controller and the server, and optimised the sensor package.

We brought the code-base up to a level where it is feasible to implement rudimentary Simultaneous Localisation and Mapping (SLAM) processes. We also demonstrated that low-power single board computers can perform live computer vision processes and elements of

SLAM processes with uncertainty based reasoning to track and follow moving objects. Our Computer-Vision-Only routines are by no means as robust as radio links and differential Global Positioning System (GPS), but the use of Computer Vision allows aesthetic framing of more than a wrist-worn GPS beacon. Our object tracker was demonstrated with a variety of navigation loops, some of which were not GPS dependent.

In this report, the chase-cam project is documented in two parts: re-purposing of the UAV test-platform; and development and testing of Object Tracker autonomous navigational systems. Methodology, results and discussion are covered separately for each of the Platform and Object Tracker sections.

2 Background

2.1 Motivation

A number of commercial micro UAVs are beginning to advertise the ability to act as an autonomous chase-cam [5] [6]. Almost all of these systems rely on a tracking device on the user, and many navigate entirely by GPS. While the performance of GPS is improving, the performance of a chase-cam will be limited to the update rate and fix accuracy of the beacon and drone GPS modules. It also requires a GPS lock and radio link to be achieved and maintained in both devices for the system to function at all.

The power distribution and plant monitoring sectors have recently been taking on small fleets of drones to perform routine inspection of remote, hazardous or otherwise difficult locations [7]. Many of the tasks are reasonably repetitive and well defined enough to fully automate, but cannot be navigated by GPS alone due to the poor repeatability resulting from GPS signal drift. During this project, we were approached by a mining safety startup interested in using UAVs for routine site inspections.

The UAVs we are targeting operate with small payloads, must react quickly to changes in the environment, and frequently feature camera systems. Despite falling costs of hobby and toy-grade multirotor systems, collision avoidance, object tracking, mapping and inspection are not well catered for in the current UAV market. Very few sensors are of the appropriate scale, or mass for low-cost UAVs. Ground based vehicles have an advantage in that a 1D sensor need only be swept in one axis to search for obstacles, whereas a UAV must collect data filling a volume ahead of it. About the only form of sensor capable of doing this is a 3D laser scanner which is an high-precision, high-cost, high-mass device. A simple camera collects data on at least two axes at high enough speeds to navigate a multirotor, but the

data is often difficult or computationally expensive to interpret.

With environmental data collected by an efficient computer-vision routine, the current generation of multirotor devices would be able to interact with an human environment.

Computer vision assisted control routines would permit common low cost UAVs to better frame extreme sport chase-cam film reels, automatically avoid obstacles, and stabilise GPS variances in remotely operated inspection tasks.

In this study we investigate the usability of computer vision alone to track and follow a moving target. It is hoped that this work can be used to improve the performance of camera tracking routines in autonomous chase-cam and cinematographic applications, and partially automate remote inspection applications.

Much of the technology related to multi-copters is applicable to most other forms of UAV. The vast array of tasks micro UAVs have already been applied to suggests our work may find use in: Agriculture; mapping; targeted crop dusting; cinematography; extreme-sport photography; data collection; remote observation and inspection; and hazard and environmental monitoring.

2.2 Related Work

2.2.1 Optical Search

Search and Rescue lend themselves to automation due to the difficulty in mobilising teams in remote, harsh or dangerous conditions. A UAV can be deployed quickly and commence the search operation before boots can be placed on the ground. The UAV Outback Challenge [8] is a regular competition to perform various search and rescue missions. The performance demands are deliberately set very high, and the competition frequently goes uncompleted. In 2012 CanberraUAV [9], a UAV development team from Canberra completed the search aspect of the competition. After trying a number of image recognition algorithms, the search algorithm that they flew with simply looked for the blue of the jeans of Outback Joe [10]. This was sufficient to locate the target in a 4x6 km area. This goes to show that even a minimally complex routine can be effectively applied in appropriate conditions.

2.2.2 Terrain Estimation via Optical Flow Methods

Adding sensors to allow the copter to understand its environment is a surprisingly difficult task. Conventional contact methods operate at ranges far too close for UAVs. Ultrasonics are buffeted by down-wash and most depth sensors are either too heavy or suffer under intense

light. In terms of simplicity of algorithms, biomimicry has turned up some surprising results. In 2004, a French research group applied a number of optical flow algorithms inspired by insect vision to the navigation of a small helicopter [11]. The computer vision routines were used to inform the navigation loops, following the middle of urban tunnels and reducing speed in dense clutter, without necessarily computing the distance to the obstacles. These routines were relatively expensive in terms of computational power, but extremely simple and parallelizeable.

2.2.3 Position Estimation using Stereoscopic Methods

Any measurement will have an associated uncertainty. Extracting the most information out of a collection of measurements rarely requires taking the most accurate measurements. It is possible to estimate the position of an object or feature using multiple images separated in either space or time, but making effective use of the information requires an understanding of how the uncertainties behave. Error Modelling in Stereo Navigation [12] utilizes a number of routines to estimate the position of a vehicle by tracking visible land-marks in a stereoscopic system; and notes the interaction between the geometry of the uncertainties, and the stability of the result.

2.2.4 Altitude Estimation and Odometry by Optical Flow

Elementary methods are still interesting for the sake of biomimicry. A twenty-element photo-array was demonstrated to provide sufficient resolution to control the altitude of an aircraft [13], coupling the altitude to the velocity. Dedicated, low-resolution optical flow sensors have become exceedingly cheap since the market-saturation of the optical mouse; and many commodity flight computers already include doppler information from GPS modules. Combining these elements allows a UAV to estimate the distance to the terrain [14]. These commodity sensors typically do not include circuitry for computing rotation, but given the cost of the sensors, that limitation can be overcome by combining two sensors [15]. A number of UAV systems such as ArduPilot already include support for these devices in their code-base [16]. This support also covers pitch and roll compensation and position stabilisation [16] [17].

2.3 Project History

2.3.1 2013

The 2013 team of O'Connor [18] and Venables [19] established the UAV research programme with the purchase of a DJI "Flamewheel" F550 Hexacopter with a NAZA lite flight controller. This copter was fitted and tested, and finally converted to an autonomous platform with the addition of a Raspberry Pi model B+ single-board computer and a circuit to switch the control channels from the radio receiver to the Raspberry Pi General Purpose Input Ouput (GPIO) outputs. The NAZA lite at this time did not feature telemetry outputs or waypoint inputs, but it was able to loiter in a stiff wind using a GPS fix. This team gathered location information for the Raspberry Pi using a QStar-Z GPS unit, and bearing information using an X-Sens Inertial Measurement Unit (IMU). The weight of the sensor duplication did not exceed the maximum payload capacity of the platform, but it did compromise flight-time. Under direct control from the Raspberry Pi, the drone was able to execute basic waypoint navigation.

2.3.2 2014

The 2014 team of Baxter [20], Mazur [21] and Targhagh [22] added an internet accessible web User Interface (UI) to control the various autonomous features of the original platform, displaying the flight-path of the copter and a live feed from the camera. The navigation routines were improved and extended to permit operation in the absence of reliable heading information, and the computer vision routines were extended.

3 Platform

3.1 Introduction

This project centred around an hexacopter based on the DJI F550 "Flamewheel" frame. This frame allows for a generous lift capacity, and plenty of space to fasten flight assist hardware, and represents a low cost platform so that our work can be reproduced by a sufficiently motivated hobbyist.

In this work, the flight tasks are based around two physically separate processors: the flight controller, and the server. This allows a stable, known-safe build to be maintained on the flight controller, while experimental code runs on the server. If the server fails for any reason,

the flight controller will maintain flight and engage various fail-safe behaviours without having to rely on the experimental code for flight critical functions.

The platform inherited from the previous years' team was tested and found to be inadequate for the planned development of object tracking systems. The platform was re-designed and re-built according to a different set of constraints. This involved:

- installing a new flight controller;
- minimising sensor duplication;
- inclusion of a dedicated telemetry channel;
- using an open-source simulation environment;
- refreshing the Web UI and HUD; and
- a new interface between the flight controller and the server.

