Towards High Speed Aerial Tracking of Agile Targets

by

Yassir Rizwan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical Engineering

Waterloo, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In order to provide a novel perspective for videography of high speed sporting events, a highly capable trajectory tracking control methodology is developed for a custom designed Kadet Senior Unmanned Aerial Vehicle (UAV). The accompanying high fidelity system identification ensures that accurate flight models are used to design the control laws. A parallel vision based target tracking technique is also demonstrated and implemented on a Graphical Processing Unit (GPU), to assist in real-time tracking of the target.

Nonlinear control techniques like feedback linearization require a detailed and accurate system model. This thesis discusses techniques used for estimating these models using data collected during planned test flights. A class of methods known as the Output Error Methods are discussed with extensions for dealing with wind turbulence. Implementation of these methods, including data acquisition details, on the Kadet Senior are also discussed. Results for this UAV are provided. For comparison, additional results using data from a BAC-221 simulation are also provided as well as typical results from the work done at the Dryden Flight Research Center.

The proposed controller combines feedback linearization with linear tracking control using the internal model approach, and relies on a trajectory generating exosystem. Three different aircraft models are presented each with increasing levels of complexity, in an effort to identify the simplest controller that yields acceptable performance. The dynamic inversion and linear tracking control laws are derived for each model, and simulation results are presented for tracking of elliptical and periodic trajectories on the Kadet Senior.

Acknowledgements

I would like to thank Professor Steven L. Waslander and Professor Sebastian Fischmeister for their wisdom, guidance, patience and continued support throughout the last couple years. I would also like to thank my labmates who helped with this research, namely, Pie, for system identification, Carlos, for vision, Kyel, PJ, Arun and Yan for hardware development, John D., for his expert advice and Mike T., for flight tests. Outside the lab I would like to thank Steve Buchanan, for flying the Kadet Senior bare handed in the middle of winter. I would also like to thank Ryan and Sid for creating a good atmosphere of collaborative research within the lab.

Above all, I'd like to thank my family and friends for their unconditional and continued support. This would not be possible without them.

Dedication

This thesis is dedicated to my late friend Jeffrey Aho and to the peaceful use of UAV technology.

Table of Contents

List of Figures

Chapter 1

Introduction

The quest for an optimal viewing angle for a sporting event or a movie has always led to huge capital expenditures. The Romans built the Colosseum for breathtaking views of the live action directly below, unhindered by the crowds in front and behind. The modern day equivalent includes cameras, televisions and cleverly placed jigs all around the stadium or the film set. Skiing events during the Winter Olympics, for example, are filmed using extensive cable and track systems and manned helicopters. Movies as diverse as Top Gun, Beautiful Mind and 007 The World is Not Enough all have scenes that were shot using manned and remotely piloted airplanes and helicopters.

Third person perspective, a view located directly behind and above the target, remains difficult to achieve due to the high speed of the action, the rapid changes in direction and the resulting safety concerns that arise from tracking at close range, particularly with manned aircraft. In recent years, there has been a significant surge in the use of remotely piloted helicopters with stabilized cameras in the production of movies, not only for high speed chases, but for precise aerial shots that would be difficult to achieve otherwise. Currently only a handful of pilots can remotely control a vehicle precisely enough to achieve these objectives. The use of small autonomous Unmanned Aerial Vehicles (UAVs), on the other hand, may open an affordable door to such a perspective for sports networks and movie directors.

In addition to the requirement of agility and high speed, there is a fundamental need to ensure safety during high speed operations. Transport Canada strictly regulates the use of any unmanned vehicles, whether remotely piloted or autonomous, for commercial use. With more and more movies being shot in Canada, Transport Canada has devised a set of rules that govern the operation of the vehicles in crowded areas. For an autonomous vehicle

to replace a skilled ground pilot, the autopilot system has to be exceptionally safe, with very low chances of an error or a software hiccup. Two conclusions can be drawn, first, the autonomous vehicle must be highly maneuverable, able to track required trajectories without human input, and second, it must do so without compromising the safety of the crew or the target.

A complete UAV system that can aggressively follow a target at close range is not known to exist. Such a system would include a robust computer vision tracker that detects and tracks aggressive high speed ground targets. This tracker would provide reference trajectories to a highly capable autopilot that would keep the aircraft stable under extreme maneuvers. Recent advances in computing power, vision algorithms and control techniques have brought such a system closer to existence.

In 2003, Saripalli et. al. [49] successfully landed a helicopter on a marked target using vision based algorithms. The target was stationery but the research demonstrated successful use of small cameras and off the shelf computing power to collectively approach a target. In 2007, Edwards et. al. [12] demonstrate a fixed wing UAV that can land autonomously on a truck bed that is painted red. They show cases where the truck is stationary and where it is moving steadily forward at 10 to 15 mph. The vision system is implemented on an FPGA running two 400MHz CPU cores. The algorithm uses color based segmentation followed by a connected component algorithm where they find the target's center of mass. They use an off-the-shelf autopilot from Procerus Technologies. Also in 2007, Campbell et. al. [5][60], use a military Scan Eagle UAV with an onboard camera gimbal developed by the Insitu Group. They use a square root, sigma point information filter and validate results for cooperative and non-cooperative ground targets. They show large errors in tracking during sharp corners or changes in velocity. In 2008, Theodorakopoulos and Lacroix [55] develop high level kinematic controllers that rely on a low level autopilot. They limit the aircraft roll angle and its lateral velocity. It results in the aircraft circling the target in concentric circles that converge towards the target. The strategy was demonstrated to work well for stationary or slow moving targets. The authors assert that evasive and aggressive targets are an area of future work. This year, Teuliere et. al. [54], tracked a remote control car with a quadrotor using visual information. Their vision system uses color-based tracking with particle filtering. Their flight controller uses linear proportional gains on position error to successfully track the car in controlled steady motions.

This thesis presents work done towards achieving the goal of autonomous tracking of an aggressively moving ground target. It includes detailed aerodynamic models of the vehicle, system identification, vision based tracking, a controller capable of tracking aggressive trajectories, its hardware implementation on custom designed electronics and a QNX based real-time autopilot software architecture.

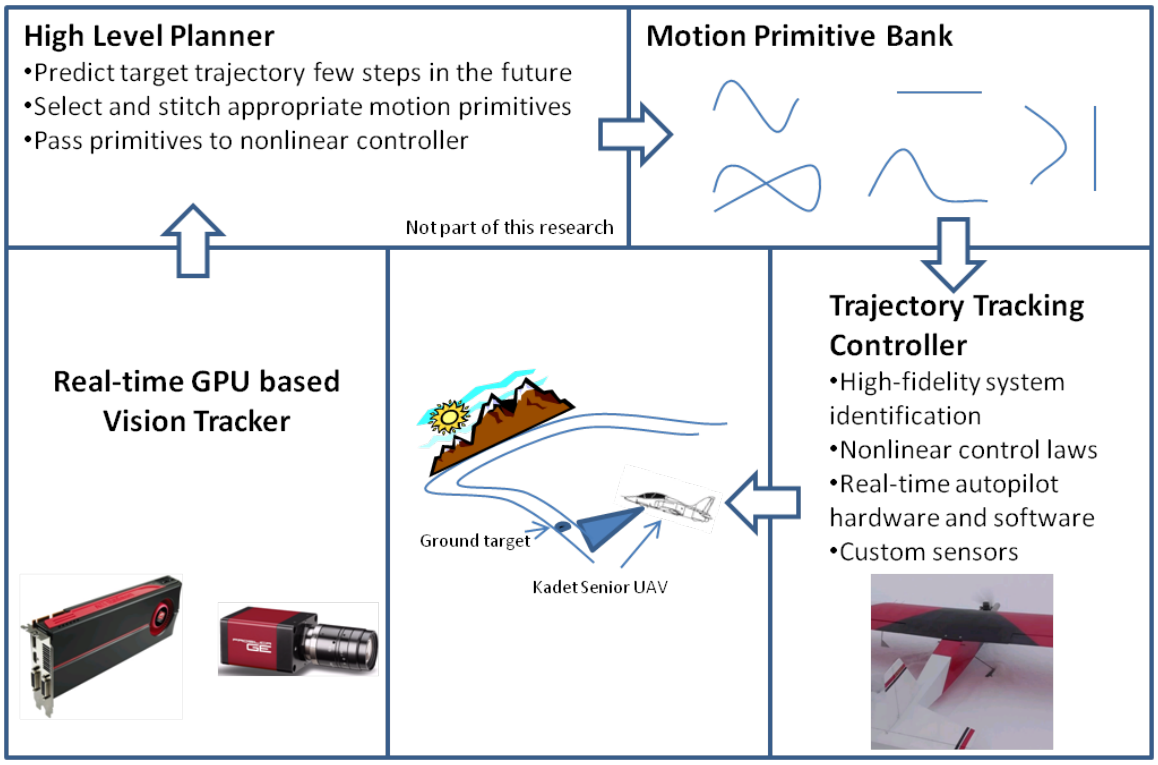

Figure 1.1: An overview of the proposed methodology and thesis contributions. Note, the high level planner is not part of this research.

A high fidelity system identification procedure is described and performed on the custom designed Kadet Senior UAV. Nonlinear control techniques are then applied to these models in simulation. Linear controls are implemented on the UAV to test various systems on the autopilot. A high speed robust vision tracking solution is also developed. These pieces will eventually be linked using a high level planner which is an active area of research at the Waterloo Autonomous Vehicles Lab (WAVE). Such a system, would extract information from the high speed video tracking, and predict the motion of the target. It would then plan an aircraft trajectory using a bank of 'motion primitives' that are designed in this thesis. Motion primitives are basic predetermined trajectories that the aircraft can follow. With sufficient motion primitives, the high level planner will be able to keep the aircraft on the appropriate trajectory to chase the target. This idea is shown in Figure (1.1).

1.1 Contributions

The main contributions of this thesis and the accompnying research project are the following:

- GPU based robust vision tracker, capable of tracking multiple targets simultaneously.
- Comparison of two different aircraft models for system identification of Kadet Senior, suitable for feedback linearization.
- A method to produce high fidelity simulation for BAC-221 using wind tunnel data.
- Application of feedback linearization to 3 progressively complex aircraft models, based on BAC-221 data.
- Simulation of the nonlinear control techniques applied to the Kadet Senior.
- QNX based autopilot, with long range communications, remote in-flight debugging, custom fast boot bios, real time characterization for crucial timing verification.

1.2 Outline of the Thesis

The thesis begins with a description of the vision tracking algorithm and implementation in Chapter 2. System identification methods and results are presented in Chapter 3 for the

Kadet Senior platform. Chapter 4 shows the design, simulation and results of a nonlinear controller which uses target trajectories from Chapter 2 and aerodynamic models from Chapter 3 to control the fixed wing Kadet Senior Unmanned Aerial Vehicle (UAV). Lastly, the main contributions are summarized and avenues for future work to complete the main goals of the project are outlined.

Chapter 2

GPU Based Vision Tracking

2.1 Motivation

The goal of this chapter is to implement an algorithm that would utilize the parallel processing capabilities of a Graphical Processing Unit (GPU) to achieve robust vision based tracking of a target. The information from the target can be used to generate reference trajectories by a high level planner, which are followed by the aircraft tracking controllers developed in the subsequent chapters.

This chapter proceeds as follows. Section (2.2) provides an overview of the current research in GPU based vision systems. OpenCL is introduced and a brief introduction to GPU programming is provided in Section (2.3). A state of the art vision based target tracking algorithm is discussed in Section (2.4) and its parallel implementation is described in Section (2.5). Lastly, tracking results and discussion of future work is presented.

2.2 Current Work in GPU Based Computer Vision

GPUs were originally meant to convert numbers into graphics for digital screens. Standards like Open Graphics Library (OpenGL) were used with languages like C for Graphics (Cg) to produce 2D and 3D computer graphics. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is mainly used for displaying graphics in CAD packages, scientific simulations, and video games.

In recent years there has been a significant surge in the performance of GPUs, outpacing even Moore's Law [8]. The latest 6 core Intel i7-980X CPU performs at roughly 107

GFLOPS[1] [57] and costs almost $1000, compared to the latest AMD HD5870 GPU that performs at 544 GFLOPS [8] and costs less than $400.

These advances have allowed a certain set of GPU users to go in reverse, i.e. graphics to numbers, opening the way to a new field known as General Purpose GPU computing or GPGPU for short. The use of GPGPU for vision processing has been steadily increasing in the last few years. It started and still continues with the use of OpenGL and Cg. Here, the programmer "fools" the GPU into thinking that it is in fact producing graphics.

In 2005, Fung, Mann and Aimone [31] demonstrated the use of OpenGL and Cg, to build an open source parallel computer vision library, known as OpenVIDIA, that ran on NVIDIA GPUs. They implemented Canny edge detection, filtering, image feature handling and image registration amongst others. GPU-KLT, a very popular parallel implementation of Kanade-Lucas-Tomasi Feature Tracker (KLT) [42] [50] was programmed using OpenGL and Cg as well by Sinha et. al. [52] in 2006. The authors showed a 20 times improvement over CPU in tracking about one thousand features at 30Hz on a 1024x768 resolution video. They also showed a 10Hz GPU based Scale Invariant Feature Transform (SIFT) [41] implementation with a 10 times improvement over an optimized CPU. Another significant effort came in 2006, with Farrugia et. al. developing GPUCV [14], a GPU library that used primitives from OpenCV and was designed to be familiar to its current users.

In 2007 Lalonde et. al. [38] used the OpenVIDIA library, which by this time had a SIFT implementation of its own, to build an eye blink detector. They were able to achieve a 10 times speedup from a CPU, and hence performed it in real time at about 25 Hz on a 640x480 pixel image. In 2008, another GPU based vision library surfaced called MinGPU [4]. This one differed from OpenVIDIA in being usable on ATI and NVIDIA hardware alike and claimed to be easier to use than OpenVIDIA. The authors implemented image derivatives, pyramid operations and 3D homography transformations between two views.

OpenVIDIA, GPUCV, MinGPU and other parallel vision libraries were developed using low level knowledge and dependence of the graphics pipeline. Compute Unified Device Architecture (CUDA) [9] was made public in 2007 by NVIDIA and changed the landscape of GPGPU programming. It allowed programmers to write C programs without the low level knowledge of thread execution. It also allowed programmers control over memory access while shielding the intricacies of low level memory handling.

CUDA is a proprietary framework and hence only works on NVIDIA GPUs. It is now widely supported in the scientific community, including its use by the Matlab's Parallel Processing Toolbox. Fung and Mann have also included CUDA in their OpenVIDIA

[1]Floating Point Operations per Second. These numbers are for double precision. For single precision the GPUs can achieve up to 2.72 TFPLOS[8]

framework [18]. Huang et. al. [24] made use of CUDA for implementing a robust motion tracking algorithm known as Vector Coherence Mapping (VCM). VCM was introduced by Quek and Bryll [48] in 1997 as a method to extract optical flow fields from image sequences. By parallelizing it, Huang et. al. show that the GPU accelerated the algorithm by 41 times when compared to the then state-of-the-art desktop CPUs. Luo and Duraiswami [43] implemented the popular Canny edge detector [6] from 1986 and showed an improvement of over 3 times for a 2048x2048 image when compared to OpenCV. They also contend, as hinted earlier, that the Canny implementations in OpenVIDIA and GPUCV are now considered obsolete. The CUDA implementation on the other hand promises to last longer since it uses a purpose built GPGPU framework that will be supported for many years by NVIDIA. In 2009, Kim, Hwangbo and Kanade implemented the KLT algorithm in CUDA [37]. They also merge rotation information from an Inertial Measurement Unit (IMU) to make the algorithm immune to rotations around the camera focal point. This fusion was previously not possible because of the increased complexity of the algorithm. The authors also mention advantages of a hybrid framework where some information is processed on a CPU. Amongst others, Li and Xiao [40] demonstrate a mean shift tracking algorithm using CUDA, implemented on a NVIDIA GeForce 8800 GTS.

Open Computing Language (OpenCL) was developed by Apple and released in 2008. It is the most recent and promises to be the most powerful of all GPGPU frameworks. It is an open standard framework maintained by the Khronos Group and is supported by AMD and NVIDIA alike. Its contributors include AMD, Intel and NVIDIA, amongst others. Similar to CUDA, OpenCL provides a C based SDK that abstracts the low level operations from the user. Unlike CUDA, not only does OpenCL run on GPUs made by both ATI and NVIDIA, it also runs on multi-core CPUs and cell processors. This is known as heterogeneous computing where the algorithm is divided into sections which are best for CPU and GPU. AMD has recently announced their Fusion platform, which is a heterogeneous computing platform that includes both a CPU and a GPU. It is also more generally known as an Accelerated Processing Unit, or APU for short. The Fusion platform is targeted for OpenCL based parallel applications that can take advantage of its CPU/GPU architecture. OpenCL's strengths are its practicality, flexibility and retargetability [28]. However, OpenCL is new and in preliminary stages of development, with only early adopters using it for GPGPU applications.

At this time vision based algorithms implemented on OpenCL are very rare. This is mainly due to the inertia that the CUDA based vision community has picked up as described above. GPGPU applications for other research have however started to surface recently. In 2010, Shimobaba et. al. [51] showed the use of an AMD HD5000 with OpenCL to produce 3D holograms at twice the speed of an equivalent CUDA based NVIDIA GPU.

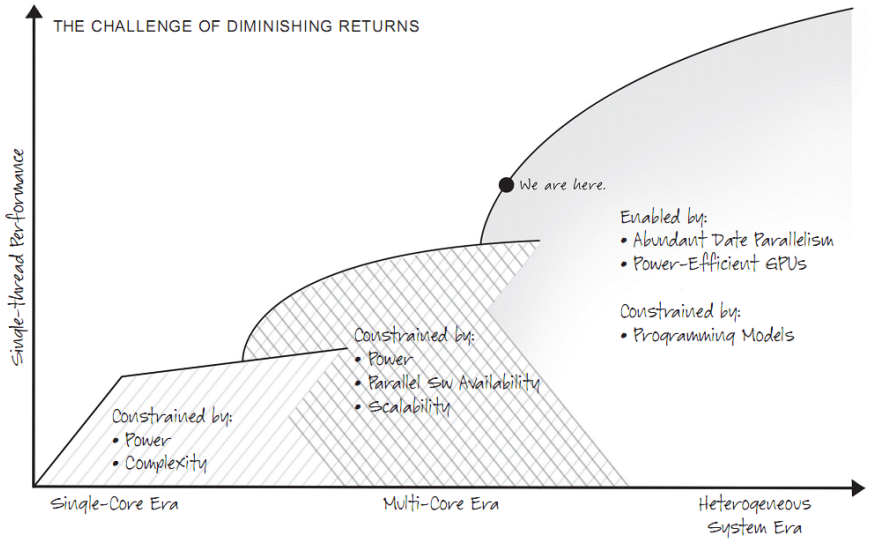

Figure 2.1: Future of Heterogenous Systems [1].

This is partly confirmed by [22], where Harvey and Fabritiis provide a tool to convert CUDA programs into OpenCL for cross platform portability as well as almost 50% improvement in performance.

In addition, documents from AMD show a promising future and support for OpenCL. Figure (2.1) shows the trend that AMD is projecting, pegging the Fusion class of processors as the direct descendants of GPUs and CPUs.

Figure (2.2) shows another reason why heterogeneous computing will eventually dominate. A GPGPU capable GPU on the same chipset as the CPU has a much faster data transfer rate to the on board memory, which is crucial for real time image analysis. Previous offerings by vendors have included Integrated Graphics Processors (IGP) on the motherboards, which use off chip system memory and slow transfer rates that bottle neck their performance. OpenCL, which supports such platforms is the only choice to adopt.

This, together with the fact that the new AMD Fusion platform is a very light weight and flyable piece of hardware, was the reason why OpenCL was selected as the basis for a vision based tracking solution for this research.

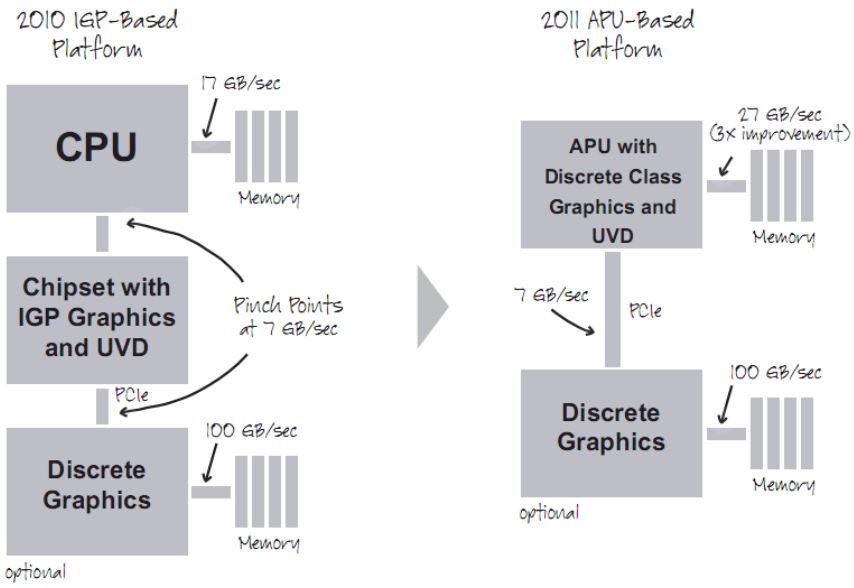

Figure 2.2: Bandwidth constraints in IGP chipsets vs an APU [1].