3.2 Methodology

At initial handover, the flight controller was a DJI NAZA lite, and the server was a Raspberry Pi model B+. These two systems were linked by Radio Control (RC) emulation using ServoBlaster [23] to generate the Pulse Width Modulation (PWM) signals.

The NAZA lite and RC emulation for autonomous flight did not fit together well. The NAZA lite is designed as an entry level free-flight controller and does not expose any interfaces for telemetry or data acquisition. As a result, the previous teams had to duplicate the GPS and IMU sensors to make such information available to the server's algorithms.

We attempted to address the issues on the existing platform individually, but quickly realised that the improvements could only go so far with the NAZA lite, and even then would involve significant purchases of DJI accessories.

We undertook a full critical review of every aspect of the copter, and prepared a report proposing changes we thought would be sufficient. The proposal document is included in Appendix B.

Discussed here are all of the significant changes made to the platform.

3.3 Results and Dicussion

3.4 Server Upgrade

The server was upgraded to a Raspberry Pi Version 2 shortly after its release in February 2015. The Version 2 is a multi-core processor with more memory, but we still regard as a computationally starved platform. At the time of writing, the Raspberry Pi V2 has less computational power than a smart-phone in the AU$200 price bracket.

3.4.1 New Flight Controller

We swapped the NAZA flight controller for a 3DR Pixhawk [24] running the Ardupilot 3.2 firmware. This gave us a feature-rich platform with a clear upgrade path to newer versions of Ardupilot, the option to use alternative firmwares like PX4 and Paparazzi, and the option for future students to recompile the firmware with new high-performance features.

3.4.2 Telemetry

The telemetry link supports the MAVlink [25] protocol, which defines various commands from interrogating the status to setting waypoints, or even setting the instantaneous velocity vector. This enabled us keep the high-speed components of the control loops firmly inside the flight controller, and direct it with the lower speed vision tasks. This also gave us linear control inputs in International System of Units (SI) units.

3.4.3 Sensor Duplication

In our initial efforts, we broke into the data channel between the NAZA flight controller and its GPS module in order to remove the redundant QStar-Z GPS module and acquire some telemetry data from the NAZA.

The data channel for the GPS had been deliberately obfuscated by DJI, making it quite clear that this was beyond the design intent of the flight controller.

A community RC group had reverse-engineered this link, and we ported their code to the Raspberry Pi, successfully decoded the data stream and integrated the NAZA GPS and compass data into our control loops. Unfortunately, the raw GPS and compass data was not particularly useful without the accelerometer and gyro data generated on the flight controller

unit itself.

After changing to the Pixhawk [24], the coordinates, body angles, velocity, and battery state were all available over the MAVlink [25] telemetry interface. The new configuration allowed us to fully eliminate duplicated hardware, an opportunity to re-build the wiring harness, significantly reduce the weight of the craft and extend the flight time to almost twenty minutes on the original batteries and motors.

Near the end of the project, we fitted a second GPS module to the flight controller to combat the poor multipath environment of the UWA campus. With the sensor fusion and Extended Kalman Filter (EKF) in the Ardupilot firmware, we were able to fly on GPS alone in some of the worst urban canyons on the campus.

3.5 Waypoints

Waypoints were initially handled on the server despite the NAZA featuring GPS assisted loiter behaviour, and a reliable Return to Launch (RTL) failsafe feature. Generally speaking, control loops should be as short as possible, and routing them through an entirely separate processor is clearly suboptimal.

Waypoints are a well-solved problem and come as standard with every open-source flight controller on the market.

3.5.1 Control Handover

Initially, control was handed from pilot to server using relays on the RC inputs of the flight controller. The ongoing development of control loops meant that the RC switch had never been configured to surrender the throttle channel, and the server was left without altitude control; this meant that the server had continuous perturbations from the user who remained responsible for maintaining a fixed altitude by sight alone.

Using the Pixhawk [24] in guided mode, the flight controller is responsible for maintaining and changing altitude based on requests from the server over the primary telemetry channel.

The flight controller is toggled between guided and manual modes by a switch on the pilot's transmitter. If the UAV ever flies out of range of the transmitter, the flight controller drops into the RTL fail-safe mode. At the request of our supervisors, this was demonstrated almost every week.

The changes give the platform complete autonomy, and still permit operator intervention at any stage.

3.5.2 Simulation Environment

By using a fully-featured open-source flight controller, we also had access to the community developed simulation environment. The Ardupilot simulator is designed to simulate the behaviour of the flight controller against various hardware configurations and flight styles. As such, it incorporates a comprehensive inertial model and injects noise into the various sensors. We were able to compile our tests for our development machines, couple them to the simulator over a local TCP socket, and test them against the same version of the flight control firmware as we had uploaded to the physical copter.

3.5.3 Web User Interface

The server hosts a webpage used to configure and activate the autonomous features. We saw fit to refresh the Web UI and Heads Up Display (HUD), implementing a more appealing and maintainable template system, and adding more information to the display. The Web UI now works on almost all devices.

More detail on the Web UI is available in the 2015 thesis of Tan [1].

3.5.4 Wiring Harness

The RC switch was a source of considerable confusion, and wiring faults had caused power supply contention issues leading to regular server failures. After installing the Pixhawk [24], the wiring harness was completely reworked, and every major component was placed on separate regulators.

Attention was paid to the mass of the wiring harness; almost 200g of cables and adaptors came out of the copter during the upgrade.

The only custom components remaining on the copter are the power supplies, interlink cables, and a GPIO breakout board for the Raspberry pi. We are very confident that this platform can be fully replicated by any motivated hobbyist.

4 Object Tracker

4.1 Introduction

Tracking objects requires a long chain of systems to work in unison. From tracking features in single images, all the way to reconstructing and navigating the environment without losing sight of the object. By focussing on extreme sport cinematography as the end goal of the project, the bar was set deliberately high. Tracking high-speed, high-acceleration objects requires a large control bandwidth and accurate compensation for perturbations injected by the flight hardware.

At handover, the platform contained a chase-cam algorithm that used a Proportional, Integral, Derivative (PID) loop between the position of the object in the image frame and the RC inputs to the flight computer. Venables [19] notes that the system does not deal well with changes in altitude, or perturbations in pitch and roll. We found it was not able to follow an high contrast marker under modest accelerations in most conditions.

We have broken the process into clearly defined estimation steps to expose implicit assumptions, and have incorporated some features from modern SLAM processes. Our efforts resulted in an end-to-end chain of modular software, with some functions migrated onto the flight controller to shorten control loops and enhance stability.

4.2 Method

4.2.1 Position Estimation

After experimenting briefly with the existing PID loop process, we moved on to basic triangulation methods to estimate the location of an object with the assumption that it was on the ground. We estimated the height based on the GPS altitude data streaming from the NAZA GPS module, but without access to the NAZA's internal barometer, the uncertainty in height caused the navigation loop stability to vary significantly minute by minute. After upgrading to the Pixhawk [24], we had access to the data from the EKF sensor fusion in the flight controller which included barometric data. We also made significant changes to the way the flight controller was guided by the server, and this prompted a near total re-write of the object-tracking code-base.

4.2.2 Desired Relative Pose

Once the object is located in 3D space, the copter will calculate a vantage point from which to view it. This vantage point could include information regarding the terrain, the size and trajectory of the object, the field of view of the camera, locations of light sources etc. This is the point at which cinematographic style is included in the chain.

The camera position used in testing selects the nearest point at the copter's current altitude, on a cone of a given slope radiating upwards from the object. The uncertainty analysis (discussed in section 4.2.5.2) is available to this algorithm, and could be used in a SLAM system to select a pose that will most efficiently reduce the uncertainty of a given landmark's location. Computation of uncertainty-minimising poses can lend an observation priority metric to systems such as those used in cooperative robotics [26].

4.2.3 Motion Control

As of version 3.3, the Arducopter code-base has support for arbitrary velocity vectors. The first iteration of the motion control algorithm mirrored the old RC emulation code base and transmitted a velocity vector in body coordinates to the flight controller and closed the motion loop through the automation server. The long feedback loops had problems with delays and required detailed tuning. The Pixhawk [24] already has advanced automatic tuning routines and performs very well as a stand-alone autonomous platform. To further reduce feature duplication, we replaced the motion control code with a guided-mode waypoint. The automation server then simply computes and transmits the desired vantage points and yaw values to the Pixhawk [24].

The coming Arducopter 3.4 release is set to include more advanced control schemes and support for GPS-independent autonomous features. All of the motion control processes were kept in the code-base for comparisons and future work, particularly to facilitate a fully GPS-independent acceleration vector control scheme.

The command structure on the Pixhawk [24] supports dead-channel detection and can execute failsafe behaviours on loss of link. Even if our motion control system were to use an Acceleration vector output, a loss of link or server crash would engage the appropriate fail-safe, and the copter would continue flying.

4.2.4 Structure from Image

Resolving structure from image is a field unto itself. In general, the computational load of modelling the environment in three dimensions live is beyond the capabilities of super-computing clusters that would fit in the mass restrictions of most multirotors. However, SLAM systems optimise components of this field into a vast array of algorithms of varying computational power requirements. One of the critical features of modern SLAM systems is an acknowledgement that every measurement comes with some uncertainty. The treatment of uncertainty can be very simple: carrying a one-dimensional confidence value; through to very complex, where every aspect of the measurement is recorded for post-processing and non-linear couplings can be accounted for.

Some systems are beginning to feature graph-traversal techniques where observations are stored as relative links in a graph, and networks are drawn through those links to optimise against the uncertainty of various aspects of the environment. These techniques allow the landmarks to be unloaded from memory very easily permitting a small live map, and can approach the detail of full-map covariance methods by using a less-than-live graph traversal process [27].

4.2.5 Observations and Uncertainties

4.2.5.1 Observation to Object Allocation

With many observations streaming from the camera simultaneously, the algorithm must decide which observation relates to which object in memory.

We developed and tested several algorithms to differentiate objects and refine uncertainties. Most of these suffered from the objects' uncertainties falling so low that the probability of intersection between observation and model fell below reasonable thresholds and new object models were generated. Part of the problem being that our object detection is still based on colour thresholding; this generates objects of significant size. Sized objects violate the assumption implicit in the normal distributions where the probability of locating an 'object' at a zero sized point, is zero.

In order to facilitate rapid testing and expose weaknesses, the algorithm with which we have field tested deletes and regenerates the objects every cycle unless the object leaves the field of view. This makes it very clear when the object estimation code is behaving erratically.

4.2.5.2 Uncertainty Analysis

A good uncertainty model encodes everything that is known and nothing of what is unknown. Good treatment of uncertainties combines knowledge without losing information, or adding assumptions. In the case of structure from image, a system that extracts maximal information from a single camera should be automatically capable of full stereoscopy using only the uncertainty analyses that applied to the monocular case.

With this in mind, we developed a framework to manipulate vectors with uncertain geometry. Our proof-of-concept and clean implementation in written in Octave is included in Appendix A

4.2.5.3 Ellipsoidal Models with Covariance Matrices

For the sake of computational load, we have assumed that the collected data has a three dimensional multivariate normal distribution. We have used this distribution in the quadratic form which uses the inverse of the covariance matrix.

$$PDF = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} e^{-\frac{1}{2}\left((x-\mu)^T \Sigma^{-1} (x-\mu)\right)} \tag{1}$$

where

$$\Sigma = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_z^2 \end{bmatrix}, k = 3 \tag{2}$$

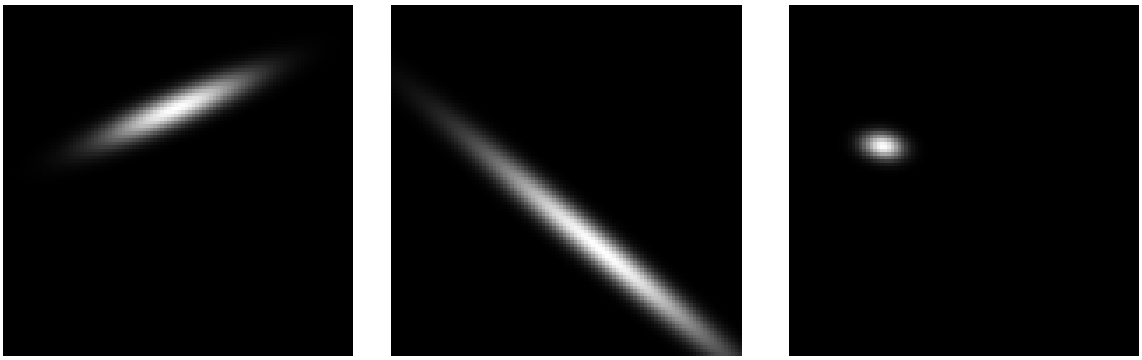
This form is particularly useful because the normal distribution behaves as a Probability Density Function (PDF) of prediction, and the combination of two observations is the normalised product of probability density functions. The product of two elliptical normal distributions is also an elliptical normal distribution.

In order to save on computing resources, we use the inverse of the covariance matrix natively C, and ignore the normalisation constant, which has a closed-form solution and can be calculated from the inverse covariance matrix directly as follows:

$$\frac{\sqrt{|C|}}{\sqrt{(2\pi)^3}} \tag{3}$$

This allows a slightly faster rasterisation of the probability density than covariance matrices, but more importantly, follows almost exactly the definition of an ellipsoid 4 and permits highly efficient geometric manipulation:

$$1 = (x - x_0)^T A (x - x_0) \tag{4}$$



(a) Measurement 1 (b) Measurement 2 (c) Combined Measurements

Figure 1: The Product of Two Ellipsoids

With no normalisation constant, the distribution always has a value of exactly one at the centroid. By permitting singular matrices, the ellipsoidal format has the flexibility to represent uncertainty distributions unbounded in length in any direction, representing rays and planes. Such distributions are extremely common as measurements; an absence of information about constraints along a given axis would just be a zero in the diagonal of the inverse covariance matrix [28]. These singular forms still combine efficiently to form non-singular localised distributions.

4.2.5.4 Combining Measurements

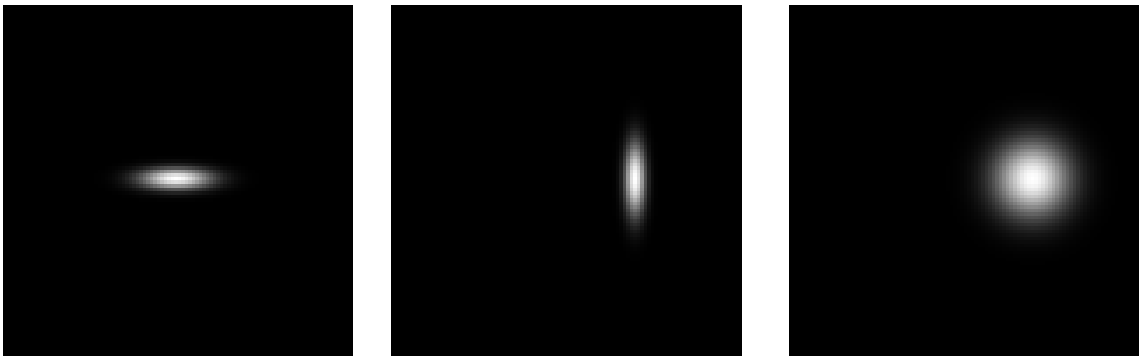
The covariance structures are well suited to use as vectors, automatically calculating the uncertainty of any combination of vectors. Multiple observations of a single point are generally combined with a weighted average, but assigning weights to superior measurements is often difficult as is a common point of discussion in Kalman Filter design.

If the subject of the measurements is known to be the same object, the object is likely to be at the centroid of the product of the probability density functions. This takes measurement accuracy into account. Using the forms described above, the product of density functions becomes the sum of polynomials.

$$C_3 = C_1 + C_2 \tag{5}$$

$$\mu_3 = C_3^{-1}.(C_1.\mu_1 + C_2.\mu_2) \tag{6}$$

Clearly apparent here, is that μ_3 is undefined if C_3 is singular. This is not a loss of generality, merely an artefact of the matrix form. If the matrix C_3 represents a ray, C_3 will have one



(a) Object at the Origin (b) Uncertain Displacement (c) Final Object Location

Figure 2: The Vector Sum of Two Uncertainty Distributions

eigenvalue of zero and μ will have a degree of freedom along the corresponding eigenvector.