2.3 OpenCL Framework

As mentioned earlier, OpenCL can be used just as well over multiple cores of a CPU as it can over multiple cores in a GPU. OpenCL classifies these as compute devices. Each compute device consists of compute units and processing elements. A GPU compute device would include more processing elements than a CPU. Each processing element in a GPU however, is much weaker than a single processing element in a CPU [8]. The improved performance mentioned earlier, is achieved by a larger number of processing elements in the GPU. The latest AMD GPUs consist of stream cores which are essentially Single Instruction Multiple Data (SIMD) engines. A SIMD engine performs the same operation on multiple data simultaneously. This classification is shown in Figure (2.3).

For the purpose of this research, the latest AMD GPU at the time, HD5870, was chosen. This GPU consists of 20 compute units, each with 16 stream cores, each of which in turn have 5 processing elements as seen in Figure (2.4). This is a total of 1600 processing elements. Each processing element can execute single-precision floating point or integer operations. One of the 5 processing elements can also perform special operations like sine, cosine and logarithms. Double precision floating point operations are performed by connecting two or four of the processing elements [2]. All stream cores within a compute unit

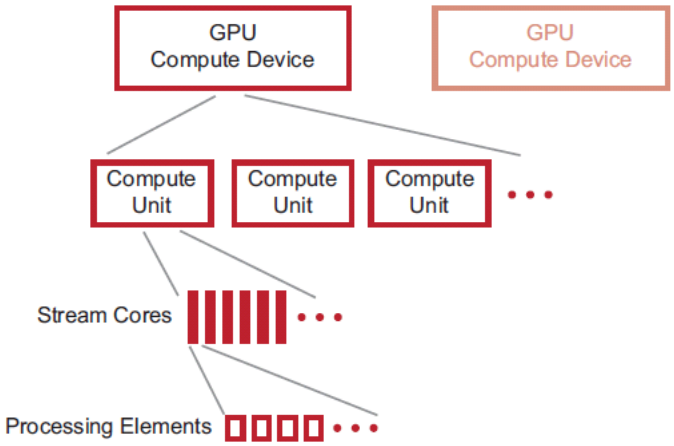

Figure 2.3: Computed devices, compute units and processing elements [2].

execute the same instruction, different compute units can execute different instructions. The instructions are programmed using stream kernels, or simply kernels. A kernel is a small, user-developed program that is run repeatedly on a stream of data. It is a parallel function that operates on every element of input streams. The GPU maps each execution of the kernel onto compute units. Arrays of input data waiting to be processed are stored in memory and are accessible by the compute units. Each instance of the kernel running within a compute unit is called a work-item. The work-items are mapped to an n-dimensional index space known as NDRange, and subdivided into workgroups. The GPU schedules the work-items on the stream cores until all have been processed. Subsequent kernels are then executed until the application completes.

Groups of work-items executed in lock-step inside a compute unit are called wavefronts. The number of work-items within a wavefront is specific to the GPU hardware. For the HD5870, up-to four work-items are pipelined to each stream core and are processed for four cycles. This is done to abstract out the low level memory latency and processing element operations from the user. With 16 stream cores per compute unit this implies a wavefront size of 64 work-items. Figure (2.5) shows a summary of this classification. The intermediate classification of wavefronts within workgroups is meant for memory management and for more control over their execution.

The NDRange can be a grid of one, two, or three dimensions. This selection is dependent on the user, and is usually dictated by the size of the output. For example, for a W x H pixel intensity image input, with a Canny edge detection kernel, the output will also

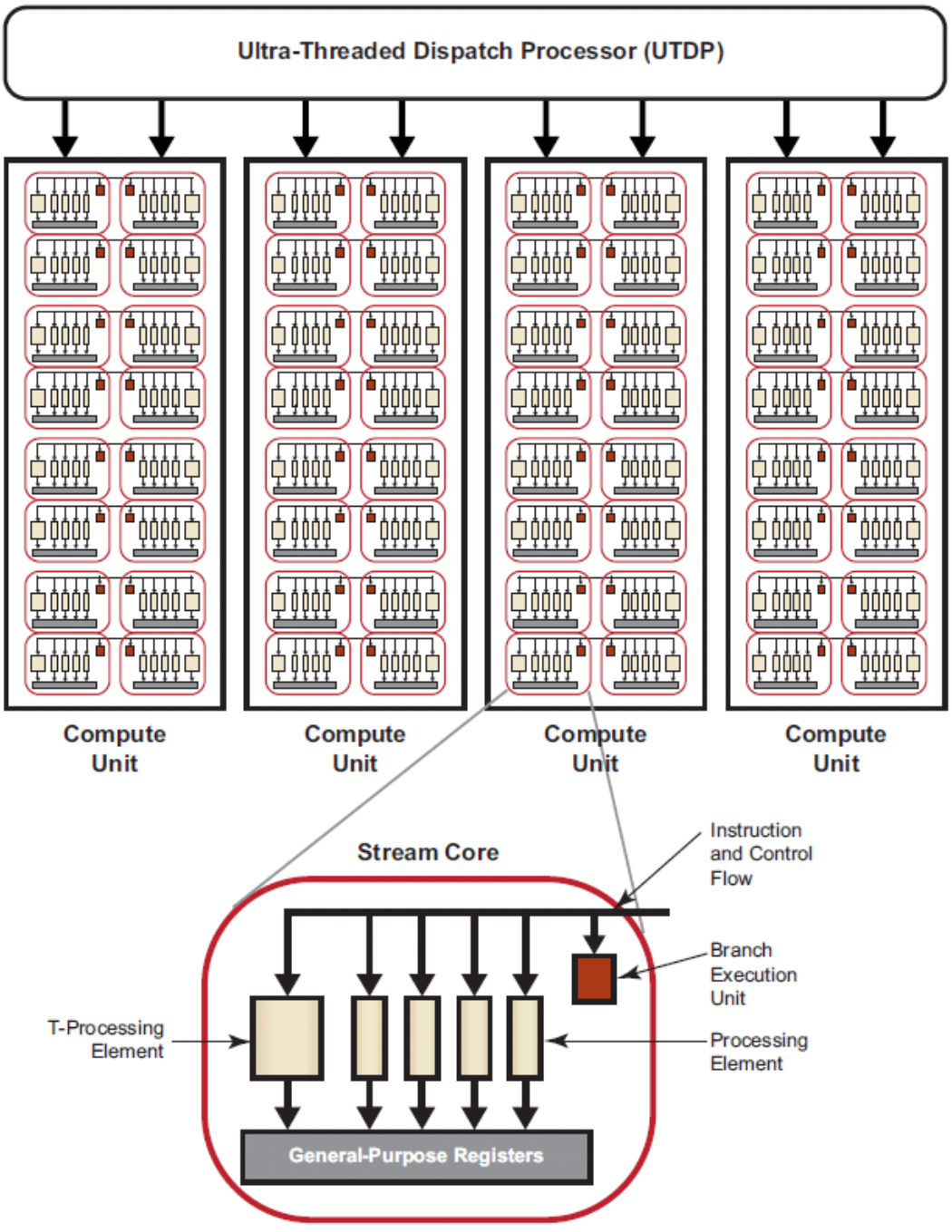

Figure 2.4: Block diagram of an AMD GPU compute device [2].

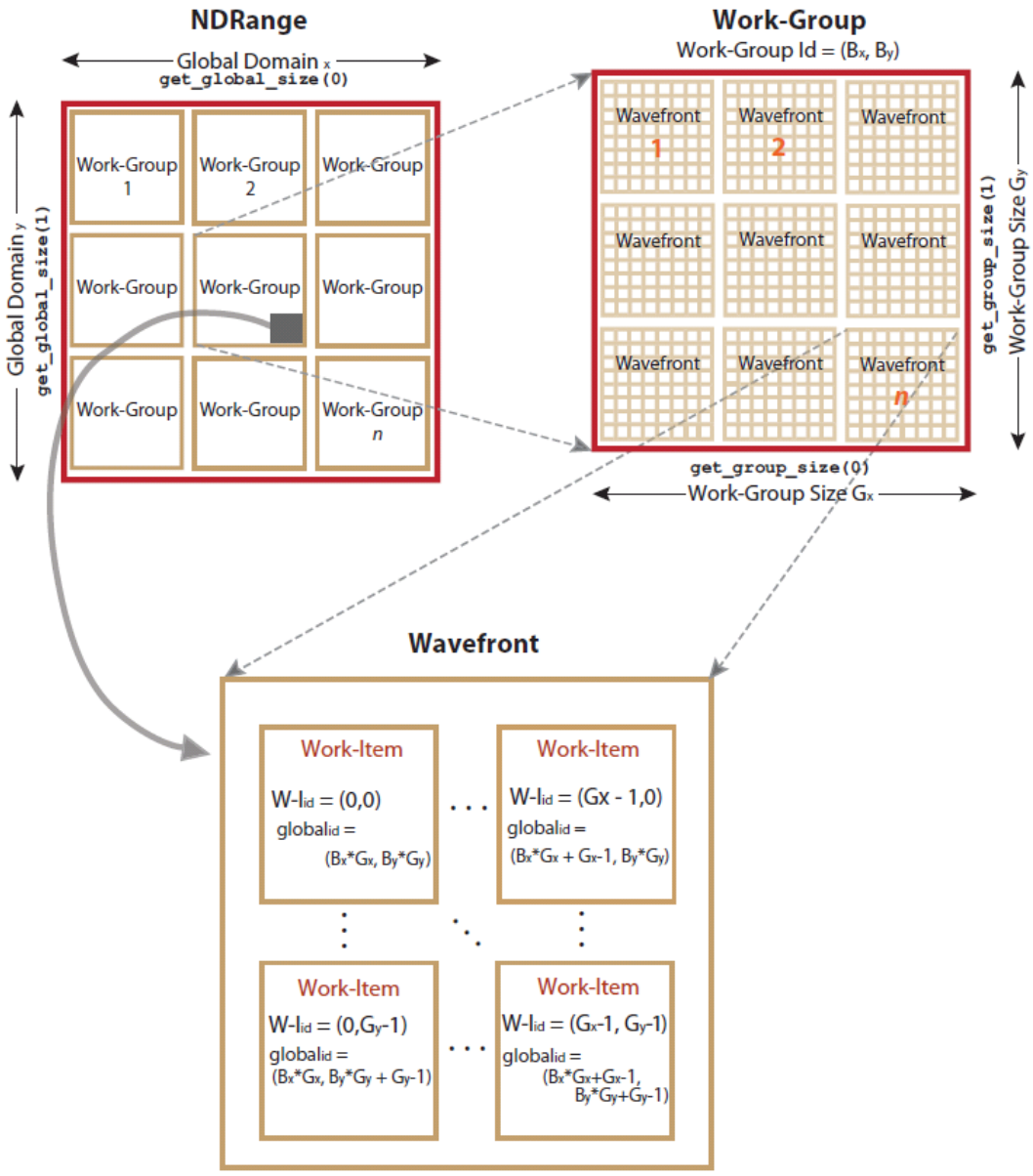

Figure 2.5: NDRange, Work-Group, and Work-Items [2].

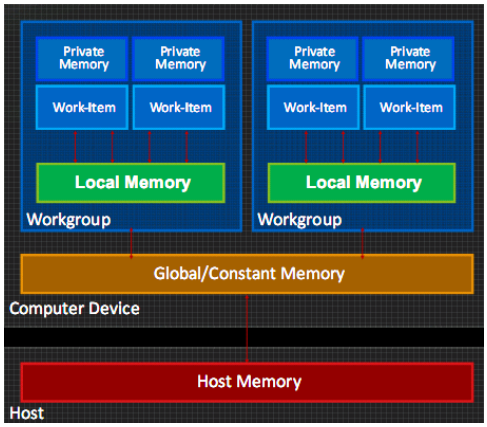

Figure 2.6: OpenCL Memory Structure [27].

be a W x H pixel image. W x H intensity values translate to a float or double matrix of size W x H. This can be represented by either a one dimensional array of size 1 x WH or a two dimensional array of size W x H. The choice depends on the user, and will be the size and dimension of NDRange. The number of workgroups, and hence work-items per workgroup can also be selected by the user or left for the GPU to decide. The choice should result to maximize the use of compute units. For example, a good choice would be one that results in 64 work-items per wavefront, because this ensures that the entire compute unit is utilized.

Figure (2.6) shows the classification of memory in OpenCL. Private memory is specific to a work-item. It is not visible to other work-items. Local memory is specific to a work-group, accessible by work-items belonging to that work-group. Global memory is a read/write memory that is accessible to all work-items in a context. Constant memory is a read-only region for memory items that are passed from the host and are not changed during the kernel's execution (for example, the input image for Canny edge detection above). Host memory is the CPU accessible memory for an application's data structures and program data.

For the HD5870, accessing the 32 kB of local memory (8 bytes/cycle) per compute unit is an order of magnitude faster then accessing the 1 GB of global memory (0.6 bytes/-cycle) [2]. Private memory is fastest at 48 bytes/cycle. However, private memory, which is physically stored in hardware General Purpose Registers (GPR) is limited to 256 kB per compute unit. This translates to 5120 kB per GPU. When executing 64 kernels per compute unit this implies 4kB of memory that can be manipulated within a kernel. In

the event that the GPR fills up, private memory is mapped to a speacial region known as "scratch", which has the same performance as global memory.

This is one of the main design considerations when programming kernels. If values can be hard coded, they should be, if loops can be unrolled to save the memory required for a counter, they should be [2]. This can be problematic in research applications where image sizes or other variables need to be changed for experimentation.

The kernel is compiled at runtime by the OpenCL compiler every time the host program is executed. During execution, memory from the host (for example, an image intensity matrix) is transferred through a buffer and into the appropriate memory of the GPU. At the end of each kernel, the appropriate result is mapped to the corresponding location on an output array. Once all the kernels are finished executing the output array is passed through the buffer and to the host memory for the CPU.

A simple example for vector addition can demonstrate these concepts. **vec_add_CPU** is a program that would normally be used to add two 1-d vectors, a and b, with the result being written to vector c. When the vector size, n, is small the CPU is fairly efficient at looping through each element, adding and assigning the results.

```
1 void vec_add_CPU(int n, const float *a, const float *b, float *c)
2 {
3   int i;
4   for (i=0;i<n;i+-)
5     c[i]=a[i]+b[i];
6 }
```

However, when n becomes large, say 128,000 elements, a parallel execution method is more justified. This is shown in **vec_add_GPU**. This is the kernel. Before the kernel is executed, the a and b matrices are passed into the constant section of the global memory. Output array c is also passed to the global memory, but not as a constant, since it needs to be written to. The NDRange is one dimensional with a size of 1 x $128,000$ work-items (the size of output vector c). The kernel is compiled and also transferred to the GPU.

Each instance of the kernel's execution (work-item) will call **get_global_id(0)** which will let it discover its position in the NDRange and hence its position in the output vector c. It will perform the addition of two corresponding elements from vectors, b and c, and return the result to the appropriate position in vector c, as shown in Figure (2.7). Once all work-items are done, the vector c will be full, and will be passed through the output buffer and into the host memory, where the host application on the CPU will read and display the result.

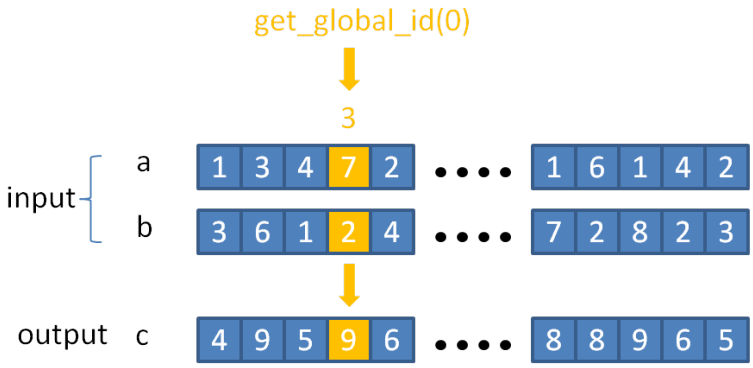

Figure 2.7: Parallel vector addition kernel.

```
1 kernel void vec_add_GPU ( global const float *a,
2          global const float *b,
3          global float *c)
4 {
5   int gid= get_global_id(0);
6   c[gid]=a[gid]+b[gid];
7 }
```

There are no loops required in this implementation. 40 workgroups can be chosen with a workgroup size of 3200 work-items. This results in 50 wavefronts per workgroup with the optimum 64 work-items per wave-front. With 20 compute units, this results in $50 * \frac{40}{20} = 100$ sequential computations vs 128,000 as in the case for a CPU.

2.4 Target Tracking Algorithm

Vision based target tracking is the technique of finding regions corresponding to given objects in a sequence of image frames. The objects can move, change their pose, occlude for a portion of time, merge with each other or display other erratic behavior from frame to frame.

Covariance based tracking was developed at Mitsubishi Electric Research Laboratories by Porikli and Tuzel [47]. They use a covariance matrix to group a number of image features together. They claim that the covariance matrix is an efficient descriptor of features because it is enough to match the region in different views and poses and is

low dimensional compared to other descriptors. They are also scale invariant and are mostly immune to ordering and inherit a certain rotation invariance [58]. To locate the target, a predetermined covariance matrix is compared against the covariance matrices of all prospect regions within a frame.

The algorithm can be used to track multiple targets from a single non-stationary camera. The camera can be monochrome, color or hyper-spectral. It has been shown to be better than mean-shift, template matching and sub-window methods [58]. However, the computation power required has so far limited its applications in real-time tracking, necessary for robotics applications.

Until now, most attempts around optimizing this algorithm were focused on novel search methods, limited search regions, target motion models and Kalman filters [46]. While helpful, each attempt at optimizing tried to reduce the series computation load on a CPU. However, what was needed and what could prove to be a much higher return on investment, is the implementation of this algorithm on a GPU. This thesis shows such an implementation for the very first time using the AMD HD5870 GPU, which was described in the previous section.

Porikli et al. [47] show that a 320 x 240 image can track a target at roughly 600 msec/frame when the algorithm is implemented on a 3.2 GHz Pentium CPU. They show the complexity to be $O(320 \text{ x } 240 \text{ x } d^2 + 320 \text{ x } 240 \text{ x } d^3)$. In 2006 Porikli [46] suggested that this algorithm could benefit greatly from a parallel implementation on an FPGA, DSP or a GPU.

2.5 Algorithm Description and Implementation

An adaptation of the covariance tracking algorithm is described below together with its parallel implementation on a GPU. For the purpose of this thesis, consider an image, defined by a W x H sized intensity matrix, $I^v \in \mathbb{R}^{WXH}$ [2]. This matrix includes the intensity values from a W x H pixel, monochrome image frame, acquired from a small camera. Features of interest within this matrix can include intensity values, image gradients, edge magnitude, edge orientation, multi-spectral intensity values (like from infrared cameras) and filter responses. A W x H x d feature matrix F^v can be extracted from I^v where d is the number of features per pixel.

$$F^v(x^v, y^v) = \phi^v(I^v, x^v, y^v) \tag{2.1}$$

[2] The use of the v superscript here and elsewhere on variables is meant to differentiate them from similar variables used in aircraft models and controllers in later chapters.

where (x^v, y^v) are the coordinates of a single pixel, and $\phi : \mathbb{R}^{WXH} \rightarrow \mathbb{R}^{WXHXd}$ is a function that maps the pixel location to its features. The target to be tracked is an unknown region within the image, I^v, that is defined as a collection of these specific features. It can be defined at the start of the video input, by drawing a rectangle around something of interest, like a car on the highway. The video input can be a stream coming from an airborne camera. The target is confined or can be sufficiently described inside a m x n region around the pixel coordinates (x_t^v, y_t^v).

For any m x n region, $R \subset I^v$, a total of $m * n$, feature vectors can be extracted. For the target tracker used in this work, the following seven features are selected:

$$f_k = \left[\begin{array}{ccccccc} x^v & y^v & I^v(x^v, y^v) & I_x^v(x^v, y^v) & I_y^v(x^v, y^v) & I_{xx}^v(x^v, y^v) & I_{yy}^v(x^v, y^v) \end{array} \right] \tag{2.2}$$

where $k = 1 \ldots mn$ and

$$I_x^v = \frac{dI^v(x^v, y^v)}{dx^v}, I_y^v = \frac{dI^v(x^v, y^v)}{dy^v}, I_{xx}^v = \frac{d^2I^v(x^v, y^v)}{d(x^v)^2}, I_{yy}^v = \frac{d^2I^v(x^v, y^v)}{d(y^v)^2}. \tag{2.3}$$

Each of these features can be found numerically using central difference approximation. The mean of these features for all the pixels is

$$\mu_R = \frac{1}{mn} \sum_{k=1}^{mn} f_k \tag{2.4}$$

where mn is the total number of pixels in R. A d x d feature covariance matrix for R can now be defined in the usual manner,

$$C_R = \frac{1}{mn} \sum_{k=1}^{mn} (f_k - \mu_R)(f_k - \mu_R)^T. \tag{2.5}$$

Within the matrix, I^v, two distinct types of region R can be defined: ones that are being searched for the target features and the one that best describes the target features. These can be represented as R_p and R_t respectively, where p stands for a prospect region and t stands for target region. Feature covariance matrices can be found for both R_p and R_t using Equations (2.2) to (2.5).

The matrix, I^v, includes $(W - m)$ x $(H - n)$ regions classified as R_p. One of these regions is the location of the target, provided that the target is in the frame. If a comparison can be made between the feature covariance for every R_p to the predetermined feature covariance of the target then it should be possible to identify the location of that target.