This method minimises the sum of the squares of the Mahalanobis distances between the result and the initial two ellipsoids. This property of minimised Mahalanobis distances is maintained with additional ellipsoids, effectively creating a least sum of squares regression method that accounts for the covariance geometry of each measurement as successive ellipsoids are added to the mix. This least-squares regression behaviour becomes obvious in the case of spherical distributions. The successive combination of n measurements, each with an assumed variance of σ^2, will have a resultant variance of σ^2/n which is consistent with treatment of uncertainties in the one dimensional case.

In a network of observations, using this transformation destroys the assumption of independence of the observations. A full covariance matrix method would track this by generating cross terms. The Covariance Intersection described in [29] provides a weighted method of fusing observational data while keeping the uncertainty estimate conservative enough to retain the assumption of independence. The algorithm in this report can be considered as a destructive intersection, where the two observations are collapsed into one; any use of the initial observations would invoke the lost cross-terms.

4.2.5.5 Vector Transformations

Vector Sum Treating the ellipsoids as approximate displacement vectors requires the definition of vector operators. Again, the matrix form below does not permit singular matrices as inputs nor outputs, but unpacking it into polynomial form does allow suitable centroids to be chosen.

$$C_3 = (C_1^{-1} + C_2^{-1})^{-1} \tag{7}$$

$$\mu_3 = \mu_1 + \mu_2 \tag{8}$$

Again, this ignores cross terms that would be generated from a full covariance matrix. It can be thought of as a means of collapsing multiple, chained, relative observations or motion estimates into a single relative observation.

Scale

$$C' = S^{-2}C \tag{9}$$

$$\mu' = S\mu \tag{10}$$

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{bmatrix} \tag{11}$$

Where S_n represents the change of scale along any axis. The scale operator can be used to reverse the direction of a vector. This allows the definition of a discreet time derivative $\dot{x} = (x(t_1) - x(t_2))/(t_2 - t_1)$. A positional derivative defined in this way has an appropriately large velocity uncertainty distribution that increases in size with shorter time-steps, but can be averaged by intersecting multiple samples of velocity.

Rotate The ellipsoidal structure can be rotated as follows:

$$C' = R^{-1}CR \tag{12}$$

$$\mu' = R^{-1}\mu \tag{13}$$

Where R is the rotation matrix.

The rotation transformation is a major source of systematic error in the system because it doesn't model covariances in the measured rotations.

4.2.5.6 Taking Measurements

Every sensor on our platform has some resolving power which favours particular directions.

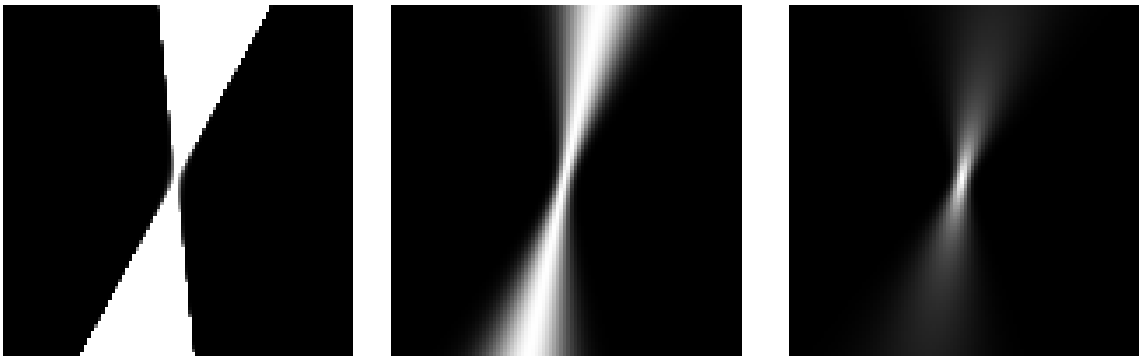
Our camera cannot locate a point in 3D space, but the direction is known within some small angular uncertainty. Our implementation takes creates a ray of some width along the Z-axis, and rotates it to where the the point of interest would lie on the focal plane in front of the camera. At this time, the ray has exactly no divergence, pending a solution to the normalisation of the ray described in 4.2.5.7.

The Lidar Lite time of flight optical range sensor has a 3 degree conical beam, a 10mm starting aperture, 10mm resolution and a small amount of sample noise. It can locate an object in 3D space, but the uncertainty model favours depth to breadth. The GPS measurements get interesting: in general we can say it has at least 3m $\sigma = 1$ radius; however, over short time periods it has a relative resolution much better than that.

4.2.5.7 The Hyperbolic Case

With a system that allows seamless representations of ellipsoids with singular matrices, it is not much of a stretch to consider ellipsoids represented by matrices with a negative determinant; the uncertainty ellipsoid becomes hyperbolic, expanding to an asymptote with an elliptical conic cross-section. This is interesting because it represents an angular uncertainty with minimal changes to the uncertainty model. A simple hyperbolic form aligned along an axis will have one negative diagonal term, and zeros for all off-diagonal terms. It is worth reiterating here that positive numbers represent increasing certainty and zero represents total uncertainty; intersecting an hyperbolic distribution with an elliptical one would suggest that the result contains less information than the elliptical distribution alone. This stems from an hyperbolic PDF requiring a normalisation function over its area. The location and size of the resulting ellipsoid is strongly affected by the density gradient of the axis of the hyperbolic distribution.

Given that hyperbolic volumes integrate to infinity, this hyperbolic probability density function would be infinitesimal everywhere. A more useful construction for its behaviour would set the density along its major axis to one (Figure 3b), or alternatively, set the integral of the cross-section to one (Figure 3c). The latter model is similar to the definition of a Gaussian laser beam profile; the total energy in the beam is proportional to its length and is therefore unbounded, but the power density is fixed for any cross section through the beam. With a single image from a camera, the only information collected about the location of an object is that it lies somewhere within a narrow cone of infinite length, ignoring exploitable imperfections in the camera such as focal length. In the case of stereo imaging, the hyperbolic measurements of points in the near field would immediately reduce to localised ellipsoids, while points in the far field would become narrow hyperbolic distributions.



(a) Negative Determinant (b) Normalised Major Axis (c) Normalised Cross Section

Figure 3: An Hyperbolic Distribution with Normalisations

The hyperbolic structure is symmetrical like the ellipse, which suggests probability behind the camera. With decent image classifiers this is unlikely to be a problem as the final localised object will fall on the intersection of the rays.

4.2.5.8 Allocating Observations to Objects

If multiple observations are made of multiple similar objects, the observations must be allocated to the appropriate object.

The observations must be ranked by some metric, and sorted. For m objects, and n observations, this takes $m \times n$ memory. In our low resolution computer vision pipeline, the number of possible visually non-unique objects is very small, so this product remains quite small in practice.

The choice of ranking metric is very much a context-dependent decision. In general though, because the ellipsoids can be interpreted as probability density functions of prediction, there is a function that defines the probability of intersection. The Bhattacharyya distance is a metric used in estimating the separability of clustered data, and makes for a reasonable first-choice of metric.

4.2.6 Assumptions

We have tried to make all of our assumptions explicitly, and have, to a large extent, succeeded. Many of the assumptions in the object localisation code have even had to be expressed in the covariance forms described above.

GPS drift has been deliberately assumed to be negligible, instead preferring to have our coordinate frame drift with the GPS signal. This was encoded as a velocity uncertainty ellipsoid encoded in all detected objects. This way, the system will operate based on what is effectively Differential GPS (DGPS) resolution, which is the case on all current chase-cam implementations anyway.

To assess the stability of the object locator, we set the observation allocator to create a new object for each observation, and assumed the object was at ground level. The assumption that the object is at ground level was defined using the same covariance model format as any other observation in the system; this time, a singular matrix defining an infinite plane at approximately zero altitude. Using a destructive Covariance Intersection described Section 4.2.5.4, locating the intersection between the ground and the ray from camera to object becomes a trivial application of uncertain geometry.