Despite all their aforementioned advantages, covariance matrices are not elements of the Euclidean space, and therefore require special methods to compare and evaluate. Forstner and Moonen [15] presented proofs of an optimal distance metric, that can be used to compare covariance matrices of arbitrary dimensions. The authors show that their proposed distance, d, between two covariance matrices, A^v and B^v, is

- invariant with respect to affine transformations of the coordinate system,
- invariant with respect to an inversion of the matrices,
- indeed a metric by proving positivity: $d(A^v, B^v) \geq 0$, and $d(A^v, B^v) = 0$ only if $A^v = B^v$, symmetry: $d(A^v, B^v) = d(B^v, A^v)$ and the triangle inequality: $d(A^v, B^v) + d(A^v, C^v) \geq d(B^v, C^v)$, where C^v is another covariance matrix.

Their distance metric is defined as

$$d(A^v, B^v) = \sqrt{\sum_{i=1}^{d} \ln^2 \lambda_i(A^v, B^v)} \tag{2.6}$$

where d is the dimension of the covariance matrices and $\lambda_i(A^v, B^v)$ is the ith eigenvalue found by solving a generalized eigenvalue problem

$$det(\lambda B^v - A^v) = 0. \tag{2.7}$$

The smaller the value for d, the "closer" the two covariance matrices are to each other. In other words, this metric is a good way to compare whether a prospect region, R_p, in the image is similar to the target region, R_t. Proofs of these claims can found in [15].

The most computationally intensive calculation involves a Cholesky decomposition and a Jacobi cyclic method implementation, required to solve the generalized eigenvalue problem in Equation (2.7). This computation is repeated for every pair of R_p and R_t. In the intensity matrix, I^v, there are $(W - m)$ x $(H - n)$ pairs, if the entire image is being searched. These calculations need to be repeated for every frame. A typical 640 x 480 camera, with 20 x 20 sized R_p regions, would result in 285,200 search pairs. For real-time tracking, this will need to be done live for every received frame.

In order to parallelize this algorithm, certain calculations are best run on the CPU, while the rest are best run on the GPU. The GPU, while consisting of a massive number of cores, also has lower computation capabilities per core when compared to a CPU. Hence,

a calculation that only needs to happen once per frame should remain on a CPU while the frequently repeated calculations can be transferred to the GPU.

For finding the target within the intensity matrix, I^v, the NDRange is set as $(W - m)$ x $(H - n)$, the dimensions of the output as described above. This way, every m x n sized region, R_p, can be assigned to a work-item, where the common instruction set would be to solve the generalized eigenvalue problem and find the metric d to R_t. The size of the workgroup can be chosen based on the discussion above. For a 640 x 480 image, with 20 x 20 regions, a one dimensional NDRange is 1 x 285,200. A workgroup size of 2560 work-items will result in 112 workgroups, with an optimum 64 work-items per wavefront. The last workgroup will have $(112 * 2560) - 285,200 = 1520$ idle work-items. For 20 compute units, this results in $40 * \frac{112}{20} = 224$ sequential computations.

The distance of each, R_p, R_t, pair can be sent back to the host memory (Figure (2.6)) where the smallest d is assigned the most probable location of the target.

The generalized eigenvalue problem from Equation (2.7) is solved by first reducing it to the standard eigenvalue problem and then using the Jacobi-Cyclic method. It can be re-expressed as

$$A^v\mathbf{v} = \lambda B^v\mathbf{v}, \tag{2.8}$$

where $\mathbf{v}$ is an eigenvector that corresponds to an eigenvalue λ. B^v can be decomposed into a product of an upper triangular matrix, U, and its transpose by using a Choleski decomposition,

$$B^v = U^T U. \tag{2.9}$$

Substituting in Equation (2.8),

$$A^v\mathbf{v} = \lambda(U^T U)\mathbf{v} \tag{2.10}$$

and multiplying both sides with $(U^T)^{-1}$,

$$(U^T)^{-1}A^v\mathbf{v} = \lambda U\mathbf{v} \implies (U^T)^{-1}A^v U^{-1}\mathbf{v} = \lambda\mathbf{v} \tag{2.11}$$

and then substituting, $D = (U^T)^{-1}A^v U^{-1}$, gives the standard eigenvalue problem,

$$D\mathbf{v} = \lambda\mathbf{v}. \tag{2.12}$$

This problem can now be solved using the Jacobi Cyclic method [19] to give eigenvalues. The Jacobi Cyclic method is a well known iterative method for finding eigenvalues and

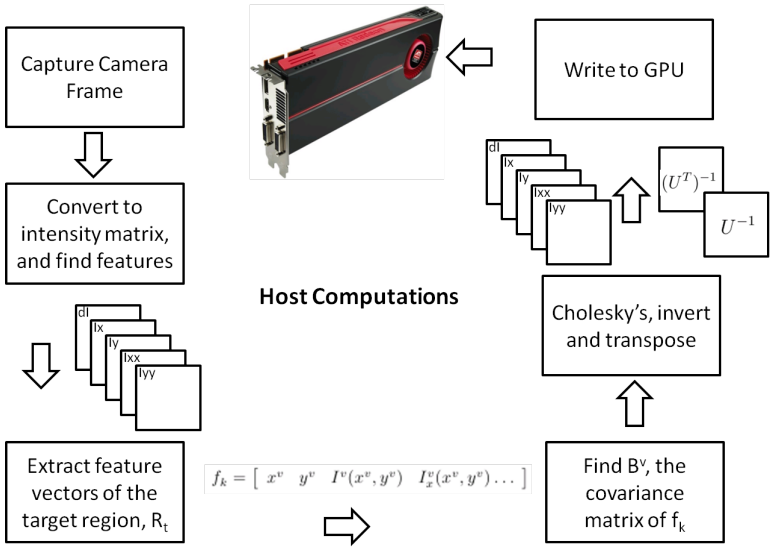

Figure 2.8: A brief overview of the calculations on the host, and transfer of required matrices to the GPU. Once the GPU finishes its computations, the host is also responsible for reading back the result and evaluating the position of the target.

eigenvectors of real symmetric matrices. If B^v is assigned the covariance matrix of R_t then the Choleski decomposition and the required matrix inverse can be performed once on the CPU. This gives the d x d covariance feature matrix defining the target, as well as the intermediate matrices (U^{-1} and $(U^T)^{-1}$) required for calculating the eigenvalues. The matrix A^v is assigned the covariance matrix of one of the R_p regions, depending on which instance of the kernel it is called from. The matrix multiplication in Equation (2.11) and the Jacobi Cyclic method (not shown here) is repeated for every R_p within the kernel.

A simplified block diagram of these operations is shown in Figures (2.8) and (2.9). Each instance of the kernel has access to the matrices that are computed on the host CPU through the GPU constant memory. These include, dI, the main image intensity matrix, Ix, Iy, Ixx and Iyy, which are the outputs of appropriate feature operations on the intensity matrix (from Equation (2.3)) and U^{-1} and $(U^T)^{-1}$ which are outputs from the above calculations. The feature operations are performed on the CPU because algorithms for those already work in real-time. If done in parallel on the GPU, the work-item barriers required to synchronize their intermediate steps would slow down the main kernel [2]. The constant memory allows accessible fast read-only memory so there are limited memory issues on accessing the multiple input matrices.

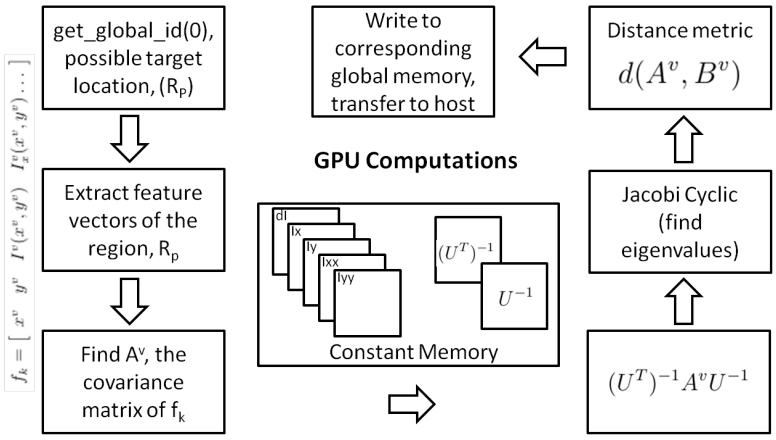

Figure 2.9: A brief overview of the calculations on the kernel, and the transfer of results to the host. This set of calculations is repeated for every work-item.

Once all the individual comparison results are calculated, they are transfered to the host memory (Figure (2.6) where the smallest distance is picked as the most likely target location.

2.6 Results

Screen shots from the results can be seen in Figures (2.11) and (2.10).

The first example tracks a set of keys on a key-chain through rotations and transformations. Part of the set is selected as a target, with no particular identifying color or structure. The algorithm maintains real-time tracking even when the keys have been picked and shuffled. It also differentiates the key set from a shoe that looks very similar in color and arrangement. The key is also obscured and then made visible again, the tracker automatically reacquires the target.

The second example tracks a high speed aircraft while it is flying at over 80 km/h, with aggressive maneuvers and sharp turns. The aircraft also comes very and then flies off far, with the vision system maintaining tracking at all times through the scale changes.

Porikli et al. [47] implied that for a 640 x 480 image, the computational load on the CPU would cause it to run at 1,200 msec/frame. The GPU implementation described here performs at roughly 100 msec/frame - an improvement of over 10 times.

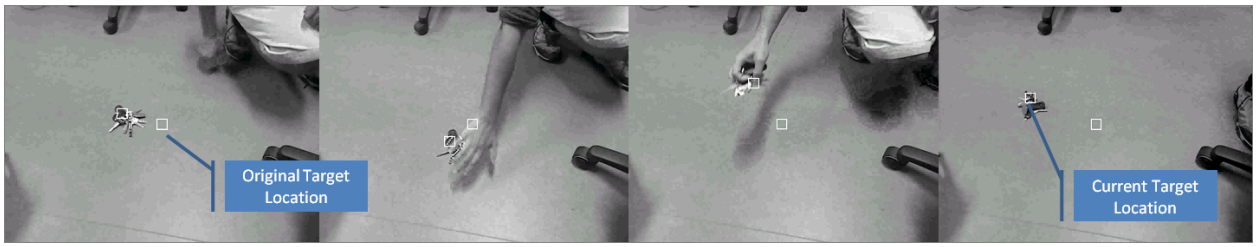

Figure 2.10: Keychain Tracking - real-time and showing robustness through deformations.

Figure 2.11: High Speed Aircraft Tracking - real-time and showing robustness through rotations and scale changes.

With a monochrome image, and the resulting intensity matrix, color information is ignored. When implemented like this, cluttered background can result in erroneous tracking. However, when color is used, the performance of the tracker is improved significantly. Porikli [46] shows the algorithm to work through cluttered environments, grainy video and complete occlusion. However, the GPU implementation of color video streams is left for future work, and may require significant optimization of the kernel algorithm to still run at 100 msec/frame.

In the future, adding advanced search methods, limiting the number of prospect regions searched for the target, and adding a motion model for the target would bring even more significant improvements (all these enhancements have been done before on CPU based implementations).

The target information acquired can be passed to a high level planner which derives its relative position to the aircraft. This information is then transferred to the tracking controllers developed in the subsequent chapters to follow a live target.

Chapter 3

System Identification

3.1 Introduction

The vision tracking system, defined in Chapter 2, gives the real-time location of a high speed moving target on the ground. The aircraft now needs to be able to follow the target through a trajectory tracking controller that can maintain the stability of the aircraft during high speed aggressive turns. Such a controller can be realized using nonlinear control techniques like feedback linearization and backstepping. Both of these methods rely on canceling the nonlinearities in the system using inverse terms calculated using precise knowledge of the system.

This chapter presents a method to obtain such knowledge for a small scale Unmanned Aerial Vehicle (UAV). It relies on tried and tested methods of system identification to extract stability and control derivatives from the flight data. The flight data is obtained through a series of carefully planned flights with custom designed on-board sensors to record airspeed, wind flow angles, attitude and control inputs. The flight is coordinated through a series of ground links making it possible to conduct multiple experiments without landing.

Early work on the development of these methods was conducted at the Dryden Flight Research Center in the 1960's [26]. At the time, the estimated flight parameters like control derivatives and inertia were used to verify wind tunnel predictions and evaluate aircraft performance. They were also used to improve aircraft simulators and basic flight control systems. In the last couple decades these methods have been used to expand flight envelopes, generate high-fidelity aerodynamic derivatives and develop controllers for unstable aircraft.

There are now two main classes of methods for aircraft system identification: online and offline. Online methods process the aircraft measurements as they become available until the parameters converge. Offline methods require a complete time history of measured data and the method is iterated over the data until the parameters converge.

The maximum likelihood method is one of the most popular offline methods. It is based on a probabilistic analysis of the parameters and takes measurement noise into account. In a 1987 lecture, Ilif [25] demonstrated the usefulness of the maximum likelihood estimate, also known as the Output Error Method, and showed 3D graphs of the cost functions. He also showed results from NASA's work on the F-14, HiMAT and the Space Shuttle using this method. In 1988, Jategaonkar and Plaetchke [33] showed the use of an EKF to estimate parameters. They applied this to the maximum likelihood method, the resulting method was called the Filter Error Method. In 1998 Hamel and Jategaonkar [21], show that these class of methods had reached a maturity level. Their applications had widened from general linear use to nonlinear, high fidelity and unstable aircraft. More recently, Naruoka et al. showed that these methods are still superior than the more recent methods based on the Unscented Kalman Filter [45].

With the recent advances in computational power online methods have gained popularity. In 2010, Meng et. al. [44] show an online method that uses a recursive technique with an EKF. Also, in 2010, Chowdhary and Jategaonkar [7] showed that the online recursive methods using an EKF were just as good as the UKF methods, and were computationally less expensive.

The algorithms presented in this chapter are based on the offline maximum likelihood method and require large amounts of processing time, on the scale of half a day to a few days. For data recorded at 100Hz for a period of just 10s, the total number of data points is 130,000 (9 outputs and 4 control inputs). Small sampling times are necessary for developing models that will be useful for nonlinear controllers. The work at Dryden was usually done at 25 to 30 Hz but this has increased significantly over the years with new applications, improved sensors and the rise in computing power.

This chapter proceeds as follows. Section (3.2) describes the general 9 state aircraft model with aerodynamic derivatives and their respective parameters. Section (3.3) goes through the most popular system identification technique, namely, the Output Error Method. Effects of wind turbulence are dealt with in Section (3.4). Proof of concept results using a nonlinear simulation of the BAC-221 jet are shown in Section (3.5). Sections (3.6) and (3.7) show the implementation and flight test results using a Kadet Senior small scale UAV.

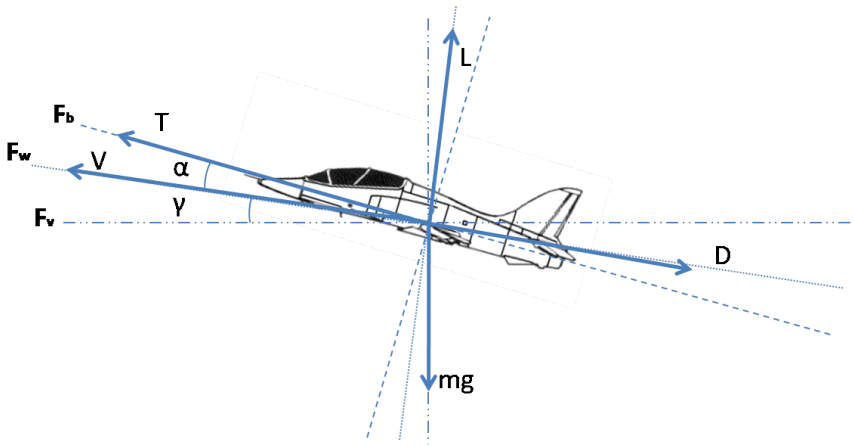

Figure 3.1: Flight path climb angle and the angle of attack.

3.2 General 9 State Aircraft Model

Three distinct coordinate frames are used to define the aircraft model in order to specify forces and moments in a straightforward manner. Aerodynamic forces and moments are defined in the wind frame $\mathbf{F_w}$, i.e., vehicle carried axes aligned with the direction of the oncoming free-stream velocity, or in a vehicle carried vertical frame $\mathbf{F_v}$. The body frame, $\mathbf{F_b}$, is also needed to define angular rates as measured by the Inertial Measurement Unit (IMU).

In the sense of Euler angles (3-2-1 convention), the frames $\mathbf{F_w}$ and $\mathbf{F_b}$ are related by the rotation sequence $(-\beta,\ \alpha,\ 0)$ as seen in Figures (3.1) and (3.2). α and β denote the aerodynamically important angles, angle of attack and sideslip, respectively. The 3-2-1 Euler angle sequence $(\psi,\ \gamma,\ \phi)$ gives the orientation of $\mathbf{F_w}$ relative to $\mathbf{F_v}$. Here, γ denotes the flight path angle of climb and ψ denotes the flight path heading. Angular velocities of the $\mathbf{F_b}$ frame are represented by $(p,\ q,\ r)$ whereas $(p_w,\ q_w,\ r_w)$ give the angular velocity of the $\mathbf{F_w}$ frame.

In the following aircraft model, dynamics are included that link aircraft movements to the individual control surfaces. More importantly, aerodynamic coefficients are included that account for lift, drag and side force created by the airflow.

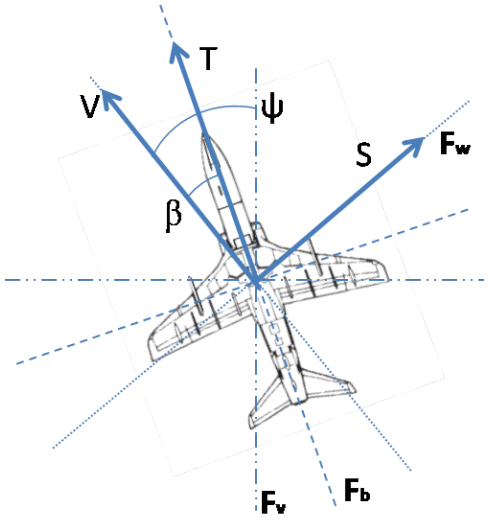

Figure 3.2: Flight path heading angle and the side-slip angle.

Aerodynamic Moments

The moments in the $\mathbf{F_b}$ frame can be expressed as,

$$\begin{aligned} \mathcal{L} &= C_l \frac{1}{2}\rho V^2 sb \\ \mathcal{M} &= C_m \frac{1}{2}\rho V^2 sc \\ \mathcal{N} &= C_n \frac{1}{2}\rho V^2 sb \end{aligned} \tag{3.1}$$

where $\mathcal{L}, \mathcal{M}, \mathcal{N}$ represent rolling, pitching and yawing moments respectively, ρ is the density of air, V is the freestream velocity, s, the wing area, b the wing span and c the chord length or characteristic length of the aircraft. C_l, C_m and C_n represent the nondimensional coefficients which can be represented as,

$$\begin{aligned} C_l &= f_{C_l}(\alpha, \beta, \delta R, \delta A) \\ C_m &= f_{C_m}(\alpha, \delta E) \\ C_n &= f_{C_n}(\alpha, \beta, \delta A, \delta R) \end{aligned} \tag{3.2}$$

where δE, δA and δR are the aircraft elevator, aileron and rudder deflections respectively. Note that the pitching moment is mainly affected by the elevators, where as the rolling and yawing moments are affected mainly by the ailerons and the rudder.

Aerodynamic Forces

Forces in the $\mathbf{F_w}$ frame are

$$\begin{aligned} L &= C_L \frac{1}{2}\rho V^2 s + T\sin\alpha \\ D &= C_D \frac{1}{2}\rho V^2 s - T\cos\alpha\cos\beta \\ S &= -C_S \frac{1}{2}\rho V^2 s + T\cos\alpha\sin\beta \end{aligned} \tag{3.3}$$

where T is the thrust, L is the lift force, D is the drag force, S is the side force (as depicted in Figures (3.1) and (3.2)) and C_L, C_D and C_S are the corresponding nondimensional aerodynamic coefficients. These depend on α and β as shown below,

$$\begin{aligned} C_L &= f_{C_L}(\alpha) \\ C_D &= f_{C_D}(\alpha) \\ C_S &= f_{C_S}(\alpha,\beta) \end{aligned} \tag{3.4}$$

where the lift and drag are functions of the angle of attack as expected, and the side force is a function of both the angle of attack and sideslip angle. Note, the direct effects of control surface deflection on the forces are neglected. The control surfaces mostly affect the moments which in turn affect the forces. It can be shown that if the effects of control surface deflection are directly included in the equations for forces, the nonlinear controllers will request unnecessarily large amounts of control inputs that could lead to an unstable system [39].

The thrust, T, can be approximated by

$$T = T_{max}\delta T \tag{3.5}$$

where δT is the throttle setting and T_{max} is the maximum engine thrust. A more complicated engine model can be used here without loss of generality (see, for example, Ducard et al. [11]).

Equations of Motion

Ignoring the effects of Earth's curvature (flat-earth approximation [13]) and using combined wind and body frames for forces, angles and angular rates [39], the equations of motion

for an aircraft can now be defined. These equations were developed by Etkin [13] and are reproduced here for reference. The model consists of nine states and four inputs.

Kinematically important terms, airspeed, V, flight path climb angle, γ, and flight path heading, ψ, can be defined in terms of forces and angular velocities

$$\begin{aligned} \dot{V} &= -\frac{D}{m} - g\sin\gamma \\ \dot{\gamma} &= q_w\cos\phi - r_w\sin\phi \\ \dot{\psi} &= (q_w\sin\phi + r_w\cos\phi)\sec\gamma \end{aligned} \tag{3.6}$$

where the angles and angular velocities are in the $\mathbf{F_w}$ frame.