4.3 Results and Discussion

4.3.1 Simulation Tests

We used the simulator built alongside the Arducopter firmware to run our tests. This simulator is detailed enough to handle the inertial model of the platform and inject noise into the sensors. This simulator allows us to run our full software stack on any computer with a functioning compiler. The object tracking code estimates the location of the objects using data from the camera and flight controller covering the gimbal pose, the roll, pitch and yaw of the copter, and the GPS location as fused with the accelerometer and gyro data.

Because the the camera cannot resolve range, an assumption model describing the ground was added to the code reducing the columnar vision model to a spot on the ground.

Using a servo-driven gimbal on the real copter we cannot perform mechanical compensation of the pitch and roll without injecting extremely large angular uncertainties. The object estimation code compensates for the pitch and roll using matrices representing body rotations under OpenCV. This compensation is designed to cleanly counter the non-linear second order signal injection, but with the camera mechanically coupled to a laptop instead of the simulated copter, the loop becomes completely unstable.

Commenting out the pitch and roll decoupling code makes the copter very well-behaved in the simulator. Using a pose generator that attempts to place the object at a ray cast forward and down from the copter, the copter can be confidently led around the simulation environment. If the object appears behind the copter (below the centre-line of the image frame), the copter

moves forward away from the object, turning around as it does so.

4.3.2 Live Tests

The copter was able to physically follow one object, while tracking multiple others. Unfortunately, problems in maintaining calibration on multiple systems, particularly the camera gimbal, severely limited the opportunities for testing edge cases in an end-to-end system.

4.3.2.1 Navigation System

The change from RC inputs to velocity vector commands for the sake of control linearity resulted in much more predictable and consistent behaviour without significant loss of control bandwidth. To reduce the length of the feedback loop, we used the guided waypoint functions and absolute yaw commands in Arducopter 3.2 onward. This change drastically improved the flight stability of the copter, but significantly reduced the control bandwidth. The waypoint updates appeared to be obeyed at several Hertz, despite a new request being sent every frame (30Hz). The behaviour of our final navigation system was refreshingly predictable and by far the best choice for future development despite the loss of bandwidth.

4.3.2.2 Object Localisation

The observation allocation routine selects the most likely object an observation could refer to. If the associated probability is below some threshold, it instead creates a new object. In one of the tests, velocity predictions were enabled along with object creation with an inverted axis on the inertial compensation matrices. This system quickly generated more than seventy unique objects in memory and ran velocity prediction on all of them live. The algorithm's processor usage never approached the requirements for the computer vision despite running an unoptimised sorting and allocation routine, and uncertainty analysis.

4.3.2.3 Object Detection

The feature detection algorithm used in all testing of this platform was very simple colour space thresholding. In uncontrolled environments, this resulted in multiple detected features with no effective means of differentiating them. The uncertainty analysis did appear to help with this in the limited live testing it received, but the code was not sufficiently ready to draw any generalised conclusions.

Many more advanced object detection processes became available on our platform over the course of the programme thanks to the work of Tan [1], but these were in active development throughout this project and offered no consistent foundation to test upon.

4.3.2.4 Bench Marks

As mentioned earlier, the Arducopter firmwares offer a GPS driven 'follow me' mode using the Android application "Tower" [30]. The GPS tether mode was very stable and eminently usable in real circumstances. The update bandwidth was lower than our autonomous modes, and it frequently allowed the beacon to leave the frame, but it never lost lock or moved beyond the GPS tether radius. We can presume that the commodity chase-cams soon to be released will optimise for bandwidth and pay more attention to camera direction.

The GPS chase system did have some serious flaws though. In the very poor GPS environment of the Great Court on the UWA campus, the Ublox Neo7m radio was deviating more than 500m every couple of seconds. With no operator assistance, the loss of GPS led to a catastrophic failure of the algorithm and a very difficult operator intervention. Fortunately, the Ardupilot firmware permits multiple GPS modules in redundant fashion, so we were able to combat the urban canyon environment with radio diversity.

4.4 Extensions

4.4.1 Atomic Graph-Traversal SLAM

The following is a speculative description of what is potentially an entirely new SLAM process using the uncertainty analysis outlined in Section 4.2.5.

Full-state covariance SLAM has demonstrated some truly incredible results, but is limited in its applications by the large computing requirements. It has also been criticised for the way it treats non-linearity in the observations makes between landmarks.

Storing a motion history of the objects would facilitate the generation of a time-dependent network of estimated locations and relative measurements. Here, the nodes represent the instantaneous locations of the objects, linked forward in time by velocity estimates, and linked in space by relative distance measurements. Assumptions can be added to this mix in the same form as the object locations, time evolution, and relative measurements.

Building a single covariance matrix for the entire state-space generates extremely large matrices very quickly, and every algorithm that implements full-state covariance pays attention

to pruning the map in some way [31] [32].

A live process need only deal with the instantaneous positions of the robot and a small collection of landmarks. This sub-set of landmarks can be dealt with in a full-state covariance matrix [31].

The lesser 'full-state' covariance matrix can be generated by collapsing dependent chains of observations down using the vector sum, and combining network loops with the intersection into landmarks. The 'full-state' covariance matrix can be populated by choosing the resulting landmarks that best reduce absolute navigation uncertainty for the copter's current and planned locations. Further observations are still generated as atomic, relative observation structures and added to the network. These additional observations are then used to refine or regenerate the lesser 'full-state' covariance matrix by the graph traversal algorithm. This graph-traversal algorithm can operate less-than-live [27].

By storing the observations as an undirected graph of relative links, initialising the algorithm is as trivial as creating an assumption in the form of an artificial survey marker at boot. The assumption can be later removed and replaced by real survey markers if they exist, with the map fully regenerated by the graph traversal process which need not be on-board the robot.

This process still retains sufficient detail to support a bundle adjustment algorithm, optimising for consistency in cycles detected by a spanning tree algorithm [27].

5 Conclusions

We reviewed, re-designed and re-built the UWA autonomous hexacopter thereby greatly improving its overall capabilities and making it fit-for-purpose as a development platform for autonomous navigational routines. The re-build involved replacement of both the flight controller and the server. The new flight controller provided the following capabilities:

- new gimbal features for the object tracker;
- waypoint navigation;
- robust and consistent failsafe behaviours;
- extended Kalman Filter for flight dynamics; and
- live telemetry and reporting of flight parameters.

During the project, the flight controller firmware advanced from Arducopter V3.2, through V3.3rc10, to V3.3 stable near the end of the project. The enhanced server supported advanced computer vision routines. We brought the code-base up to a level where it is feasible to implement rudimentary SLAM processes. We also demonstrated that low-power single board computers can perform live computer vision processes and elements of SLAM processes with uncertainty based reasoning to track and follow moving objects. Our computer-vision-only routines are by no means as robust as radio links and differential GPS, but the use of computer vision allows aesthetic framing of more than a wrist-worn GPS beacon. Our object tracker was demonstrated with a variety of navigation loops, some of which were not GPS dependent. In commodity chase-cams, these control schemes could allow a graceful handover from radio beacon to computer vision in the event of a GPS failure.

The processes described here can be applied to almost any UAV with modest processing power, and offer additional capabilties for handling edge-case scenarios. This work brings us one step further to allowing UAVs to interact with an human environment.

6 Recommendations

The UWA hexacopter platform is reaching a level of hardware maturity where future work can focus on new algorithms and applications. Key areas for improvement mainly relate to the gimbal and camera.

The servo-driven gimbal is a large source of error and can removed entirely in favour of a fixed camera mount, or replaced by a brushless DC motor driven gimbal for accurate mechanical image stabilisation. The camera could benefit from a wider angle lens, although the narrow field of view did prompt us to find a sound solution for tracking an object beyond the frame boundary.

As discussed in section 4.2.5.7, the uncertainty analyses are incomplete. Without solutions to the hyperbolic uncertainty case or rotational transforms, the object tracker will have somewhat unpredictable accuracy. However, the system does function and can still benefit from case-specific functions to generate camera angles to suit artistic direction.

Further exploration of the hyperbolic case and singular matrices would create support for rotational uncertainties and greatly improve the tracking stability. Using a vector difference of positions to estimate the velocity, models of objects can be evolved over time and updated with additional measurements. With additional statistical analysis, this landmark velocity can be extended to a form consistent with a Kalman-style motion filter for the each visible landmark.

A problem addressed in section 4.2.5.1 is that the objects detected by colour thresholding have a non-zero size which goes against the assumptions implicit in the statistical treatment of the observations. An algorithm like Scale Invariant Feature Transform (SIFT) would track multiple points on any given object, and provide nearly unique visual identifiers for allocating observations to individual objects. While this would eliminate a small sorting routine in the object allocator, the overheads of SIFT may well require a larger computing budget.

Optical Flow is an extremely interesting avenue of research for small UAVs because the high compute-cost can be handled by dedicated silicon like a Field Programmable Gate Array (FPGA). Coupled with a covariance uncertainty analysis, optical-flow is able to resolve the relationship between relative velocity and proximity between copter and object with every frame from the camera.

As it stands, the uncertainty analysis in our implementation does not account for correlations between velocity and position, but the matrices can simply be expanded to 6×6 inverse correlation matrices instead of storing independent 3×3 matrices for position and velocity. The 6×6 matrix also allows the observation-to-object allocator to use velocity as a distinguishing feature when only a correlation between $\dot{x}$ and z is known. For sensors that cannot resolve velocity, the additional matrix elements are just zeroes.

Implementing Optical Flow with the uncertain geometry methods extended to 6×6 position and velocity covariances is very straightforward. The probability distributions of optical flow are described in the literature [33], and from there, the application is exactly the same as any other camera data; set depth-certainty to zero and rotate into place.

The current structure of the code base is sound, but makes it difficult to run processes like object tracking and object avoidance simultaneously. This situation could be improved by using a subscriber model between different processes, such as that used by Robot Operating System (ROS).

References

[1] Jeremy Tan. https://github.com/jtanx/, 2015.

[2] Richard Allen. https://github.com/20739367, 2015.

[3] Manish Mohanty. https://github.com/manishmohanty09, 2015.

[4] Brett Downing. https://github.com/brettrd/, 2015.

[5] "Lily - The Camera That Follows You." https://www.lily.camera/, 2015.

[6] "Auto-Follow Drone for GoPro Camera." https://www.airdog.com/, 2015.

[7] "UAV Inspection." http://www.ropeaccess-wa.com/unmanned-aerial-vehicle-inspection/, 2015.

[8] "Uav ouback challenge." http://www.uavoutbackchallenge.com.au/.

[9] C. Anderson, "Open source civilian uav development." http://canberrauav.org.au/, 2012.

[10] Andrew Tridgel, "2012 UAV Outback Challenge." Personal discussion at Make-Hack-Void following Linux Conf Canberra, 2013.

[11] L. Murateta, S. Doncieuxa, Y. Briereb, and J.-A. Meyera, "A contribution to vision-based autonomous helicopter flight in urban environments," *Robotics and Autonomous Systems*, vol. 50, p. 195209, March 2004.

[12] L. Matthies and S. Shafer, "Error modeling in stereo navigation," *IEEE Journal Of Robotics And Automation*, vol. 3, pp. 239–248, June 1987.

[13] T. Netter and N. Franceschini, "A robotic aircraft that follows terrain using a neuromorphic eye," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[14] S. Griffiths, "Remote terrain navigation for unmanned air vehicles," Master's thesis, Brigham Young University, 2006.

[15] J. Kim and G. Brambley, "Dual optic-flow integrated navigation for small-scale flying robots."

[16] "Documentation for the installation of optical flow sensors on the ardupilot." http://copter.ardupilot.com/wiki/common-optional-hardware/common-optical-flow-sensors-landingpage/common-mouse-based-optical-flow-sensor-adns3080/.

[17] H. Lim, H. Lee, and H. Kim, "Onboard flight control of a micro quadrotor using single strapdown optical flow sensor," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 495–500, Oct 2012.

[18] R. OConnor, "Developing a multicopter uav platform to carry out research into autonomous behaviours, using on-board image processing techniques." http://robotics.ee.uwa.edu.au/theses/2013-Multirotor-OConnor.pdf, 2013.

[19] C. Venables, "Multirotor unmanned aerial vehicle autonomous operation in an industrial environment using on-board image processing." http://robotics.ee.uwa.edu.au/theses/2013-Multirotor-Venables.pdf, 2013.

[20] M. Baxter, "Autonomous hexacopter software design." http://robotics.ee.uwa.edu.au/theses/2014-Hexacopter-GPS-Baxter.pdf, 2014.

[21] A. Mazur, "Autonomous operation and control of a multirotor unmanned aerial vehicle through 4g lte using onboard gps and image processing." http://robotics.ee.uwa.edu.au/theses/2014-Hexacopter-Interface-Mazur.pdf, 2014.

[22] O. Targhagh, "Autonomous navigation of unmanned aerial vehicles." http://robotics.ee.uwa.edu.au/theses/2014-Hexacopter-Navigation-Targhagh.pdf, 2014.

[23] "ServoBlaster source code." https://github.com/richardghirst/PiBits/tree/master/ServoBlaster.

[24] 3D Robotics, "3DR Pixhawk." http://3drobotics.com/pixhawk/, 2015.

[25] ArduPilot Community, "MAVLink." https://github.com/mavlink/mavlink, 2011.

[26] A. Bahr and J. J. Leonard, "Minimizing Trilateration Errors in the Presence of Uncertain Landmark Positions," in *3rd European Conference on Mobile Robots (ECMR)*, pp. 48–53, 2007.

[27] E. Eade and T. Drummond, "Monocular slam as a graph of coalesced observations," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pp. 1–8, Oct 2007.

[28] H. Durrant-Whyte, "Uncertain geometry in robotics," *Robotics and Automation, IEEE Journal of*, vol. 4, pp. 23–31, Feb 1988.

[29] S. Julier and J. Uhlmann, "A non-divergent estimation algorithm in the presence of unknown correlations," in *American Control Conference, 1997. Proceedings of the 1997*, vol. 4, pp. 2369–2373 vol.4, Jun 1997.

[30] 3D Robotics, "Tower (DroidPlanner 3)." https://play.google.com/store/apps/details?id=org.droidplanner.android, 2015.

[31] A. Davison, I. Reid, N. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, pp. 1052–1067, June 2007.

[32] N. Sunderhauf, S. Lange, and P. Protzel, "Using the unscented kalman filter in mono-slam with inverse depth parametrization for autonomous airship control," in *Safety, Security and Rescue Robotics, 2007. SSRR 2007. IEEE International Workshop on*, pp. 1–6, Sept 2007.

[33] E. Simoncelli, E. Adelson, and D. Heeger, "Probability distributions of optical flow," in *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference on*, pp. 310–315, Jun 1991.

Glossary

DGPS Differential GPS. 22

EKF Extended Kalman Filter. 11, 13

FPGA Field Programmable Gate Array. 27

GPIO General Purpose Input Ouput. 8, 12

GPS Global Positioning System. 5–11, 13, 20, 22, 24, 26, 31

HUD Heads Up Display. 12

IMU Inertial Measurement Unit. 8, 9

PDF Probability Density Function. 16, 20

PID Proportional, Integral, Derivative. 13

PWM Pulse Width Modulation. 9

RC Radio Control. 9–14, 23

ROS Robot Operating System. 27

RTL Return to Launch. 11

SI International System of Units. 10

SIFT Scale Invariant Feature Transform. 27

SLAM Simultaneous Localisation and Mapping. 4, 5, 13–15, 24, 26

UAV Unmanned Aerial Vehicle. 4–8, 11, 26, 27

UI User Interface. 8, 9, 12

UWA the University of Western Australia. 4, 11, 24–26

Appendix A Implementation of Uncertain Geometry

For completeness, included here is a functioning copy of the core elements of the uncertain geometry processes used in our work. This proof-of-concept code was implemented first in Octave, then ported to C++ for the live tests. This is the original Octave code.