The angle of attack, α, the side-slip angle, β and wind-axes roll, ϕ are functions of angular velocities (in both $\mathbf{F_b}$ and $\mathbf{F_w}$ frames) and take the form

$$\begin{aligned} \dot{\alpha} &= q - q_w\sec\beta - p\cos\alpha\tan\beta - r\sin\alpha\tan\beta \\ \dot{\beta} &= r_w + p\sin\alpha - r\cos\alpha \\ \dot{\phi} &= p_w + q_w\sin\phi\tan\gamma + r_w\cos\phi\tan\gamma. \end{aligned} \tag{3.7}$$

The moment equations are best described in the $\mathbf{F_b}$ frame to simplify the relationship to control inputs

$$\begin{aligned} \dot{p} &= \frac{1}{I_x}(\mathcal{L} + I_{zx}(\dot{r} + pq) + (I_y - I_z)qr) \\ \dot{q} &= \frac{1}{I_y}(\mathcal{M} + I_{zx}(r^2 - p^2) + (I_z - I_x)rp) \\ \dot{r} &= \frac{1}{I_z}(\mathcal{N} + I_{zx}(\dot{p} - qr) + (I_x - I_y)pq) \end{aligned} \tag{3.8}$$

where inertia, I, is given by

$$I = \begin{bmatrix} I_{xx} & 0 & -I_{zx} \\ 0 & I_{yy} & 0 \\ -I_{zx} & 0 & I_{zz} \end{bmatrix} \tag{3.9}$$

In Equations (3.6) to (3.8), angular velocities in the $\mathbf{F_w}$ frame (p_w, q_w, r_w) can be expressed as functions of the system states $(V, \gamma, \psi, \alpha, \beta, \phi, p, q, r)$ as

$$\begin{aligned} p_w &= p\cos\alpha\cos\beta + (q - \dot{\alpha})\sin\beta + r\sin\alpha\cos\beta \\ q_w &= \frac{1}{mV}(L - mg\cos\gamma\cos\phi) \\ r_w &= -\frac{1}{mV}(S - mg\cos\gamma\sin\phi) \end{aligned} \tag{3.10}$$

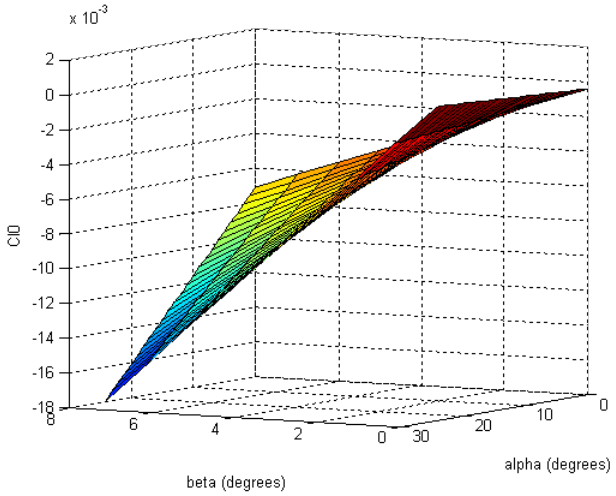

Figure 3.3: $C_{l0} = k_1\alpha + k_2\beta + k_3\alpha\beta + k_4$

In summary, Equations (3.6) to (3.8) define the 9 states of the aircraft, driven by the control inputs, δT, δE, δA and δR through the moment and force definitions of Equations (3.1) to (3.5).

The non-dimensional coefficients defined above can be expanded into a series of unknown parameters. Starting with the moment equations, relationships of the following form can be derived [39],

$$\begin{aligned} C_l &= C_{l0} + C_{l\delta R}\delta R + C_{l\delta A}\delta A \\ C_m &= C_{m0} + C_{m\delta E}\delta E \\ C_n &= C_{n0} + C_{n\delta A}\delta A + C_{n\delta R}\delta R \end{aligned} \tag{3.11}$$

where each C_{ij}, with $i = l, m, n$ and $j = \delta E, \delta A, \delta R$, represents the derivative of C_i w.r.t j (also known as control derivatives). Each of these can be approximated as general functions of α and β as follows,

$$C_{ij} = k_1\alpha + k_2\beta + k_3\alpha\beta + k_4 \tag{3.12}$$

where $k_1 \ldots k_4 \in \mathbb{R}$. As an example, the first term of C_l, C_{l0}, for a BAC-221 can be seen in Figure (3.3).

Following a similar approach, appropriate relations for the force coefficients can be formed. The resulting expressions are presented in the 1988 paper by Lane and Stengel [39] and take the following form, with $k = L, D, S$,

$$\begin{aligned} C_k =& C_{k1} + C_{k2}\alpha + C_{k3}\beta + C_{k4}\beta^3 + C_{k4}\alpha^3 + \\ & C_{k5}\alpha\beta^3 + C_{k6}\alpha^3\beta + C_{k7}\alpha\beta \end{aligned} \tag{3.13}$$

where $C_{k1} \ldots C_{k7}$ form the main force parameters. The parameters for moments are defined above and with mass, m, and Inertia components in matrix, I, complete the parameters required to characterize the aircraft for nonlinear control applications. For linear control systems designed around particular operating points, one or more of these general parameters can be set to zero.

3.3 System Identification Using the Output Error Method

The Output Error Method is one of the most widely used system identification methods. The Dryden Flight Research Center has been using this method on experimental and new aircraft since the 1960's [26]. Although initially only used to estimate small linear models, the method can now be used for nonlinear models owing much of its success to increased computing power.

3.3.1 System Setup

Figure (3.4) shows the main idea behind the Output Error Method. A general nonlinear system with a time invariant vector of parameters, Θ, can be defined as

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t), \Theta) \\ y(t) &= g(x(t), u(t), \Theta) \\ z(t_k) &= y(t_k) + v(t_k) \end{aligned} \tag{3.14}$$

where $x \in \mathbb{R}^n$ is the state ($n = 9$ in this case: V, γ, ψ, α, β, ϕ, p, q, r), $u \in \mathbb{R}^m$ is the input ($m = 4$ in this case: δT, δE, δA, δR), $y \in \mathbb{R}^{n_y}$ is the output ($n_y = 9$ in this case: same as the number of states) and $z \in \mathbb{R}^{n_y}$ is the measured response (also, $n_y = 9$ in this case: measuring all the states with independent sensors). v is then the measurement error which is assumed to be distributed with zero mean and covariance matrix R given by

$$R = E([v(t_k)].[v(t_k)]^T). \tag{3.15}$$

where $E()$ is the expected value or mean. Furthermore, u is assumed to be generated by an exogenous system that is not affected by the system output and sufficiently excites a broad spectrum of frequencies on the system.

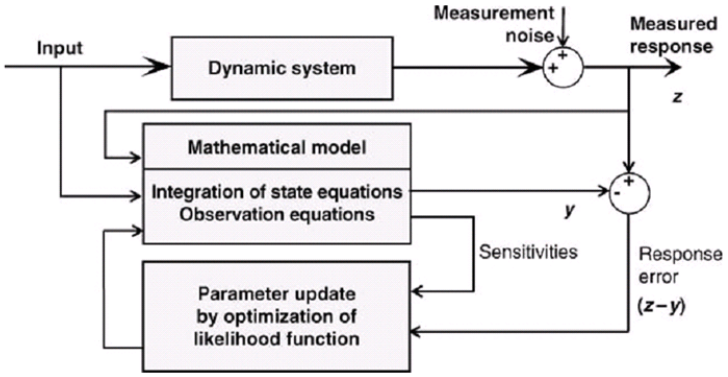

Figure 3.4: An adaptive system identification method that changes the model based on the output error [32].

3.3.2 Defining the Cost Function

The likelihood function of z can be defined as,

$$P(z|\Theta, R) = \prod_{k=1}^{N} P(z_k|\Theta, R) \tag{3.16}$$

where z_k is the measurement at t_k and $P(z|\Theta, R)$ is the probability of z given Θ and R. Maximizing this function with respect to Θ would result in parameter estimates within an acceptable range of their true values. It is important to note that this likelihood function represents the probability density of the observed variables and not of the parameters. The maximum likelihood estimate of the parameters, Θ_{ML}, can therefore be defined as

$$\Theta_{ML} = \arg\max_{\Theta} P(z|\Theta, R) \tag{3.17}$$

or equivalently, in its standard form as

$$\Theta_{ML} = \arg\min_{\Theta} (\ln P(z|\Theta, R)). \tag{3.18}$$

Solving this equation yields parameter estimates that are asymptotically unbiased and normally distributed around their true value (see Wald, A. [59]). This can be expressed by

$$\lim_{N\to\infty} E(\Theta_{ML}) = \Theta \tag{3.19}$$

where N is the total number of measurements $(z_1, z_2, ..., z_N)$.

Given the above system, it can be shown that

$$\ln P(z|\Theta, R) = \frac{1}{2}\sum_{k=1}^{N}[v(t_k)]^T R^{-1}[v(t_k)] + \frac{N}{2}\ln[\det(R)] + \frac{Nn_y}{2}\ln(2\pi). \tag{3.20}$$

A cost function, J, can now be defined as

$$J(\Theta, R) = \ln P(z|\Theta, R) \tag{3.21}$$

whose minimization can be performed using any of the common optimization methods.

3.3.3 Optimization Methods

J is minimized with respect to Θ when $\frac{\partial J}{\partial \Theta}$ is equal to zero. Using a Taylor series approximation, this can be represented as

$$\left(\frac{\partial J}{\partial \Theta}\right)_{i+1} \approx \left(\frac{\partial J}{\partial \Theta}\right)_i + \left(\frac{\partial^2 J}{\partial \Theta^2}\right)_i \Delta\Theta \tag{3.22}$$

$$\Delta\Theta \approx \left[\left(\frac{\partial^2 J}{\partial \Theta^2}\right)_i\right]^{-1}\left(\frac{\partial J}{\partial \Theta}\right)_i \tag{3.23}$$

Applying this to Equation (3.20) and deriving further results in (see Jategaonkar, R. [32]),

$$\frac{\partial J}{\partial \Theta} = -\sum_{k=1}^{N}\left[\frac{\partial y(t_k)}{\partial \Theta}\right]^T R^{-1} v(t_k) \tag{3.24}$$

$$\frac{\partial^2 J}{\partial \Theta^2} = \sum_{k=1}^{N}\left[\frac{\partial y(t_k)}{\partial \Theta}\right]^T R^{-1}\frac{\partial y(t_k)}{\partial \Theta} + \sum_{k=1}^{N}\left[\frac{\partial^2 y(t_k)}{\partial \Theta^2}\right]^T R^{-1} v(t_k). \tag{3.25}$$

Referring back to the initial assumption that the measurement noise is Gaussian with zero mean, the second term in Equation (3.25) will disappear as N becomes large. The second derivative can now be approximated as

$$\frac{\partial^2 J}{\partial \Theta^2} \approx \sum_{k=1}^{N}\left[\frac{\partial y(t_k)}{\partial \Theta}\right]^T R^{-1}\frac{\partial y(t_k)}{\partial \Theta} \tag{3.26}$$

and forms the basis for the modified Newton-Raphson method, also known as the Gauss-Newton method. Once the $\Delta\Theta$ is found it can be used to improve the parameter vector for the next iteration. The R matrix is also recalculated at every iteration.

To gain further insight into the cost function, notice that the noise covariance in Equation (3.15) can be rewritten as

$$R = \frac{1}{N}\sum_{k=1}^{N}[v(t_k)][v(t_k)]^T \tag{3.27}$$

and substituted into Equation (3.20) to give

$$J(\Theta) = \frac{1}{2}n_y N + \frac{N}{2}ln[det(R)] + \frac{Nn_y}{2}ln(2\pi) \tag{3.28}$$

where the first and third term on the right are constants and the cost function becomes a function of the determinant of the covariance. This method is also sometimes known as the Gauss-Newton method with determinant minimization and can be set up using the built-in functions in Matlab.

Other optimization methods can also be used within the Output Error Method. The choice would depend on the size of the model being estimated and on how much apriori information is available. For example, the Gauss-Newton method described above performs poorly and may never converge if a reasonable starting point is not given. The method can be improved by adding Lagrange multipliers and slack variables to limit the value of the parameters to a predefined range. This is known as the Constrained Gauss Newton Method.

3.3.4 Algorithm Steps

Once the model and cost function are set up as above and an optimization method is chosen, the following steps can be implemented to converge to suitable parameter values:

1. Input initial guesses on Θ and R.
2. Calculate predicted output, y, and find measurement errors $(z - y)$.
3. Calculate R.
4. Minimize the cost function with respect to Θ.
5. Go back to 2 and iterate until Θ converges.

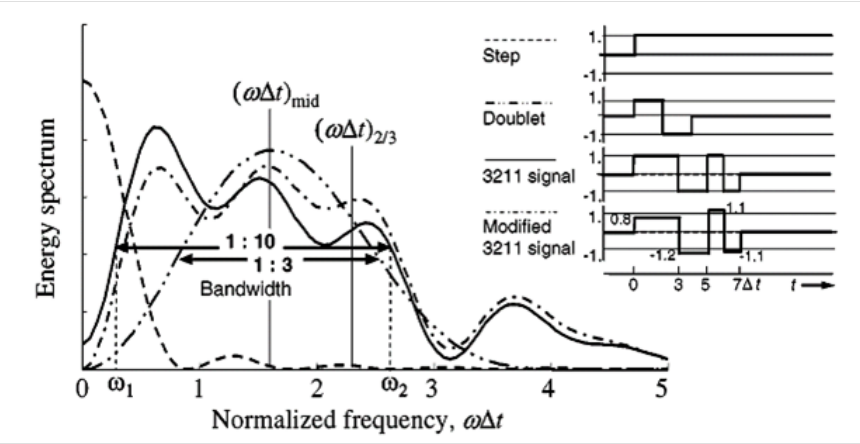

Figure 3.5: Frequency Spectrum of Various Maneuvers [32].

3.3.5 Flight Maneuvers

The quality of parameter estimates and the limits of their application to flight controls depend directly on the maneuvers performed during the test flight. These methods are the same as the ones used for estimating the flight envelope and performance parameters of a new aircraft. Usually the test pilot is given a preflight briefing on the type of control inputs required and their purpose. For stable aircraft, it is best if no closed loop control aides are used during the test flight. This ensures that all the natural modes are captured within the system model.

The estimation and optimization methods described above assume that the control inputs excite a wide frequency range of the system. This would include maneuvers that excite the basic aircraft modes including phugoid, short period and Dutch roll. These modes can be achieved by applying pulse, step and multistep inputs on the control surfaces. Figure (3.5) shows the bandwidth of various control inputs where it is evident that multistep inputs cover a much wider range of frequency than simple step inputs as shown by Jategaonkar [32]. Optimal design of these control inputs is also possible where the inputs cover the desired range. These can be programmed onto a flight computer for open loop implementation.

A typical set of maneuvers used for system identification can be seen in Figure (3.6) [32]. Note, the 3-2-1-1 maneuvers are a modified form of the doublet and excite the largest frequency range.

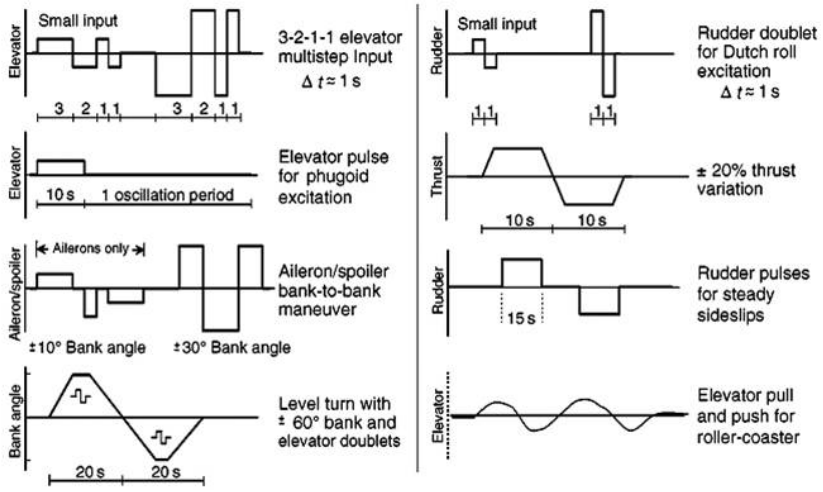

Figure 3.6: Maneuvers to excite different modes on an aircraft [32].

3.4 Atmospheric Turbulence

Before looking at the parameter estimation results for the Kadet Senior UAV, it is worthwhile to note the limitations of the above techniques. The Output Error Method, which takes into account the measurement noise, doesn't account for unmeasurable process noise, or in the case of aircraft, atmospheric turbulence. Three main solutions exist for improving the model fit:

Measure Wind Components

The obvious first solution would be to derive the unmeasurable disturbances using measured airspeed, wind flow angles (α, β), attitude and inertial acceleration. This assumes that the effect of turbulence on these measured values is known which is usually not available for small scale UAVs.

Dryden Wind Gust Model

The Dryden Wind Gust model can be used in the method explained above where the parameters of the gust are included as part of the overall parameter estimation problem.

Shown in Figure (3.7) are results from the Dryden Flight Research Center's work using similar methods performed on flight data obtained during atmospheric turbulence [26].

Filter Error Method

The Filter Error Method falls under the general class of the Output Error Method described above. The general scheme is shown in Figure (3.8). The main difference is the inclusion of unmeasurable process noise. This method was initially developed for linear models, using Kalman filters for improving the predicted state at every iteration. In the 1980's Jategaonkar and Plaetschke adapted this method using a mixed version of the Extended Kalman Filter [34]. Here, the prediction step relies on numerically integrating the nonlinear state equation using the previous state as its initial condition. The correction step, however, still uses linearization about the current state to calculate the Kalman gain. This can be written as,

State and Measurement Prediction:

$$\begin{aligned} \tilde{x}(t_k) &= \hat{x}(t_{k-1}) + \int_{t_k}^{t_{k+1}} f(x(t), u(t_k), \Theta) \\ \tilde{y}(t_k) &= g(\tilde{x}(t_k), u(t_k), \Theta) \end{aligned} \tag{3.29}$$

Correction:

$$\hat{x}(t_k) = \tilde{x}(t_k) + K(z(t_k) - \tilde{y}(t_k)) \tag{3.30}$$

where $\tilde{x}(t_k)$ is the predicted state, $\tilde{y}(t_k)$ is the predicted measurement, $\hat{x}(t_k)$ is the corrected belief over the state at time, t_k, and K, the Kalman gain, is defined as:

$$K = PC^T R^{-1} \tag{3.31}$$

where P is the covariance matrix of the state prediction error calculated from a steady-state form of the Ricatti equation and C is the observation matrix calculated using a first-order approximation:

$$C = \left[\frac{\partial g(x(t), u(t), \Theta)}{\partial x} \right]_{x(t)=\tilde{x}(t_k)} \tag{3.32}$$

Similar to the Output Error Method, the algorithm steps are:

1. Input initial guesses on Θ and R.

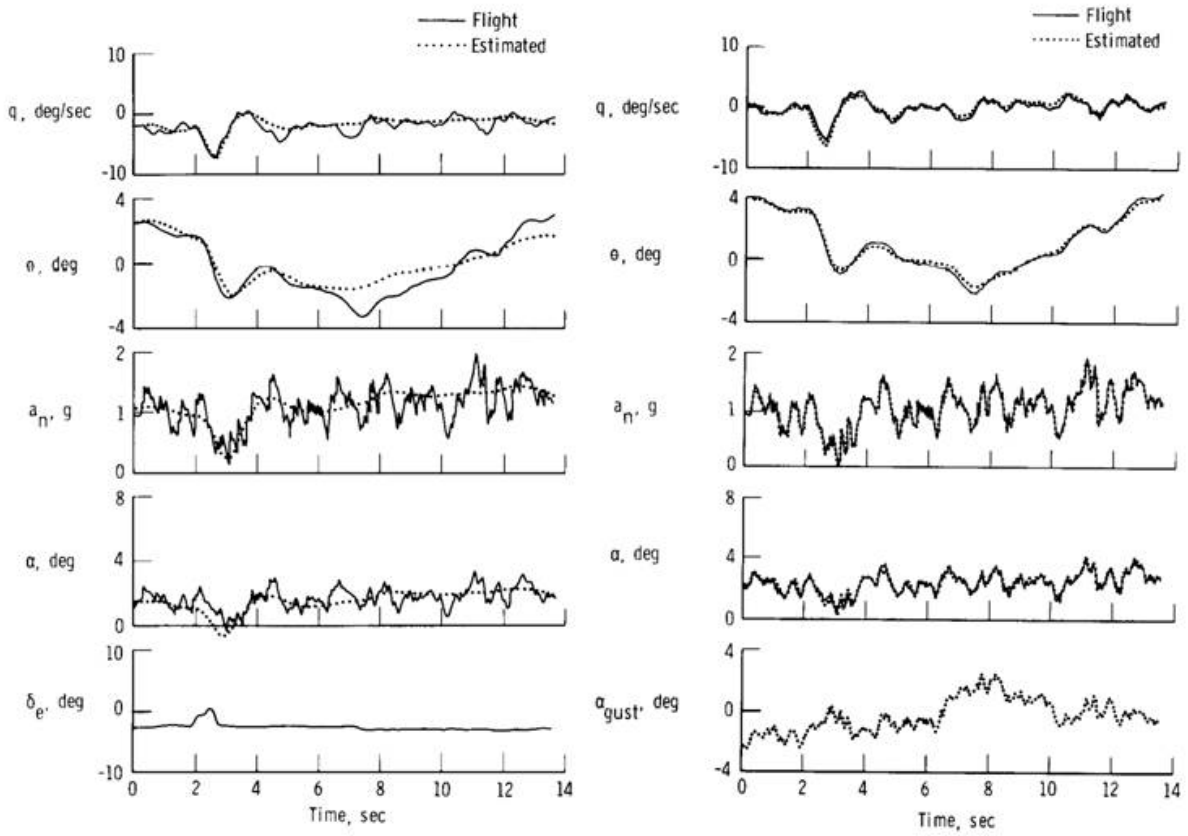

Figure 3.7: Direct application of the output error method without accounting for atmospheric turbulence on the left and accounting for turbulence using the Dryden Wind Gust model on the right [26].