Generation of primitives:

```
function [M,L] = GenerateBlob(x,y,z, sx,sy,sz, tx,ty,tz)
    M = [1, 0, 0; 0, 1, 0; 0, 0, 1];
    L = [0; 0; 0];
    [M,L] = StretchBlob(M,L, sx,sy,sz);
    [M,L] = RotateBlob(M,L, tx,ty,tz);
    [M,L] = TranslateBlob(M,L,x,y,z);
endfunction
```

```
function [M,L] = GenerateRay(x,y,z, sigma_x, phi, theta, tx,ty,tz);
    M = [1, 0, 0; 0, -1, 0; 0, 0, 1];
    L = [0; 0; 0];
    [M,L] = StretchBlob(M,L, 1,1/tan(phi), 1/tan(theta));
    [M,L] = RotateBlob(M,L, tx,ty,tz);
    [M,L] = TranslateBlob(M,L,x,y,z);
endfunction
```

Destructive Covariance Intersection:

```
function [M,L] = IntersectBlob(M1,L1,M2,L2)
        M = M1+M2;
        L = inv(M) * (M1*L1 + M2*L2);
endfunction
```

Transformations of primitives:

```
function [M,L] = RotateBlob(M,L, tx, ty, tz)
    Rx = [  1,        0,         0;...
            0, cos(tx), -sin(tx);...
            0, sin(tx),  cos(tx)];
    Ry = [   cos(ty), 0, sin(ty);...
                   0, 1,       0;...
            -sin(ty), 0, cos(ty)];
```

```
    Rz = [    cos(tz), -sin(tz),  0;...
              sin(tz),  cos(tz),  0;...
                    0,        0,  1];
    R = Rx*Ry*Rz;
    M = inv(R)*M*R;
    L= inv(R) * L;
endfunction
```

```
function [M,L] = StretchBlob(M,L, sx,sy,sz)
        S = [ sx, 0, 0;...
          0, sy, 0;...
          0, 0, sz];

    M =  inv(S) * inv(S) * M;
    L = S*L;
endfunction
```

```
function [M,L] = SumBlob(M1,L1,M2,L2)
        L = L1+L2;
        M = inv(inv(M1)+inv(M2));
endfunction
```

```
function [M,L] = TranslateBlob(M,L, x, y, z)
        L += [x; y; z];
endfunction
```

Sampling and rasterising functions:

```
function H = GaussianBlob(M,L,x,y,z)
        for ix = 1:x;
                px=ix-((x+1)/2);
                for iy = 1:y;
                        py=iy-((y+1)/2);
                        for iz = 1:z;
                                pz=iz-((z+1)/2);

                                X = [px; py; pz];
                                H(iy, ix, iz) = SampleBlob(M,L,X);

                        endfor
```

```
                endfor
        endfor
endfunction
```

```
function H = SampleBlob(M,L,X)
        H =  e^(- (1/2)*transpose(X-L)*M*(X-L) );
endfunction
```

The square of the Mahalanobis distance of a point from the centroid of a primitive:

```
function D = MahalSq(M1,L1,X)
        D = transpose(X-L1)*M1*(X-L1);
endfunction
```

Appendix B ArduPilot (Pixhawk) Proposal

ENGINEERING PROPOSAL

Hexacopter flight controller upgrade

24 April 2014

This document outlines the proposal for replacing and upgrading the flight controller used on the hexacopter to a 3DR Pixhawk [1] running the open-source flight control software ArduPilot [2]. This marks a significant departure from the current 'black-box' NAZA flight controller. The primary motivation for replacement is based on the strong concern that the current system is not sustainable: The current system restricts any automation improvements, provides no value to the wider community, and limits opportunities for future students.

The key benefits of switching to ArduPilot are:

- Tighter integration and feedback of the hexacopter state with the automation software;
- Avoiding sensor duplication
- Access to existing ground control and simulation software
- Well documented communication interfaces
- Wide community support
- Already being used in the research field.

ArduPilot was used in the winning entries in the 2012 and 2014 Canberra outback challenges [3, 4] and is the platform of choice in the forum DIYdrones.

The Pixhawk will use the MAVLink communications protocol to communicate with the Raspberry Pi. It is acknowledged that there are risks associated in the switchover, which will be discussed and addressed in this document.

1 Scope

The scope of this project involves:

1. Removing the NAZA-M V1 flight controller, power monitor (PMU) and GPS/Compass module
2. Removing the autonomous/manual switching circuit
3. Installing the 3DR Pixhawk, uBlox Neo-7m GPS/Compass module, power monitor, LED and wireless telemetry kit
4. Reworking the wiring harness between the flight controller and the Raspberry Pi
5. Reconfiguring the automation software to interface with the Pixhawk

2 Feature comparison

Feature	Old (NAZA) system	Proposed (ArduPilot) system
Failsafe (return to base) mode	Yes.	Yes.
Low-power failsafe	Yes, voltage monitor.	Yes, voltage and current monitor.
Auto/Manual switch	Yes (via switching circuit).	Yes, built-in to Pixhawk.
Buzzer	Yes, but not controllable. Pi has separate buzzer.	Yes, controllable. Pi can also have separate buzzer as before.
Fully autonomous (takeoff and waypoint travel)	No.	Yes. Can be programmed for fully automated missions.
IMU (pitch, roll, heading, altitude) and GPS data access	Limited (undocumented, unsupported, deliberately obfuscated interface). Current break-in cable to the GPS is unreliable and may cause damage to the GPS.	Yes. Well documented communications protocol (MAVLink).
Radio telemetry	Possible (not currently installed).	Yes.
Pan/Tilt gimbal for camera	Limited (tilt control only from the Pi).	Yes.
Simulation environment	No.	Yes, both hardware and software (HITL/SITL) with physics simulation [5]
Existing software tools	Flight assistant software performs calibration only.	1. Mission planner - handles system configuration, calibration and programming for autonomous flight [6]. 2. Tower - an Android mobile app for controlling the hexacopter [7]
Modifiable	No, closed system.	Yes. It is possible to modify all aspects of the system (open-source), but not recommended for our application without thorough testing.
Autonomous control from the Raspberry Pi	Yes, via the switching circuit and PWM output.	Yes, digital control via MAVLink (a communication protocol).
Built-in support for the PIKSI GPS	No.	Yes, but it is unclear how mature this support is at this stage.

2.1 Further discussion of features

2.1.1 Failsafe

The ArduPilot configuration has a fail-safe mode equivalent to the NAZA, activated in exactly the same way. Once configured, the flight computer will return-to-land on loss of hand-held transmitter signal. The failsafe can be configured as situations demand.

2.1.2 Modification of the flight controller

While it is possible to modify ArduPilot itself, changes to the low-level controls is not recommended nor required. This option will remain open for future work but will require extensive testing. The ArduPilot's default flight modes are sufficient for the scope of our work.

2.1.3 Camera gimbal

The pan servo cannot be controlled by the Pi due to a limitation of the NAZA. This is an out-of-the-box feature on the ArduPilot.

2.1.4 Sensor availability

The Raspberry Pi cannot access pan or tilt data as collected by the NAZA. Any pan/tilt data received will be different to the NAZA, causing the Pi to fight the NAZA. MAVLink provides access to the GPS and IMU data directly from the flight computer, which allows both the QStarz GPS and the XSens IMU to be removed. MAVLink also reports remaining battery capacity, which will allow us to write software that issues warnings and aborts missions intelligently.

2.1.5 Telemetry downlink

A new redundant data channel will permit monitoring outside range of WiFi, or in the case of server failure, which has caused near misses in the past.

The ArduPilot software supports fully autonomous missions, with instruction sent over this link. However, we expect to maintain the availability of manual override at all times and will configure the fail-safe to reflect this.

2.1.6 Control debugging

Using an external monitor interferes with the current control interface between the NAZA and Raspberry Pi. The proposed modification will save a great deal of time and effort in working around this problem.

The PWM signal from ServoBlaster must be calibrated with the NAZA flight control software. With no hardware PWM outputs, exact calibration with the NAZA flight computer is difficult. This is not an issue using MAVLink as it is a digital protocol.

2.1.7 Additional sensor compatibility

The ArduPilot community has added support for a wide variety of sensors over a wide variety of interfaces. This support includes integration into the navigation Kalman filter where appropriate.