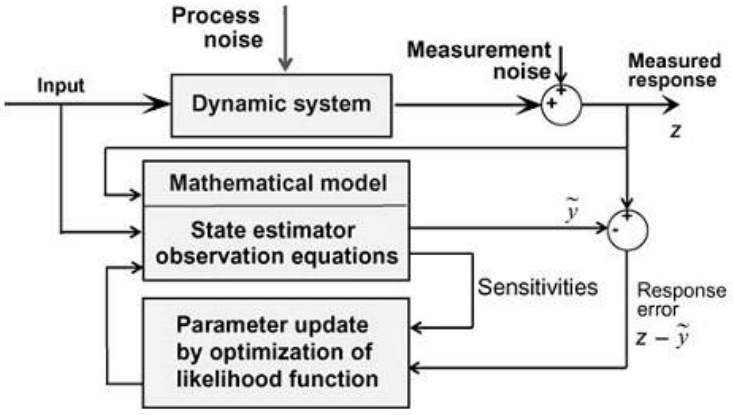

Figure 3.8: An adaptive system identification method that changes the model based on the output error and accounts for process noise [32].

2. Estimate states using the modified EKF as above and calculate errors $(z - \tilde{y})$.
3. Calculate R.
4. Minimize the cost function with respect to Θ.
5. Go back to 2 and iterate until Θ converges.

This method leads to an optimization problem that is nearly linear, has fewer local minima and a faster convergence rate [33].

3.5 Results from BAC-221 Simulation

The BAC-221 was a British supersonic test fighter, developed in the 1970s as a precursor to the Concorde. It had a wingspan of 26 ft, a Rolls-Royce Avon 200 jet engine that produced 8000 lbf of thrust and a max speed of Mach 1.7. At the time, the Royal Aircraft Establishment at Farnborough produced a set of aerodynamic data from various wind tunnel tests ranging from Mach 0.2 to 0.955 [20]. This detailed data was used to define a realistic full aerodynamic model for simulation purposes.

This data (valid for $0 < \alpha < 24^\circ$ and $0 < \beta < 8^\circ$) can be used to generate nonlinear functions for standard nondimensional coefficients that relate the moments and forces on

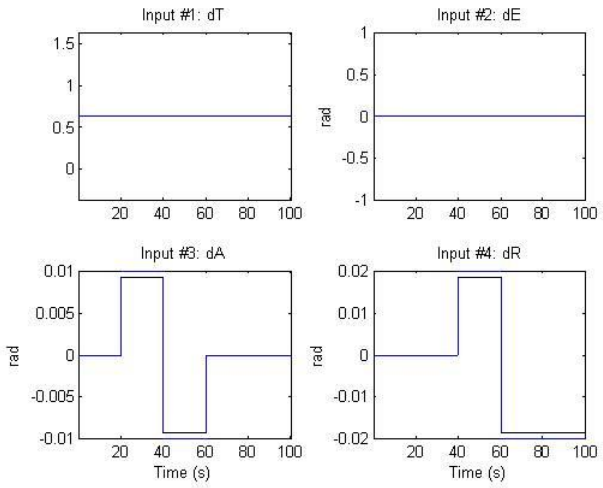

Figure 3.9: Inputs to the BAC-221 nonlinear simulation.

an aircraft to the states and control inputs. These functions can then be used to form a full aerodynamic model of the aircraft as discussed in section (3.2).

As a first test of the system identification methods described above, a full nonlinear model of this aircraft was developed and simulated in Matlab with the required maneuvers. The "measurements" were sampled at a fixed time interval without any atmospheric turbulence. The model included 32 unknown parameters (from the general forms in Equations (3.12) and (3.13)) with initial guesses of the same order of magnitude as the true parameter values. The results of the Output Error Method using Gauss-Newton optimization are shown in Figures (3.9) and (3.10). The predicted and measured states are reasonably close. Notice the poor fit on V, which is mainly a result of no input on dT (insufficient excitation). This demonstrates the need for designing appropriate control inputs.

3.6 Data Collection

The methods described above rely on in-flight data collection of the system outputs and result in accurate estimation of parameters if the following guidelines by Jategaonkar [32] are followed:

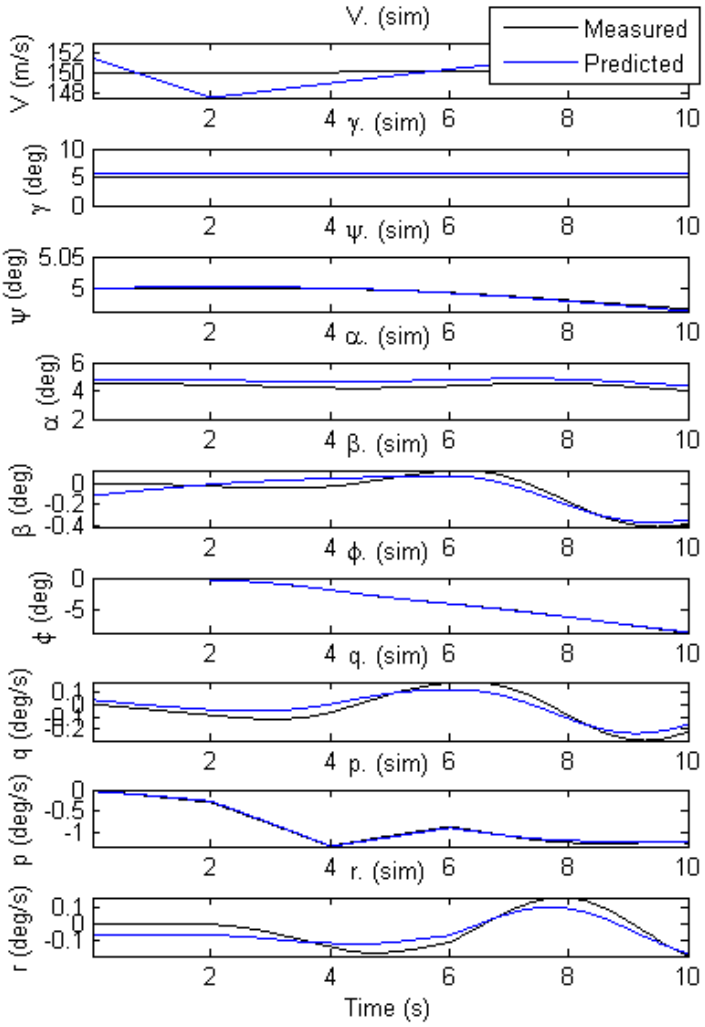

Figure 3.10: Output Error Method fit on the BAC-221.

1. The minimum sampling rate for data collection should be twice the frequency of interest (also known as the Nyquist frequency).

2. Delays introduced in the data due to filtering (post or in-flight) should be kept as consistent as possible across all measurements.

3. All outputs should be measured at the same sampling rate if possible. This is specially true for attitude and wind data.

4. Ideally, all outputs should be measured and recorded at the same time instant.

5. The signal to noise ratio to should be limited to below 10:1 either in flight or during post process filtering.

Required measurements for a reasonable system identification include all the 9 states in the general nonlinear aircraft model described in Section (3.2) and can be termed as system outputs. In addition, the control inputs to the throttle and the control surface deflections of the elevator, aileron and rudder also need to be recorded on board the aircraft. Implementation details of these on a remotely piloted Kadet Senior UAV are provided below.

3.6.1 Attitude and Wind Sensors

Airspeed can be measured using pitot tubes and MEMS based differential pressure sensors. They are mounted where the flow is considered "free-stream" and is affected minimally by the propeller wash and wing-tip vortices.

α and β angles can be determined using either a multiport pitot tube or two rotary vanes. Multiport pitot tubes require careful calibration and are less accurate than two vanes connected to precision rotary encoders [45]. However, the vanes also induce more drag than a multiport pitot tube. In addition, if the UAV has a pusher configuration the multiport pitot tubes are a better choice because they can be mounted along the centerline and are not affected by roll or yaw. Figure (3.11) shows the custom designed angle of attack sensor and pitot tubes mounted on the Kadet Senior wing. The vanes use magnetic absolute encoders that relay readings over an analog signal. A similar assembly is used on the other half of the wing to measure the angle of sideslip and to balance the UAV.

Euler angles and angular rates in $\mathbf{F_b}$ are recorded using an IMU consisting of 3 gyros, 3 accelerometers and 3 magnetometers. These are converted to $\mathbf{F_w}$ using α and β measured

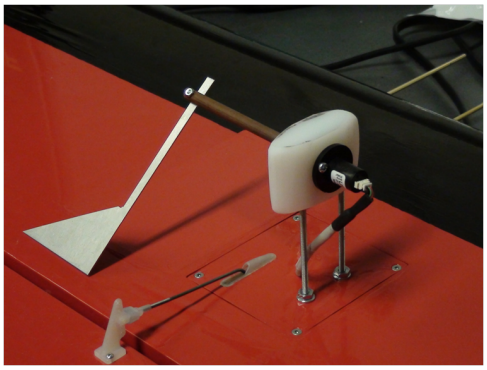

Figure 3.11: Custom designed angle of attack vane mounted under the wing with pitot tubes visible in the background. A similar angle of sideslip vane is also used.

above. This is done during post processing to minimize the effect of excessive calculations on sampling time.

Lastly, the control inputs from the ground are recorded on board by tapping into the RC receiver signals using an ATMEGA 2560 microcontroller, shown in Figure (3.12). Control inputs can also be measured more precisely using rotary encoders. However, the reference signals to the servo motors, which run an internal position control loop, are sufficient indicators of surface deflections and throttle inputs.

The airspeed, angles, α and β, and control inputs, dT, dE, dA and dR are all recorded using the ADC and PWM inputs on the microcontroller. The PWM inputs are programmed to be interrupt driven on rising and falling edges and use a 250 kHz clock to time the interval. The ADC inputs are also interrupt driven. All these inputs are packaged in a string and transmitted over USB to be recorded by an onboard computer running a QNX program utilizing the Termios API.

3.6.2 Ground Control and Monitoring

A good ground control station can serve multiple purposes and is vital for a successful test flight. The custom designed Kadet Senior ground control station has the following features:

- Matlab based live plots showing current control inputs, airspeed, angle of attack and angle of sideslip.

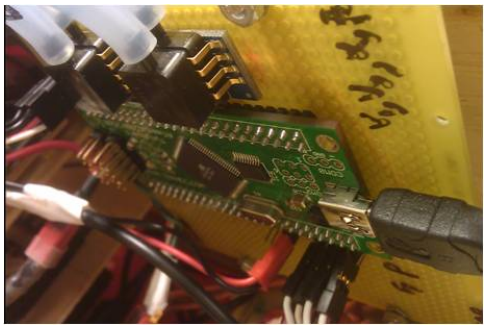

Figure 3.12: Atmega 2560 with 2 differential pressure sensors and inputs for alpha, beta vanes as well as 4 PWM inputs from the RC receiver.

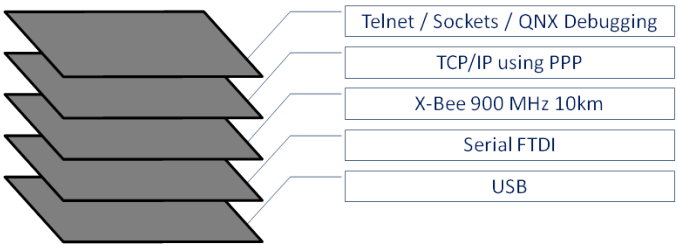

Figure 3.13: Low cost, custom designed communication stack for the UAV allowing multiple channels through a single serial line.

- Telnet terminal with access to on-board QNX and the ability to start and stop different recording programs mid-flight.
- In-flight remote debugging of new software.
- Mid-flight review of recorded data, by direct access to the text file.
- Other TCP/IP connections using the on-board BSD Socket library.

These features are all accessed through a single low cost (<$50) serial modem with a 10km line of sight range. Since serial ports only allow access to one program at a time a TCP/IP layer is added to allow multiple connections. Matlab communicates with QNX over sockets running on this layer. Similarly, the remote debugging and Telnet sessions also require the TCP/IP layer. Figure (3.13) shows the communication stack.

3.6.3 Real-time Thread Execution in QNX

The rate at which the data is recorded plays a pivotal role in identifying an accurate model. Data that is collected at a lower and inconsistent rate is bound to only provide a model that is useful for flight controls requiring low update rates. Since the intention here is to eventually implement an aggressive nonlinear feedback linearizing controller, the sampling rate needs to be reasonably fast and consistent. A NASA report on the practical aspects of these system identification methods is detailed in [26]. The report shows how the sampling rate affects the quality of estimated parameters. Notably, rapid excursions, caused by rapid control inputs require a faster and consistent sampling rate, than say, when estimating slower motions like phugoid modes. In addition, time and phase shifts are also very important. When measurements are sampled sequentially, the time shift between a measured sample at the beginning of the interval and a measured sample at the end is crucial. With a low sampling rate and phase shifts, the initiation of a control input might be missed, causing the vehicle to appear to respond before the control input. The Output and Filter Error Methods discussed above, assume that all measurements are sampled simultaneously and at adequate sampling rates.

These requirements can be implemented in QNX using either timer interrupt handlers or pulses, with the high priority threads sitting in a pulse receive state. The pulses themselves are generated by the kernel [1] using the real time clock and interrupts. The frequency of these interrupts dictate the resolution of the timing achieved. The goal is to achieve the best consistent update rate possible while keeping the system overheads low. For example, a 1ms resolution clock will give a sampling rate of anywhere from 71.43Hz to 76.29Hz if the desired frequency is 75Hz. This varying frequency will produce errors in parameter estimates as discussed above. Increasing the clock resolution also increases the system load and would delay thread execution. A clock resolution of 100us gives a frequency of 74.63Hz to 75.19Hz which is reasonably acceptable. Moving further down to 10us improves this even more but delays thread execution as most of the processing time is used to handle interrupts. A pie chart showing this comparison is shown in Figure (3.14).

A multi-threaded program that extracts data from multiple sensors and records them onto a harddrive is run on-board the aircraft. It features constant time intervals with hard real-time constraints and synchronized data recording. Figure (3.15) shows the timing trace of the program recorded in real-time over a period of about 180ms. The socket server serving Matlab on the ground is also one of the threads (No. 6) but is run at a much

[1] The term, kernel, in this chapter refers to an Operating System kernel, which is different from the OpenCL kernel in the previous chapter.

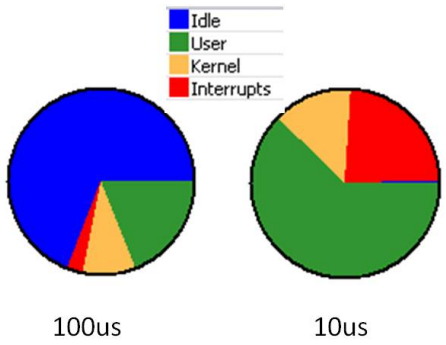

Figure 3.14: CPU usage comparison at different clock resolutions. Notice the virtual absence of Idle at 10us.

lower priority. This same architecture can later be used directly for nonlinear controls implementation.

3.7 Results from Kadet Senior

A portion of the data collected during test flights and a model fit obtained using the Output Error Method is shown in Figure (3.16). The fit is similar to the one seen in Figure (3.7). It was obtained using multiple runs of the algorithm where the values from the first run were used as the initial parameters for the next run but some were "kicked" to let the solution escape local nonlinear minima.

3.7.1 Simplifying the Model and Reducing Identification Errors

Two ways to improve this fit would be to include the turbulence methods discussed earlier and to improve the initial guesses on the parameters. The performance of the OEM is heavily dependent on the initial parameter estimates due to the highly nonlinear nature of the model, which results in many local minima. A good starting point for these would be wind tunnel tests. Lacking those, one can look up the most common parameters for other UAVs of similar scale and extrapolate to all required parameters. The Dryden Wind Gust model can also improve the fit as discussed earlier. The OEM only accounts for measurement noise, therefore disturbances from wind still need to be tackled.

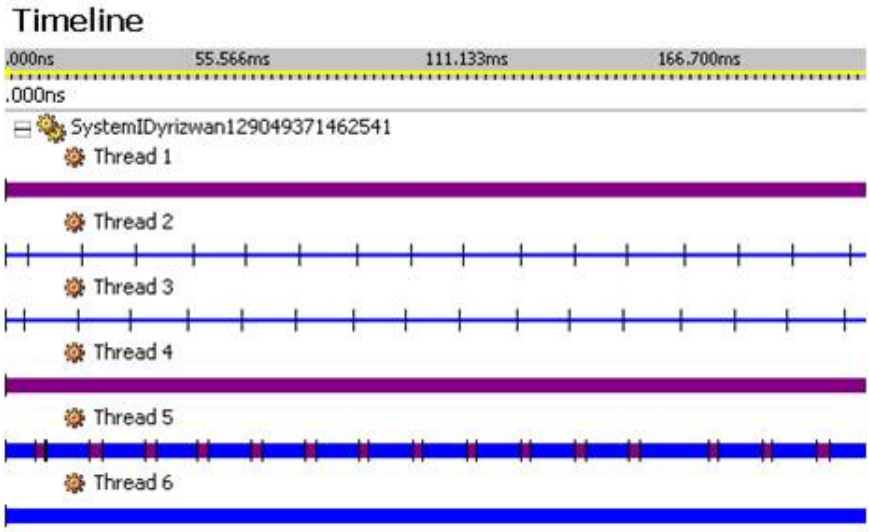

Figure 3.15: Timing diagram showing uniform and synchronized thread execution at 75Hz.

Another way to reduce wind disturbance errors is the Filter Error Method describe earlier. This is left for future work.

Implementing nonlinear controls on a model of this size can be challenging. While exact requirements for controls are held off until the next chapter, it is sufficient to say that a smaller analytical model with a better fit would fare better when designing control laws. The aerodynamic model resulting from the parameters in Equations (3.12) and (3.13) would be usable for nonlinear controls when some of the general parameters are zero, and hence simplifying the resulting control laws. The identification process was started with most of the parameters fixed to zero and was incrementally increased to all available parameters. The results, as shown in Figure (3.16), can be used for high fidelity simulations of the Kadet Senior. However, for control law derivations, a simpler analytical model, that fits equally as good, if not better, is required.

Consolidated Model: Replacing Polynomials with Sin and Cos

While using the same 9 state model from Equations (3.6) to (3.10), as described earlier, one can replace the underlying aerodynamic forces and moments polynomials with trigonometric functions (akin to a Taylor series vs a Fourier series). The idea is to reduce the number of parameters so that the resulting nonlinear controller equations are simpler and easy to

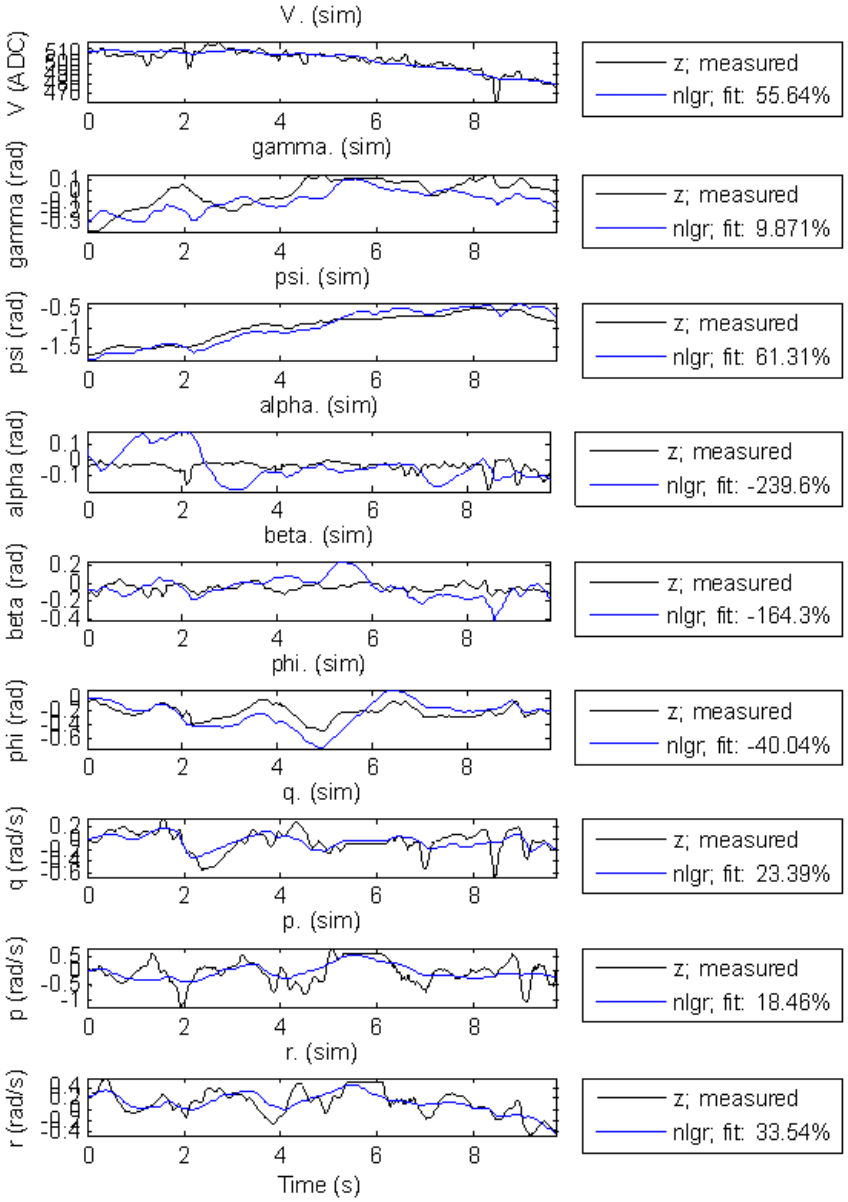

Figure 3.16: Results from the Kadet Senior UAV, with a large number of underlying aerodynamic parameters from (Equations (3.12) and (3.13)).

compute on the flight computer. Isidori [29] proposes the following as an alternative to Equations (3.1) to (3.3).

$$\begin{aligned}\begin{bmatrix} \mathcal{L} \\ \mathcal{M} \\ \mathcal{N} \end{bmatrix} &= V \begin{bmatrix} a_{12}r + a_{13}p \\ a_{23}q \\ a_{32}r + a_{33}p \end{bmatrix} + V^2 \begin{bmatrix} a_{11}\sin\beta \\ a_{21} + a_{22}\sin\alpha \\ a_{31}\sin\beta \end{bmatrix} \\ &\quad + V^2 \begin{bmatrix} b_{11}\cos\beta & 0 & b_{13}\cos\beta \\ 0 & b_{22}\cos\alpha & 0 \\ 0 & 0 & b_{33}\cos\beta \end{bmatrix} \begin{bmatrix} \delta A \\ \delta E \\ \delta R \end{bmatrix}\end{aligned} \tag{3.33}$$

$$\begin{bmatrix} L \\ D \\ S \end{bmatrix} = V^2 \begin{bmatrix} c_{21} + c_{22}\sin 2\alpha \\ c_{11} + c_{12}\cos\alpha \\ c_{31}\sin 2\beta \end{bmatrix} + T_{max} \begin{bmatrix} \sin\alpha \\ -\cos\alpha\cos\beta \\ \cos\alpha\cos\beta \end{bmatrix} \delta T \tag{3.34}$$

Just like in Equation (3.2), the rolling moments here are mainly affected by δR and δA, the pitching moments are affected mainly by δE and the yawing moments are mainly affected by δR. However, the inclusion of δA in the calculation of yawing moments is missing. This correlation can be observed in Figure (3.17). Notice that the effect on yaw rate, r, is only noticeable during large changes in δA movements. An extra parameter (b_{31}) is added to improve the model.