2.1.8 Forward compatibility

DJI has released two new flight computers since the NAZA-M V1. The NAZA-M V1 is still supported, but the latest firmware updates require a CAN bus expander valued at A$80 to make use of any new features. Very few of these new features address our current concerns.

The Pixhawk was designed to replace the APM2.6 as the firmware files became too large for the 8-bit processor it carried. It was designed to be largely future-proof, boasting a generous amount of ram, flash and CPU power. Even if the ArduPilot project out-grows the Pixhawk, it is likely to have continued support and backported features, such as the APM2.6 owners currently enjoy.

2.1.9 Portability to future platforms

The ArduPilot project began with fixed wing aircraft and has since been extended to various configurations of multi-rotor, Helicopters, Rovers and Boats. Any work done building functionality against ArduCopter (the multirotor firmware) is immediately relevant to almost any other autonomous platform.

2.2 Alternative solutions

2.2.1 Upgrade the current flight controller

As a commercial system, there is *no* official interface that allows for automation or access to any of its sensors. In the past, this project achieved automation by using a remotely controlled switching circuit and the Raspberry Pi to emulate the PWM signals of joystick commands (aileron, elevator and rudder controls). This approach also requires the Raspberry Pi to have its own set of sensors, where progress would be bogged down in trying to interface with the low-level components and having to re-invent, test and tune basic control functions.

Although there is some limited upgrade capability (upgrading to the PMU V2), which would provide access to the IMU data, this is only possible through an unofficial, undocumented, reverse-engineered system. Even with this extra information, this does not address the other issues outlined in terms of community support, the camera gimbal, telemetry and the existing level of software available for the ArduPilot platform.

2.2.2 APM 2.6

The APM2.6 is a known-good ArduPilot board. The hardware is cheaper than the Pixhawk, but it is version capped as of last year. This board is not future-proof and will not benefit from new features in ArduPilot.

2.2.3 MultiWii

MultiWii is another general purpose open-source flight controller. Originally developed to use the gyroscope and accelerometer system from the Nintendo Wii game controller, it has developed into a general flight controller that operates on the Arduino platform. Community support for this platform is less than that for the ArduPilot. The platform also has lesser specs than the out-moded APM2.6.

2.2.4 Paparazzi

Paparazzi is another open-source flight controller system, which has been developed since 2003. It is an older project that appears to be extremely versatile, but not very beginner friendly. Their focus appears to be on modularity and wide applications. The community encourages significant modification of the core flight control software, which is likely to remain out of scope for this platform. Many of the demonstrated applications are single-purpose scientific flights.

2.2.5 OpenPilot

The OpenPilot appears to have a strong community backing for FPV and acrobatics, but the ArduPilot appears to have a sample of simpler flight modes and a stronger autonomous focus, and a stronger Australian community. The OpenPilot hardware does support ArduCopter, but does not have the extensive feature set of the Pixhawk, nor its sensor redundancy.

3 Risks

3.1 Risk matrix

	Very unlikely	Unlikely	Possible	Likely	Very Likely
Negligible	Very low	Very low	Very low	Low	Low
Minor	Very low	Very low	Low	Low	Medium
Moderate	Low	Low	Medium	Medium	High
Significant	Medium	Medium	High	High	Very high
Severe	Medium	High	High	Very high	Very high

3.2 Risk register

No.	Description	Probability	Impact	Rating	Mitigation	Contingency
1.1	The UAV community moves to a different platform	Unlikely	Moderate	Low	Many people, including developers have invested in ArduPilot compatible hardware which represents a small community lock-in	Find alternative, compatible flight controller software
1.2	Unforeseen hardware incompatibilities	Possible	Moderate	Medium	Conduct prior investigation and identify any potential hardware issues	Consult community for potential fix. OR replace incompatible hardware
2.2	Late deliveries or manufacturing faults	Possible	Significant	High	Request priority delivery, AND Much of the integration testing can be performed with an existing ArduPilot owned personally by a team member.	Source parts elsewhere

3.3 Further risk comments

Risk 1.1 - The UAV community moves to a different platform

The developer forums are extremely active, and new features are being added regularly. Of the open-source systems discussed, ArduPilot is the most actively developed [8, 9, 10, 11]. If an open-source flight controller is to be used, it makes sense to use the most actively developed platform to avoid project stagnation. Should ArduPilot development cease, there are a number of compelling platforms with a great deal of support - OpenPilot, MultiWii and Paparazzi.

Consequences
External development of the ArduPilot software would halt, but it would still be available in its current state. Being an open source project, many of the alternative drone software projects already support the Pixhawk hardware, just as ArduPilot has been ported to hardware from other projects.

Risk 1.2 - Unforeseen hardware incompatibilities

The Hexacopter has 30A Opto ESCs and a Futaba R7008SB receiver [12]. Support for these components with the ArduPilot has been demonstrated in other UAV projects, so hardware incompatibility is unlikely. The Raspberry Pi has been demonstrated to work with ArduPilot [13].

Consequences
Hardware incompatibilities would result in additional debugging work and lost time. It could also result in the replacement of other hardware, or the purchase of additional interfacing hardware.

Controls
Preliminary research into each subsystem of current hexacopter.

Risk 2.2 - Late deliveries or manufacturing faults

Online purchases carry the risk of components not arriving on time. It may also make it more difficult to return components, should they be faulty.

Consequences
Late deliveries would lead to lost time, potentially putting the project on hold or incurring additional costs.

4 Costs

Component	Cost ($AUD)	Source	Comments
Pixhawk	$256	3DR store	
UBlox GPS + Honeywell Compass	$41	eBay	
Telemetry	$32	eBay	Not strictly necessary, but allows in-flight full recovery after a complete server crash
Indicator LED	$12	eBay	

Total cost: A$341 (24/04/2015)

5 Conclusion

We fully expect the components listed above to serve as a drop-in replacement for the NAZA and other ancillary components.

The removal of the NAZA flight computer and the inclusion of the Pixhawk radically simplifies the wiring harness, removing many of the components and links that the 2014 team complained about being 'unreliable' and the cause of multiple crashes.

The move to the Pixhawk will greatly increase the maintainability of the platform and significantly widen the plausible scope of any future work with this hardware.

References

[1] 3D Robotics, "3DR Pixhawk." http://3drobotics.com/pixhawk/, 2015.

[2] DIY Drones team, "ArduPilot." http://ardupilot.com/, 2015.

[3] C. Anderson, "CanberraUAV (ArduPlane) wins Outback Challenge!." http://diydrones.com/profiles/blogs/canberrauav-arduplane-wins-outback-challenge, 2012.

[4] G. Mortimer, "Congratulations Canberra UAV, stunning Outback Challenge win." http://diydrones.com/profiles/blogs/congratulations-canberra-uav-stunning-outback-challenge-win, 2014.

[5] Ardupilot community, "Simulation." http://dev.ardupilot.com/wiki/simulation-2/, 2015.

[6] M. Oborne, "Mission planner home." http://planner.ardupilot.com/, 2015.

[7] 3D Robotics, "Tower (DroidPlanner 3)." https://play.google.com/store/apps/details?id=org.droidplanner.android, 2015.

[8] A. Tridgell, R. Mackay, P. Hickey, *et al.*, "ArduPilot source code." https://github.com/diydrones/ardupilot/, 2015.

[9] OpenPilot Foundation, "OpenPilot source code." https://github.com/openpilot/OpenPilot, 2015.

[10] F. Ruess, G. Hattenberger, C. D. Wagter, *et al.*, "Paparazzi source code." https://github.com/paparazzi/paparazzi, 2015.

[11] MultiWii Authors, "MultiWii source code." https://github.com/multiwii/multiwii-firmware, 2015.

[12] NoLimitLab, "Setting up Futaba 14SG and R7008SB to work with the pixhawk." https://nolimitlab.wordpress.com/resources/setting-up-futaba-14sg-and-r7008sb-to-work-with-the-pixhawk/, 2015.

[13] ArduPilot community, "Communicating with Raspberry Pi via MAVLink." http://dev.ardupilot.com/wiki/companion-computers/raspberry-pi-via-mavlink/, 2015.