Furthermore, as discussed in Section (3.2), the inclusion of control surface deflections in the force equations is still avoided here. However, due to a strong correlation (seen in Figure (3.18)), and the need to simplify the model, a term with q is also added to the main Lift equation. By including q, the affect of δE is still captured, but is delayed to the next derivative, the advantage of doing this is evident when deriving feedback linearizing control laws in Chapter (4). However, when simulating the nonlinear controller on the Kadet Senior, this extra parameter is used in the plant but is explicitly excluded from the control law.

The resulting forces and moments then look like,

$$\begin{aligned}\begin{bmatrix} \mathcal{L} \\ \mathcal{M} \\ \mathcal{N} \end{bmatrix} &= V \begin{bmatrix} a_{12}r + a_{13}p \\ a_{23}q \\ a_{32}r + a_{33}p \end{bmatrix} + V^2 \begin{bmatrix} a_{11}\sin\beta \\ a_{21} + a_{22}\sin\alpha \\ a_{31}\sin\beta \end{bmatrix} \\ &\quad + V^2 \begin{bmatrix} b_{11}\cos\beta & 0 & b_{13}\cos\beta \\ 0 & b_{22}\cos\alpha & 0 \\ b_{31}\cos\beta & 0 & b_{33}\cos\beta \end{bmatrix} \begin{bmatrix} \delta A \\ \delta E \\ \delta R \end{bmatrix}\end{aligned} \tag{3.35}$$

$$\begin{bmatrix} L \\ D \\ S \end{bmatrix} = V^2 \begin{bmatrix} c_{21} + c_{22}\sin 2\alpha + c_{23}q \\ c_{11} + c_{12}\cos\alpha \\ c_{31}\sin 2\beta \end{bmatrix} + T_{max} \begin{bmatrix} \sin\alpha \\ -\cos\alpha\cos\beta \\ \cos\alpha\cos\beta \end{bmatrix} \delta T \tag{3.36}$$

This final model replaces Equations (3.1) to (3.5), where the overall model is drastically simplified, and still has room to predict a broad envelope of flight data. After incorporating into the main equations of motion, this model now has a total of 23 parameters, where each parameter is lumped and (other than mass and inertia) has no physical meaning.

Better Results from Kadet Senior: Using an Improved Data Set and the Consolidated Model

Once the underlying aerodynamic model is improved, the other source of improvement is the data itself. IMUs, as discussed earlier, consist of accelerometers, gyroscopes and magnetometers. They come in various grades, with the best being based on laser systems, that have virtually no drift. In fact, measurements from a laser ring gyro can be integrated to find the position for navigation in GPS denied environments. The IMU used on the Kadet Senior UAV is a MEMs based unit which is prone to noise and drift issues. To fix the drift and noise, particularly in measuring p, q and r in the first data set, an Extended Kalman Filter was designed with a constant acceleration motion model. This limits its use to less aggressive maneuvers, but also smooths the data for system identification purposes.

Further improvements on the system include better messaging protocols between the microcontroller and the QNX flight computer, proper calibration of the wind vanes and control surfaces as well as precise measurements of the aircraft mass just before take-off.

A subset of the improved data can be seen in Figures (3.17) to (3.20). The final system identification results are shown in Figure (3.21). Note that since the forces are reflected in ϕ, γ and ψ and the moments are reflected in p, q and r, the final model accurately predicts the aerodynamic moments and forces acting at any time on the Kadet Senior. This information can now be estimated on-board the flight computers using this model, and inputs from the control surface and wind sensors.

3.8 Conclusions

System identification methods using flight data have been studied extensively over the last few decades. These methods, which were initially designed for linear models and large

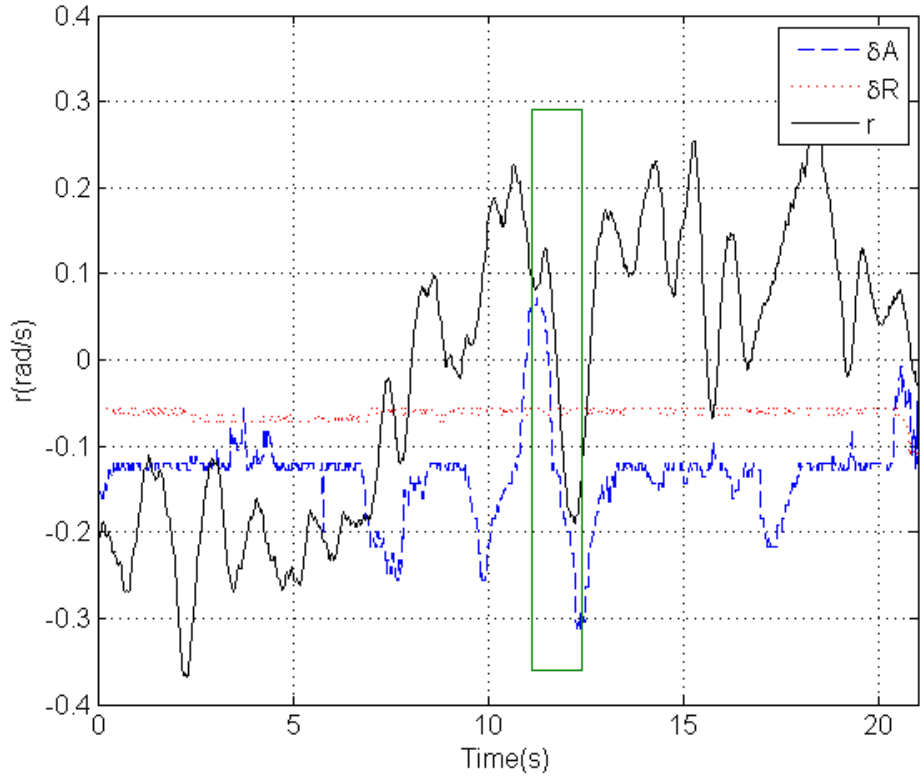

Figure 3.17: Data collected from the Kadet Senior UAV, showing strong correlation between ailerons, δA, and body axis yaw rate, r, (highlighted region). The rudder, δR, is kept constant. Where the ailerons do not conform to the yaw rate, the effect of wind on the tail is noticeable.

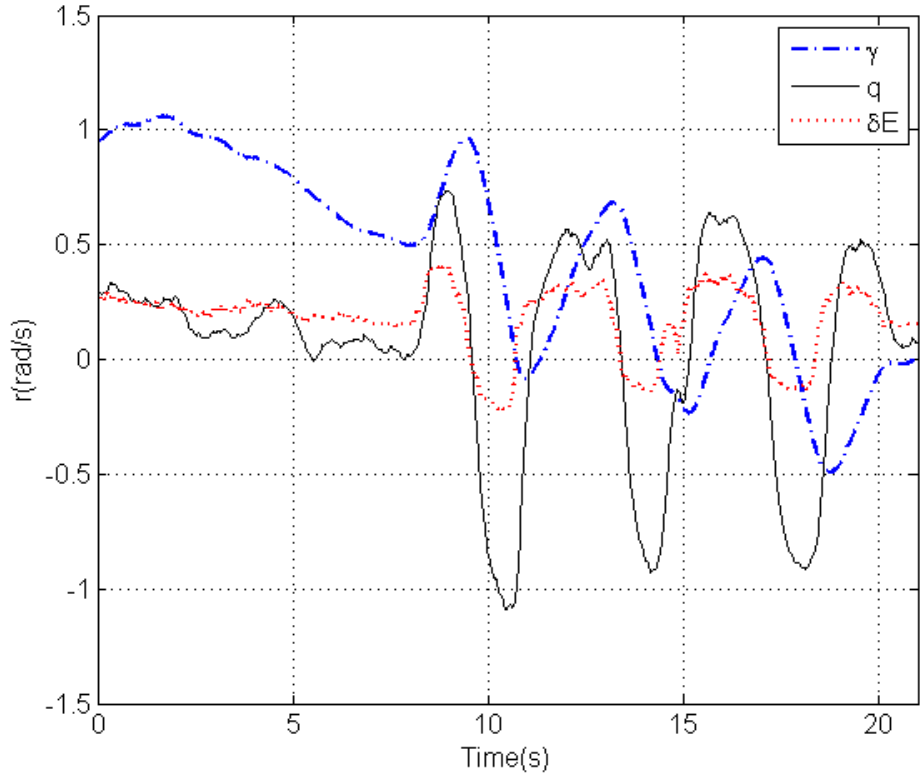

Figure 3.18: Data collected from the Kadet Senior UAV, showing correlations between elevator, δE, and body axis pitch rate, q and flight path climb angle, γ.

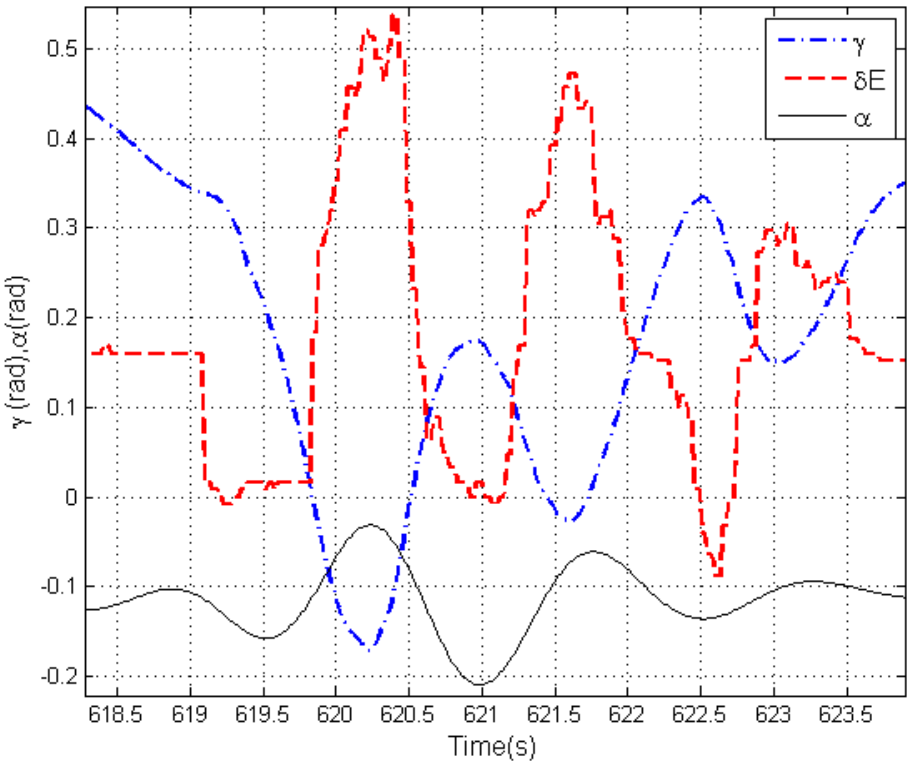

Figure 3.19: Data collected from the Kadet Senior UAV, showing correlations between elevator, δE, angle-of-attack, α and flight path climb angle, γ. α is measured using the custom designed vane assembly discussed earlier.

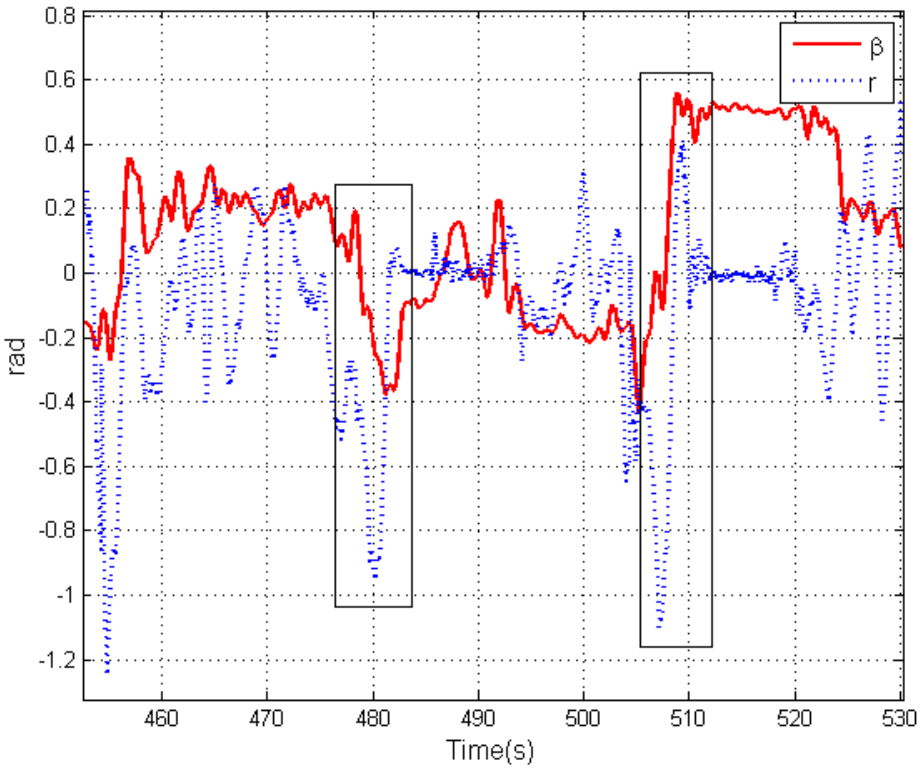

Figure 3.20: Data collected from the Kadet Senior UAV, showing how sudden changes in yaw rate affects the sideslip angle, β, (highlighted regions). β is measured using a custom designed vane assembly.

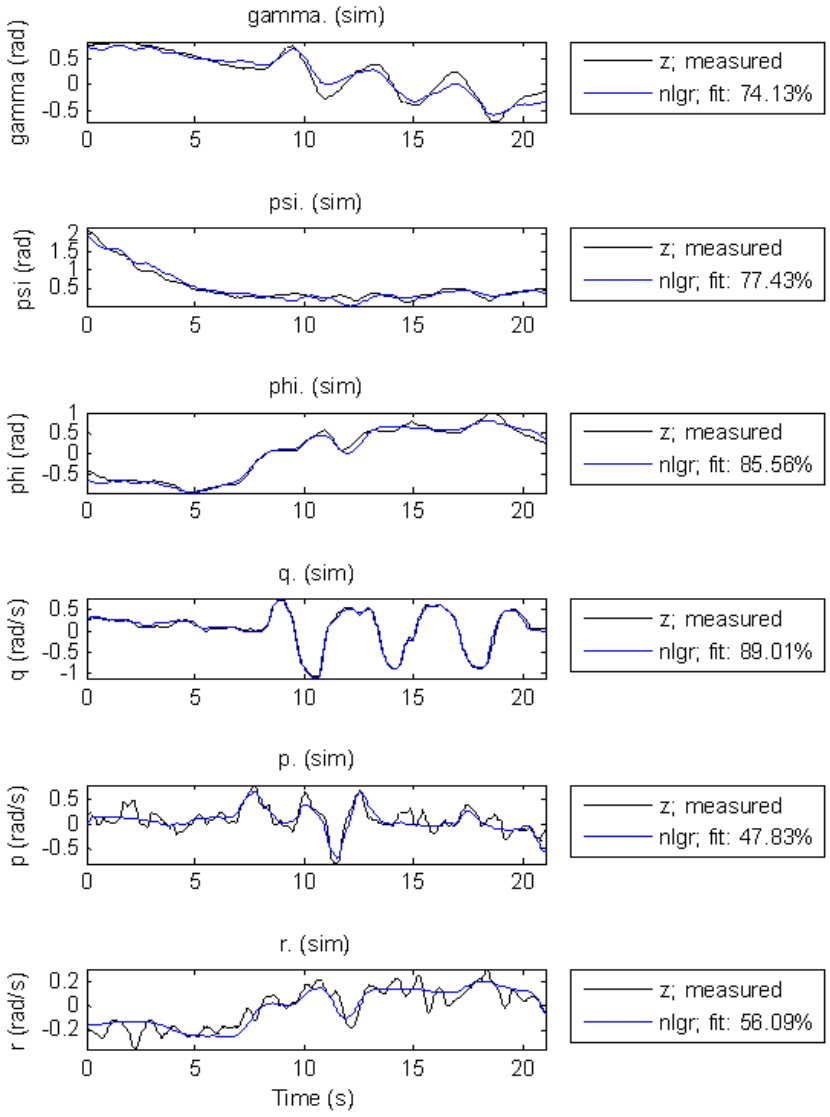

Figure 3.21: Drastic improvements in the fit and with only 23 parameters: using an Extended Kalman Filter on the flight computer and the consolidated model.

scale aircraft, can now be used for small scale Unmanned Aerial Vehicles. This has much to do with the rise in on-board and off-board computing power as well as the recent advent of light-weight MEMs based sensors.

A class of methods known as the Output Error Method are discussed in this chapter. Extensions to the basic method can be used to account for atmospheric turbulence. Initial results from the Kadet Senior UAV seemed promising and were improved by changing the base aerodynamic model and improvements in the hardware and software system.

The parameters identified in this chapter can now be used to design linear and non-linear controllers.

Chapter 4

Nonlinear Controller

4.1 Introduction

The problem of tracking an arbitrary moving ground target from an aircraft can be broken down into two main classes. In the first class, the target takes on a relatively steady motion with small changes in speed and direction. In the second class the target starts maneuvering in an aggressive fashion. Examples of this breakdown can be seen in car races, where the first class includes all straight sections and the second, includes the sharp turns. Another example can be observed during the coverage of a downhill ski event. Here, the first class will include all steady downward motions, and the second, will include all the challenging maneuvers.

With a known target position, steady motion can be tracked using linear control techniques that have been developed in the past. The control techniques in this chapter focus on the second class of tracking problems, where the aircraft is required to track aggressive trajectories. A base set of such trajectories can be programmed into the flight computer as motion primitives. The motion primitives can then be called from a high level planner whenever it foresees the need for an aggressive maneuver based on the info from the vision based target tracking discussed in Chapter (2). The high level planner will be responsible for stitching a reference trajectory in real time which will include linear and nonlinear portions. The nonlinear portions will be small segments of aggressive maneuvers where the aircraft will move through a predictable path with timing constraints.

It is expected that tracking such targets will push the aircraft to the extremes of its flight envelope, requiring investigation into nonlinear flight regimes where the aerodynamics

and actuator limitations play a big role. The motion primitives can be developed with predetermined reference trajectories that are generated using linear exosystems.

In addition to the models discussed in Section (3.2), two simpler models are presented in this chapter. The motivation for developing control laws for each model stems from the desire to ultimately implement the simplest controller that yields acceptable performance. All models are demonstrated to be feedback linearizable, and a common linear tracking control methodology is employed on each. The simplest model is a 3 degree of freedom (DOF) model that is used to establish the possibility of using nonlinear techniques for trajectory tracking, but the model does not capture, for example, nonlinearities in the relationship between lift, drag and vehicle speed or angle of attack. The second follows Hauser and Hindman [23], and describes a coordinated flight regime where the sideslip angle is deliberately maintained at zero, restricting the possible trajectories of the aircraft, but improving on the first model in terms of fidelity.

Since all the models are feedback linearizable, the tracking control approach used involves first performing the feedback linearization and then applying linear tracking control using the internal model principle of Francis and Wonham [16]. For this, linear exosystems are defined which allow tracking of elliptical and oscillatory trajectories, combinations of which can give a vast number of motion primitives. An alternative to this approach which relaxes the need for feedback linearization would be to define nonlinear tracking controllers per Isidori and Byrnes [30], however this method requires the solution of a partial differential equation for each trajectory under consideration. Khalil's robust tracking control could also be considered [35], however it relies on high-gain assumptions to ensure tracking which is impractical to implement.

It should be noted that the particular control approach presented in this work relies on the possibility of feedback linearization. This requirement has therefore led to the selection of a particular subset of the available dynamic models for aircraft, and has resulted in the omission of valid models [13, 56] which include coupling between forces and angles. The result of such coupling, which does indeed exist in practice, is that the system becomes non-minimum phase, as was demonstrated by Tomlin et al. [56], and therefore is no longer feedback linearizable. For non-minimum phase systems Devasia, Chen and Paden [10] presented a method for nonlinear tracking based on non-causal inversion. This aspect of the modeling and controller design remains an area for future work.

This chapter proceeds as follows. Section (4.2) presents the two new aircraft models in order of increasing complexity, and some representative linear target models are presented. The tracking controllers to be used with these models and the ones from Section (3.2) are defined in Section (4.3), and results are presented in Section (4.4). Finally, a brief

discussion of future directions is included in the final Section.

4.2 Aircraft and Target Models

The coordinate frames and nomenclature here follows from Section (3.2). In addition, here the aircraft position, $x \in \mathbb{R}^3$ is defined in an inertial frame $\mathbf{I}$, with $\dot{x}$ and $\ddot{x}$ being the velocity and acceleration respectively.

4.2.1 Simple Flight Model

This model assumes that direct control over the rates of change of velocity and flight path angles of climb and heading is possible. Let $V := \|\dot{x}\|$ and treat $\dot{V}$, $\dot{\gamma}$, $\dot{\psi}$ as control inputs where γ and ψ are the two flight path angles introduced above. Then the kinematics of the simple flight model can be modeled as

$$\dot{x} = \begin{bmatrix} V\cos(\gamma)\cos(\psi) \\ V\cos(\gamma)\sin(\psi) \\ -V\sin(\gamma) \end{bmatrix} \tag{4.1}$$

$$\begin{aligned} \dot{V} &= u_1 \\ \dot{\gamma} &= u_2 \\ \dot{\psi} &= u_3 \\ y &= x \end{aligned} \tag{4.2}$$

where u_1, u_2 and u_3 are control inputs and y is the output. This model entirely ignores, among other things, the roll dynamics of the vehicle, as well as any coupling between the three inputs.

4.2.2 Coordinated Flight Model

The second model depends on the assumption that the aircraft is restricted to coordinated flight, which requires that β, the sideslip angle, be zero for all time. In practice, coordinated flight is possible by implementing regulation of sideslip using the aircraft rudder. Let $v^w \in \mathbb{R}^3$ and $a^w \in \mathbb{R}^3$ denote, respectively, the aircraft velocity and acceleration expressed in the wind frame $\mathbf{F_w}$. The rotation from $\mathbf{F_w}$ to the inertial frame is defined by $R \in SO(3)$.

The rotation rates in wind axes are $\omega = (p_w, q_w, r_w) \in \mathbb{R}^3$ and the evolution of R in time is described by the classical equation

$$\dot{R} = R\hat{\omega} \tag{4.3}$$

where

$$\hat{\omega} = \begin{bmatrix} 0 & -r_w & q_w \\ r_w & 0 & -p_w \\ -q_w & p_w & 0 \end{bmatrix}.$$

The assumption of coordinated flight can be imposed with the constraint

$$\dot{x} = VRe_1 \tag{4.4}$$

where $e_1 = (1, 0, 0)^\top$. The control inputs are taken as the aircraft forward acceleration a_1^v, the upward acceleration a_3^v and the roll rate p_w. Let $g^w = R^\top g$ be the gravity vector $g = (0, 0, g_3)^\top$ rotated into $\mathbf{F}_w$. Then, differentiating Equation (4.4), we obtain

$$\ddot{x} = g + Ra^w \tag{4.5}$$

where $a^w = (a_1^w, 0, a_3^w)$ and the control inputs are $u = (u_1, u_2, u_3) = (p_w, a_1^w, a_3^w)$. The coordinated flight constraint imposes conditions on the evolution of the rotation rates in $\mathbf{F_w}$

$$\begin{aligned} q_w &= -\frac{a_3^w + g_3^w}{V} \\ r_w &= \frac{g_2^w}{V}. \end{aligned} \tag{4.6}$$

In summary, the coordinated flight model is given by Equations (4.3) to (4.5) with states $(x_1, x_2, x_3, \dot{x}_1, \dot{x}_2, \dot{x}_3, R)$. The state space dimension is 7 instead of 9 because the coordinated flight requirement imposes additional constraints on the motion of R [23].

4.2.3 Full Aerodynamic Model

This model is the same as described in Section (3.2). The kinematic model in Equation (4.1) can be used to generate the aircraft trajectory for the 9 state system.

4.2.4 Target Models

The internal model approach to tracking requires a reference trajectory generator that is given by a linear time-invariant (LTI) system. This trajectory generating system is called

an exosystem. These exosystems create the reference trajectories for the motion primitives discussed above.

Three such exosystems are defined below. The first one generates a fixed altitude elliptical trajectory which can find many uses in aerial reconnaissance or auto racing applications. The second, a periodic trajectory generator, can be used to chase targets up or downhill along a oscillatory path. The third, can be used to generate a variety of aggressive maneuvers by manipulating the rotation rates p, q and r.

Elliptical Trajectory

The desired elliptical trajectory is given by

$$\begin{aligned} x_{1ref} &= 100\cos(t) \\ x_{2ref} &= 50\sin(t) \\ x_{3ref} &= 5 \end{aligned} \tag{4.7}$$

where x_{1ref}, x_{2ref} and x_{3ref} represent the desired aircraft position in inertial coordinates. This trajectory can be generated by the following system

$$\begin{aligned} \dot{w}_1 &= w_2 \\ \dot{w}_2 &= -w_1 \\ \dot{w}_3 &= w_4 \\ \dot{w}_4 &= -w_3 \\ \dot{w}_5 &= 0 \end{aligned} \tag{4.8}$$

with initial conditions $w_1(0) = 100, w_2(0) = 0, w_3(0) = 0, w_4(0) = 50$ and $w_5(0) = 5$ and with $w_1 = x_{1_{ref}}$, $w_3 = x_{2_{ref}}$, and $w_5 = x_{3_{ref}}$.

Basic Ski Slope

A simple ski slope trajectory can be generated using a combination of periodic functions for V_{ref}, γ_{ref}, ψ_{ref} and β_{ref}. Setting V_{ref}, γ_{ref} as constants and β_{ref} to zero gives a trajectory with a constant climb angle that maintains zero sideslip through aggressive turns. These turns can be formulated using any dynamically feasible time varying functions for ψ_{ref} as

follows

$$\begin{aligned} V_{ref} &= 150 \\ \gamma_{ref} &= 5 \\ \psi_{ref} &= 5\cos\left(\frac{t}{10}\right) \\ \beta_{ref} &= 0 \end{aligned} \tag{4.9}$$

The reference trajectories from Equation (4.9) can be generated by the exosystem

$$\begin{aligned} \dot{w}_1 &= \dot{w}_2 = \dot{w}_5 = 0 \\ \dot{w}_3 &= w_4 \\ \dot{w}_4 &= -\frac{1}{100}w_3 \end{aligned} \tag{4.10}$$

where $w_1(0) = 150, w_2(0) = 5, w_3(0) = 5, w_4(0) = 0$, $w_5(0) = 0$ and $w_1 = V_{ref}$, $w_2 = \gamma_{ref}$, $w_3 = \psi_{ref}$, and $w_5 = \beta_{ref}$. This exosystem provides a simple example of how downhill ski slope trajectories can be represented by a linear exosystem, which can be extended to more complex reference trajectories as needed.

Aggressive Turns

The basic form is given by

$$\begin{aligned} p_{ref} &= 0.08\cos\left(\frac{t}{10}\right) \\ q_{ref} &= 0.04\cos\left(\frac{t}{10}\right) \\ r_{ref} &= 0.05 \end{aligned} \tag{4.11}$$

This trajectory can be generated by the following system

$$\begin{aligned} \dot{w}_1 &= w_2 \\ \dot{w}_2 &= -(1/100)w_1 \\ \dot{w}_3 &= w_4 \\ \dot{w}_4 &= -(1/100)w_3 \\ \dot{w}_5 &= 0 \end{aligned} \tag{4.12}$$

with initial conditions $w_1(0) = 0.08, w_2(0) = 0, w_3(0) = 0.04, w_4(0) = 0$ and $w_5(0) = 0.05$ and with $w_1 = p_{ref}$, $w_3 = q_{ref}$, and $w_5 = r_{ref}$.

4.3 Tracking Controllers

4.3.1 Inverse Dynamics and Output Tracking

Consider a system of the form

$$\begin{aligned} \dot{x} &= f(x) + \sum_{i=1}^{m} g_i(x)u_i \\ y_1 &= h_1(x) \\ &\vdots \\ y_m &= h_m(x) \end{aligned} \tag{4.13}$$

where $f(x), g_1(x), \ldots, g_m(x)$ are smooth vector fields and $h_1(x), \ldots, h_m(x)$ are smooth real-valued functions defined in a domain $D \subset \mathbb{R}^n$. By definition, the system in Equation (4.13) with output $y = (y_1, \ldots, y_m)$ has a vector relative degree of $\{r_1, \ldots, r_m\}$ at $x_0 \in D$ if

$$\begin{gathered} L_{g_j} L_f^k h_i(x) = 0 \\ \forall j \in \{1, \ldots, m\}, \quad \forall k < r_i - 1, \quad \forall i \in \{1, \ldots, m\} \end{gathered} \tag{4.14}$$

for all x in a neighborhood of x_0 and the matrix

$$B^*(x) = \begin{bmatrix} L_{g_1} L_f^{r_1-1} h_1(x) & \ldots & L_{g_m} L_f^{r_1-1} h_1(x) \\ L_{g_1} L_f^{r_2-1} h_2(x) & \ldots & L_{g_m} L_f^{r_2-1} h_2(x) \\ \vdots & \vdots & \vdots \\ L_{g_1} L_f^{r_m-1} h_m(x) & \ldots & L_{g_m} L_f^{r_m-1} h_m(x) \end{bmatrix} \tag{4.15}$$

is nonsingular at $x = x_0$ [29], [36].

Differentiating each component, y_i, of the output r_i times and using the output and its derivatives to partially define a coordinate transformation in a neighborhood of x_0 yields a system of the form

$$\begin{aligned} \dot{\xi}_1^i &= \xi_2^i \\ &\vdots \\ \dot{\xi}_{r_i-1}^i &= \xi_{r_i}^i \\ \dot{\xi}_{r_i}^i &= L_f^{r_i} h_i(x) + \sum_{j=1}^{m} L_{g_j} L_f^{r_i-1} h_i(x) u_j \end{aligned} \tag{4.16}$$

for all $i \in \{1, \ldots, m\}$. If $\sum_{i=1}^{m} r_i = n$, then the output y and its derivatives completely specify a coordinate transformation $\xi = T(x)$ where, in transformed coordinates, the system has the form as in Equation (4.16). However, in the case where $\sum_i r_i < n$, the coordinate transformation $T(x)$ is only partially defined by y and its derivatives. In order to complete the coordinate transformation we must find $n - \sum_i r_i$ additional functions and augment them to the output and its derivatives. In doing so we can obtain a valid coordinate transformation $(\eta, \xi) = T(x)$ defined in a neighborhood of x_0. In general one cannot impose any particular structure to the additional states η and their evolution will be described by a general equation of the form

$$\dot{\eta} = q(\xi, \eta) + p(\xi, \eta)u. \tag{4.17}$$

Setting $\eta = 0$ and $u = 0$ yields $\dot{\eta} = q(\xi, 0)$ which are the zero dynamics of the system in Equation (4.13). The system in Equation (4.13) is minimum phase if these zero dynamics have an asymptotically stable equilibrium point in the domain of interest.

When the output y yields a vector relative degree $\{r_1, \ldots, r_m\}$ with $\sum_i r_i < n$, then one can appeal to differential flatness or dynamic feedback linearization to obtain a system that is linear in transformed coordinates via the use of dynamic feedback. Specifically, in dynamic feedback linearization, sometimes called dynamic extension, state variables are introduced in the controller that correspond to a chain of integrators in order to obtain a relative degree

$$\sum_{i=1}^{m} \tilde{r}_i = n + v \tag{4.18}$$

where $\tilde{r}_i$, $i \in \{1, \ldots, m\}$ is the number of derivatives of y_i that must be taken before a control input appears and v is the number of additional states introduced in by the controller. In other words, by adding integrators in the controller, the appearance of the control input in the derivatives of y is delayed until the desired relative degree is obtained.

In either the static case when $\sum_i r_i = n$, or the dynamic case when $\sum_i \tilde{r}_i = n + v$, it is possible to define a regular feedback transformation that when applied to the system in Equation (4.16) yields a controllable linear system with m chains of integrators $\xi_1, \ldots, \xi_n$. The highest order derivatives from the system can be grouped together to obtain

$$\begin{bmatrix} \dot{\xi_{1_{r_1}}} & \dot{\xi_{2_{r_2}}} & \ldots & \dot{\xi_{m_{r_m}}} \end{bmatrix}^T = A^*(x) + B^*(x)u \tag{4.19}$$

where $B^*(x)$ (also known as the decoupling matrix) is as defined in Equation (4.15) and

$$A^*(x) = \begin{bmatrix} L_f^{r_1} h_1(x) \\ \vdots \\ L_f^{r_m} h_m(x) \end{bmatrix}. \tag{4.20}$$

Since B^* is nonsingular near x_0, the feedback law

$$u = B^*(x)^{-1}(v - A^*(x)), \tag{4.21}$$

where $v \in \mathbb{R}^m$ is an auxiliary control input, is well-defined in a neighborhood of x_0. When the controller in Equation (4.21) is applied to Equation (4.13) a controllable linear system is obtained

$$\dot{\xi} = \begin{bmatrix} \dot{\xi}_1^1 \\ \vdots \\ \dot{\xi}_{r_1}^1 \\ \vdots \\ \dot{\xi}_1^m \\ \vdots \\ \dot{\xi}_{r_m}^m \end{bmatrix} = A_1\xi + B_1 v. \tag{4.22}$$

Using the internal model approach by Wonham and Francis [16], controllers can now be designed that will track trajectories generated by LTI exosystems like those presented in Equations (4.8) and (4.10). To this end, this system is augmented by

$$\begin{aligned} \dot{w} &= A_2 w \\ e &= D_1\xi + D_2 w \end{aligned} \tag{4.23}$$

where ξ is the state of the system in Equation (4.22), w is the state of the exosystem and e is the tracking error. The problem is solvable if and only if A_2 has only unstable eigenvalues, (A_1, B_1) is stabilizable and there exist matrices X and U such that

$$\begin{aligned} D_1 X + D_2 &= 0 \\ A_1 X - X A_2 + B_1 U &= 0 \end{aligned} \tag{4.24}$$

Once X and U are found, the resulting controller takes the form

$$v = F_1 q + F_2 w \tag{4.25}$$

where F_1 is chosen such that $A_1 + B_1 F_1$ is Hurwitz and $F_2 = U - F_1 X$.

4.3.2 Simple Flight Model

The kinematic aircraft model from Section (4.2.1) has a well defined vector relative degree of $\{2, 2, 2\}$ at each point in the regions of interest for our application. To see this, note that the decoupling matrix B^* is given by

$$B^* = \begin{bmatrix} \cos\gamma\cos\psi & -V\cos\psi\sin\gamma & -V\cos\gamma\sin\psi \\ \cos\gamma\sin\psi & -V\sin\gamma\sin\psi & V\cos\gamma\cos\psi \\ -\sin\gamma & -V\cos\gamma & 0 \end{bmatrix}. \tag{4.26}$$

This matrix is singular if and only if $V^2\cos\gamma = 0$, which corresponds to either a zero velocity or a climb angle of $\gamma = \frac{\pi}{2}$. The decoupling matrix is therefore invertible during normal flight conditions. We conclude that the output $y = x$ yields a well-defined relative degree almost everywhere and $\sum_{i=1}^{3} r_i = 6$ which implies that the system is feedback linearizable via a static controller. The corresponding coordinate transformation, as discussed in section (4.3.1), is given by

$$\begin{bmatrix} \xi_1^1 \\ \xi_2^1 \\ \xi_1^2 \\ \xi_2^2 \\ \xi_1^3 \\ \xi_2^3 \end{bmatrix} = \begin{bmatrix} x_1 \\ \dot{x_1} \\ x_2 \\ \dot{x_2} \\ x_3 \\ \dot{x_3} \end{bmatrix} \tag{4.27}$$

and is valid in a neighborhood of any point at which $V \neq 0$ and $\gamma \neq \frac{\pi}{2}$. The static feedback linearizing control law is given by

$$u = B^{*-1}v \tag{4.28}$$

because $A^* = 0$ in this case. The auxiliary control input v can be designed for tracking an elliptical trajectory. The exosystem in Equation (4.8) results in a tracking error

$$e = \begin{bmatrix} w_1 - \xi_1^1 & w_3 - \xi_1^2 & w_5 - \xi_1^3 \end{bmatrix}^\top. \tag{4.29}$$

By appropriately defining D_1 and D_2 based on Equation (4.29), the matrices X and U can be found by solving Equation (4.24). Furthermore, F_1, can be suitably picked depending on the magnitude of gains required for minimizing e and v can then be computed using Equation (4.25).

4.3.3 Coordinated Flight Model

The coordinated flight model with 7 states $(x_1, x_2, x_3, \dot{x}_1, \dot{x}_2, \dot{x}_3, R)$ and output $y = (x_1, x_2, x_3)$ as described in Section (4.2) has a well defined vector relative degree of $\{2, 2, 2\}$ but $\sum_{i=1}^{3} r_i = 6 < 7$ and therefore static feedback cannot be used to feedback linearize this system. In order to obtain a fully linear system in transformed coordinates, two new states need to be added to the system by adding integrators on two of the inputs. The resulting system will have 9 states $(x, \dot{x}, R, a_1^w, a_3^w)$ with a well defined vector relative degree of $\{\tilde{r}_1, \tilde{r}_2, \tilde{r}_3\} = \{3, 3, 3\}$. The system can equivalently be expressed in coordinates $(x_1, \dot{x}_1, \ddot{x}_1, x_2, \dot{x}_2, \ddot{x}_2, x_3, \dot{x}_3, \ddot{x}_3)$ as shown in [23]. The new control inputs are given by

$$u = R \begin{bmatrix} \dot{a}_1^v \\ p_w \\ \dot{a}_3^v \end{bmatrix}. \tag{4.30}$$

The A^* and B^* matrices can be formed using Equations (4.20) and (4.15) respectively which results in

$$A^* = R \begin{bmatrix} q_w a_3^v \\ r_w a_1^v \\ -q_w a_1^v \end{bmatrix}, \; B^* = R \begin{bmatrix} 1 & 0 & 0 \\ 0 & -a_3^v & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.31}$$

Finally, a feedback linearizing control law is generated using Equation (4.21) and (4.30). A linear tracking controller can now be designed for v as before. The tracking error, e, for the elliptical exosystem is also given by Equation (4.29).

4.3.4 Full Aerodynamic Model

Taking $y = (y_1, \ldots, y_4) = (V, \gamma, \psi, \beta)$ as the output we get $r_1 = r_2 = r_3 = r_4 = 1$. The sum of these relative degrees is less than the number of states (9) in the system which is a direct result of the throttle input (δT) appearing in the force terms, L, D and S. Using the previous discussion on zero dynamics, this implies 5 uncontrollable and potentially unobservable internal states. Extending the system with two additional states and replacing δT with $\delta \ddot{T}$ as the new control input ensures differential flatness by delaying the appearance of the control inputs in the output derivatives. The new control input is

$$u = \begin{bmatrix} \delta \ddot{T} & \delta E & \delta A & \delta R \end{bmatrix}^\top. \tag{4.32}$$

With these re-defined control inputs, the decoupling matrix is given by

$$B^* = \begin{bmatrix} \dddot{V}_{\delta\ddot{T}} & \dddot{V}_{\delta E} & \dddot{V}_{\delta A} & \dddot{V}_{\delta R} \\ \ddot{\gamma}_{\delta\ddot{T}} & \ddot{\gamma}_{\delta E} & \ddot{\gamma}_{\delta A} & \ddot{\gamma}_{\delta R} \\ \dddot{\psi}_{\delta\ddot{T}} & \dddot{\psi}_{\delta E} & \dddot{\psi}_{\delta A} & \dddot{\psi}_{\delta R} \\ 0 & 0 & \ddot{\beta}_{\delta A} & \ddot{\beta}_{\delta R} \end{bmatrix} \tag{4.33}$$

where each entry is expressed using partial derivatives of $\dddot{V}$, $\ddot{\gamma}$, $\dddot{\psi}$ and $\ddot{\beta}$ with respect to each control input.

These higher order derivatives are extremely complex and their length prohibits them from being reproduced here with clarity. An analytical check of the singularity conditions for the matrix in Equation (4.33) is therefore close to impossible. The matrix was however checked numerically to be nonsingular in the region of interest when tracking the basic ski slope reference trajectories from Section (4.2.4). This implies that the system yields a well defined vector relative degree of $\{\tilde{r}_1, \tilde{r}_2, \tilde{r}_3, \tilde{r}_4\} = \{3, 3, 3, 2\}$. Using Equation (4.21), and an appropriate A^* term (not shown here for brevity), a feedback linearizing controller is formed. For tracking control, the error can be defined as

$$e = \begin{bmatrix} w_1 - \xi_1^1 & w_2 - \xi_1^2 & w_3 - \xi_1^3 & w_5 - \xi_1^4 \end{bmatrix}^T \tag{4.34}$$

where ξ_1^1, ξ_1^2, ξ_1^3 and ξ_1^4 refer to V, γ, ψ and β respectively. The controller can then be completed, with results for a BAC-221 high performance aircraft shown in the next section.

4.3.5 Subset of the Consolidated Model

A subset of the 9 state consolidated model from Section (3.7.1) can also be used to control the rotation rates of the aircraft and produce additional motion primitives. With a selection of 3 states (p, q, r), output $y = (p, q, r)$ and input $u = (\delta E, \delta A, \delta R)$, the model has a well defined vector relative degree of $\{1, 1, 1\}$ and $\sum_{i=1}^{3} r_i = 3$. This implies that a static feedback linearization is sufficient and can be used with the model identified for the Kadet Senior. This selection of states assumes that a low level linear controller maintains V, which can be easily accomplished as the only inputs required here are δE, δA and δR, leaving δT decoupled for velocity control. V does not need to be actively changed and can be maintained at a constant level. α and β can be kept in check by subtracting or adding to δR and δA. This may not be necessary in calm weather due to the stability provided by the tail and the inherent stability of the platform.

The A^* and B^* matrices can be formed using Equations (4.20) and (4.15) respectively which results in

$$A^* = \begin{bmatrix} a_{11}\sin\beta V^2 + (a_{13}p + a_{12}r)V + qr(I_{yy} - I_{zz}) \\ (a_{21} + a_{22}\sin\alpha)V^2 + a_{23}qV - pr(I_{xx} - I_{zz}) \\ a_{31}\sin\beta V^2 + (a_{33}p + a_{32}r)V + pq(I_{xx} - I_{yy}) \end{bmatrix} \tag{4.35}$$

$$B^* = \begin{bmatrix} 0 & V^2 b_{11}\cos\beta & V^2 b_{13}\cos\beta \\ V^2 b_{22}\cos\alpha & 0 & 0 \\ 0 & V^2 b_{31}\cos\beta & V^2 b_{33}\cos\beta \end{bmatrix} \tag{4.36}$$

The tracking error is the same as Equation (4.29) and applies just as well to Equation (4.12). The results of this simple yet powerful controller as applied to the Kadet Senior UAV can be seen in the next section.

4.4 Results

Simulation results for trajectory tracking control of the simple flight model are shown in Figure (4.1). The aircraft position achieves perfect tracking of the elliptical exosystem described in section (4.2.4), and also rejects errors in initial conditions smoothly. To implement this controller, the aircraft is assumed to have a low level autopilot which would provide the required V, γ and ψ to track the trajectory. The scale of the reference ellipse and the time to traverse it can be adjusted based on the performance limitations of the aircraft.

Simulation result for the coordinated flight model with the elliptical exosystem is shown in Figure (4.2). The aircraft rapidly reduces the initial errors and achieves perfect tracking. As with the simple flight model, this controller can be used where an autopilot is already present that can command the required forward and vertical acceleration and the required roll rate a_1^v, a_3^v, p_w, respectively. Since the model doesn't account for aircraft limitations, these control outputs can be adjusted by altering the flight path as required. As an example, an autopilot could be developed that would regulate the sideslip to zero at all times through rudder deflection, δR. Similarly, a_1^v and a_3^v can be generated through throttle control, δT, and elevator deflection, δE, respectively. p_w can be produced by actuating the ailerons δA.

The BAC-221 aerodynamic model (developed in Section 3.5) can also be tested with the proposed control strategy and the basic ski model exosystem. The final trajectory result

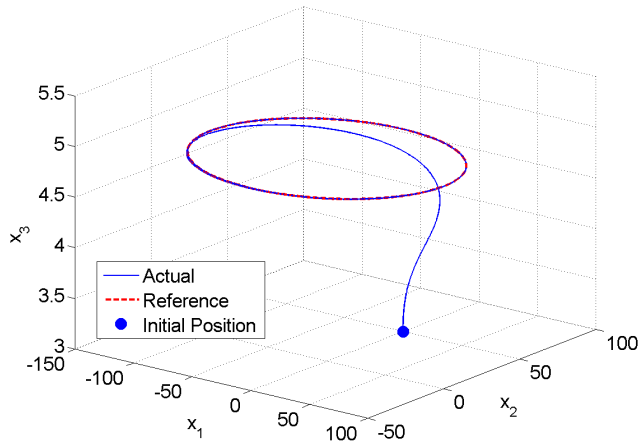

Figure 4.1: Simulation Results for the Simple Flight Model.

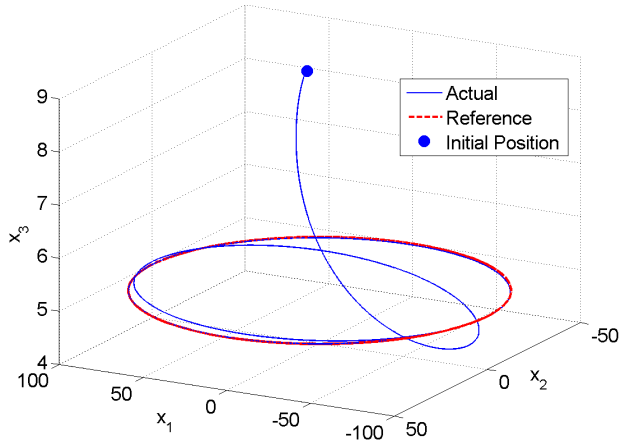

Figure 4.2: Simulation Results for the Coordinated Flight Model.

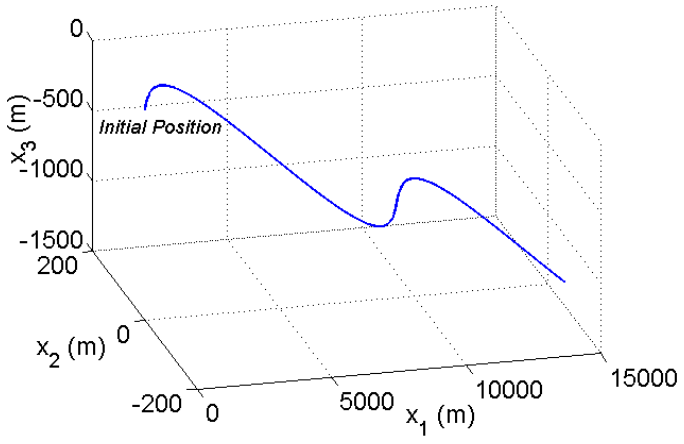

Figure 4.3: Aircraft trajectory over a basic ski-slope simulated using the BAC-221 fully aerodynamic model and the proposed controller.

is shown in Figure (4.3), which very precisely tracks the reference trajectory. The sideslip angle can be seen in Figure 4.4. Here the controller is trying to maintain coordinated flight by actively regulating the angle to zero through a combined effort of rudder and aileron deflection. Note that during the sharp turns, peaks can be seen which implies that some sideslip ($< 0.001°$) does inevitably occur.

One drawback in the proposed controller, when applied to the full aerodynamic model, is that it is not robust to small deviations from the reference. This implies a small domain of attraction around the trajectory. Hence, this approach maybe difficult to implement in practice, but could be addressed by adding a robust control component to the linear controller.

The best results are achieved when the controller is applied to the subset of the consolidated model. This scheme is a compromise. The overtly complex A^* and B^* matrices from the full 9 state model capture the entire aircraft model but are too rigid to accommodate deviations from initial conditions and require measurements of higher order derivatives, which are very difficult to determine accurately from MEMs based sensors. The very simple controllers from Sections (4.3.2) and (4.3.3) depend on complex low level autopilots that will have to meet demands for acceleration, velocity and attitude angles as the aircraft progresses through the reference trajectory. The subset model on the other hand connects with the control surfaces directly, and is still able to perform with a very simple low level autopilot whose demands don't change over time. This controller was applied to the Kadet

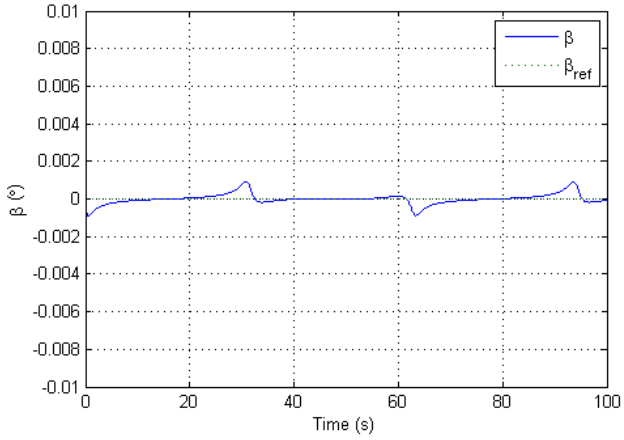

Figure 4.4: β being regulated to zero for the BAC-221.

Senior model that was identified using real flight data. Results with small variations in the exosystem from Equation (4.12) can be seen in Figures (4.5) to (4.7).

The Kadet Senior UAV was test flown with linear controllers only. This was mainly to verify the custom designed autopilot hardware, the onboard QNX system, long range radio links, ground station, remote debugging and safety takeover systems. The UAV itself is not structurally sound to perform aggressive maneuvers. Results from the linear autopilot tests are shown in Figures (4.8) to (4.10).

4.5 Conclusions

In this work, a nonlinear controller is designed which achieves trajectory tracking for three progressively complex models of an aircraft. Each of these models can be used to design autopilots provided that an underlying low level controller is already present that ensures that the input requirements are met. The controllers for the basic and coordinated flight models can be scaled up or down based on the type of aircraft being used. For a small scale UAV, where the thrust to weight ratio is usually more than one, fairly aggressive trajectories should be within reach. The controller for the full aerodynamic model needs to be built ground up for the specific aircraft after a thorough system identification. It may also require the inclusion of a robust control component to achieve tracking in non-ideal conditions. However, this controller would only require a simple underlying controller

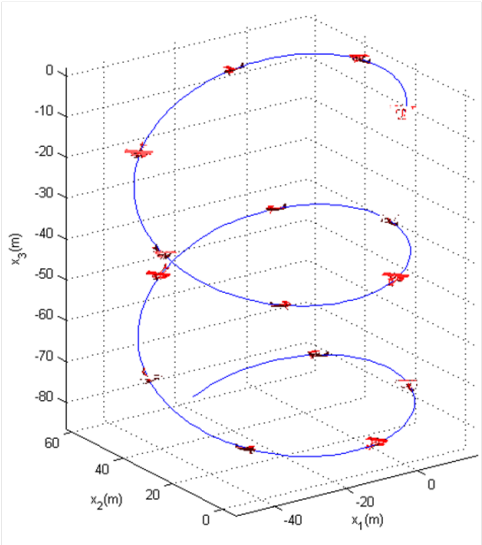

Figure 4.5: Motion primitive 1. Nonlinear controller applied to the Kadet Senior model: fully controlled downward spiral. Note, for illustrative purposes z is positive up here.

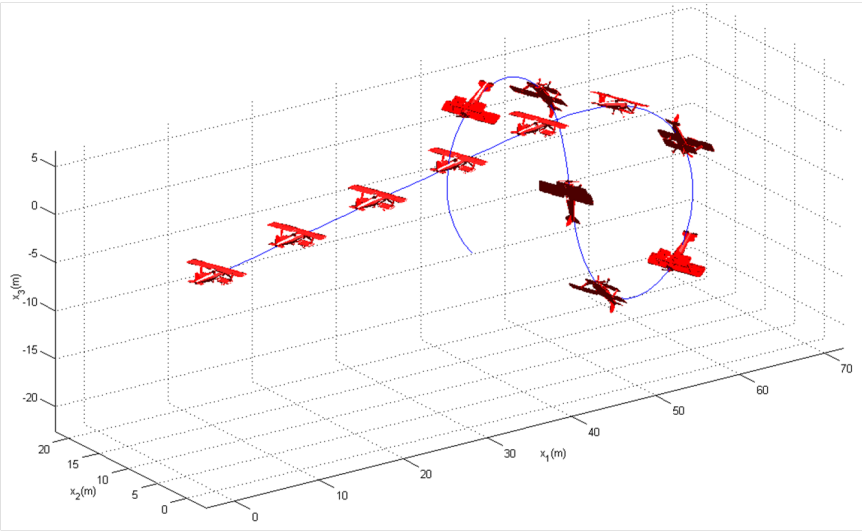

Figure 4.6: Motion primitive 2. Nonlinear controller applied to the Kadet Senior model: undergoing an aggressive eight pattern. Note, for illustrative purposes z is positive up.

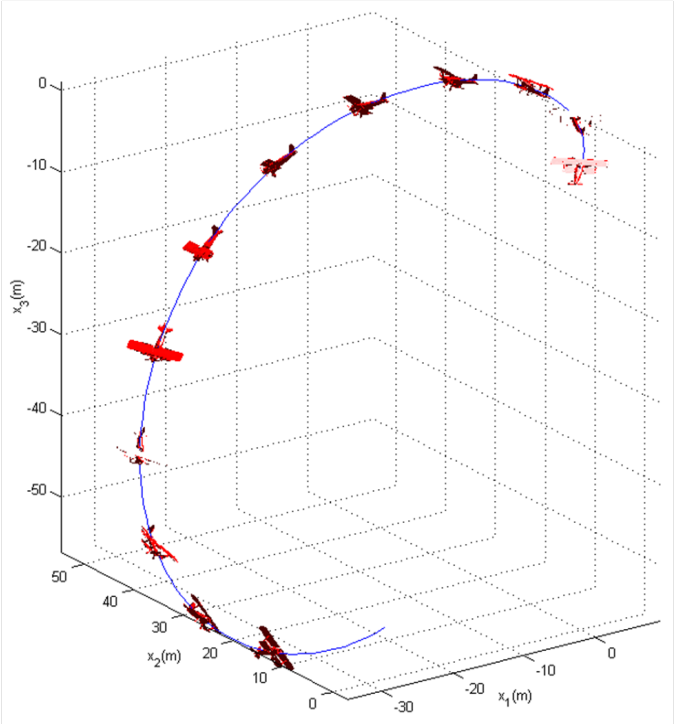

Figure 4.7: Motion primitive 3. Nonlinear controller applied to the Kadet Senior model: a displacement roll, usually used during dog fighting maneuvers, can be used in this application to chase agile targets susceptible to frequent changes in direction. Note, for illustrative purposes z is positive up.

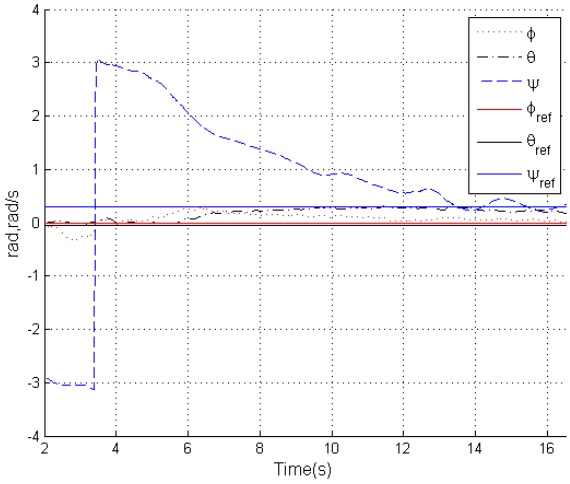

Figure 4.8: Flight test results: using a decoupled linear controller to level the aircraft and achieve a desired heading. The linear controllers used were Proportional-Derivative (PD), whose gains were tuned midflight to control the attitude states.

Figure 4.9: Kadet Senior taxiing at the flying field.

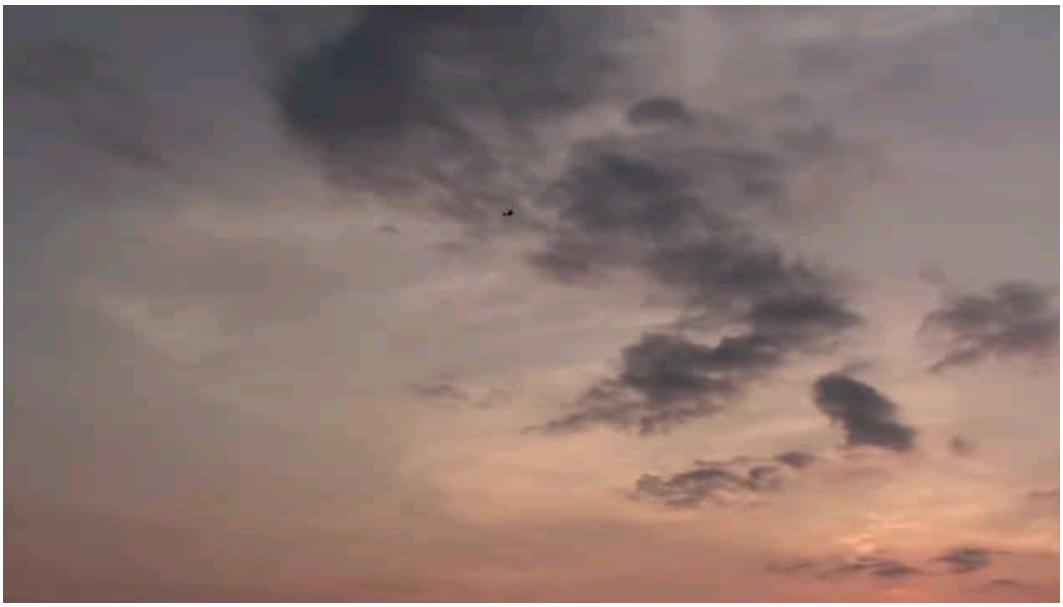

Figure 4.10: Kadet Senior airborne and in autopilot mode. A higher altitude is required for safe operation: in case of an autopilot failure, the ground pilot can resume control and recover the vehicle before a crash landing.

to meet control surface deflections as opposed to the other two models which require controllers for flight path angles and accelerations. The controller for the subset of the consolidated model is relatively simple and does not need robust control components. It only requires a simple low level autopilot.

Future work in this area involves flight tests using all aircraft models implemented on the Kadet Senior UAV after sufficient structural enhancements. New tracking controls, with less computationally intensive control laws will also be considered. One of these includes the output tracking method developed by Devasia, Chen and Paden [10], where the inverse is solved numerically. The high level system, which stiches these trajectories for meaningful real-time tracking is also an area of active research within the WAVE Lab.

Chapter 5

Conclusion

This thesis presents three crucial components towards real-time tracking of high speed, aggressive targets using an Unmanned Aerial Vehicle.

The vision system employs the latest in off-the-shelf parallel computing technology to achieve robust real-time target tracking. It uses the new OpenCL GPGPU framework, and AMD's new line of stream computing GPUs and Fusion processors. This processor, with a GPU and CPU on a single silicon die, provides a lightweight and low power computing solution for vision based avionics. With its large number of parallel cores, it rivals the computing power of high-end desktop based CPUs. Algorithms that are slow performers on a CPU, but hold the potential to be parallelized can be ported over to such units. A state-of-the-art vision tracking algorithm is described, together with implementation details using OpenCL and a HD5870 GPU. The results convert a once offline-only technique, to online real-time tracking. Object tracking is performed under rotation, scale changes and occlusion. The system is tested on a key-chain target undergoing rotations and transformations and subsequently on an aerial video stream of a high speed RC aircraft.

System identification is performed using the well-established Maximum Likelihood method, also known as the Output Error Method, to develop a wide envelope and high-fidelity model for the Kadet Senior. The techniques are first tested using wind tunnel data from a 1970's British jet aircraft known as the BAC-221. The data is used to create mathematical models of the jet, which are in turn used to simulate and extract appropriate data for performing system identification. The result of the identification procedure is compared with the original wind-tunnel based model to prove its effectiveness. A Kadet Senior model aircraft is outfitted with custom sensors and avionics running the latest QNX real-time operating system to acquire clean and reliable flight data. Control inputs are de-

signed to capture the different flight modes and are flown by a ground pilot. A ground station monitors the progress while the pilot is flying. All maneuvers are completed in one flight without the need to land. The data is processed offline over a 9 state model, with different underlying aerodynamic assumptions. The best set of parameters results in a relatively simple model that is usable for feedback linearization.

This model forms the basis for nonlinear controls techniques which are shown to be effective for generating aggressive motion primitives. The control law uses feedback linearization to cancel the known nonlinearities from the high fidelity system identification. Linear tracking control using the internal model approach is then applied to the resulting linear system, to direct the aircraft on a prescribed trajectory. The control laws are simulated on two general aircraft kinematic models, on the BAC-221 model and then finally on the Kadet Senior model identified earlier. A motion primitive bank is created, with examples of aggressive trajectories shown.

Linear autopilot controls are implemented on the Kadet Senior using the same custom designed avionics, showcasing the autonomous UAV and providing a safe platform for future controls development.

The vision system can be improved by using advanced search methods and incorporating predictive motion models on the target. Its adaptation to an AMD Fusion APU will allow the tracking algorithm to be flown onboard the aircraft. Cameras can be mounted on a two axis gimbal to consistently track the target even when repositioning of the aircraft is not possible.

The nonlinear controller can be improved by adding robust control techniques to maintain tracking during atmospheric disturbances. The proposed exosystems allow a wide variety of trajectories to be generated by changing the initial conditions. This approach works because motion primitives can be generated ahead of time and have the advantage of being fine tuned during experimental flights.

Future research will focus on integrating the components developed here into a complete aggressive maneuver tracking UAV solution. A high level planner will need to be devised to maintain target visibility during the tracking mission. This planner would use the provided motion primitives to abstract out the controls required during aggressive maneuvers and meet the demands created by the vision system. In a farther future, the system will be reliable enough to be accepted as a mainstream tool for covering live action sports, shooting high speed car chases and providing autonomous pursuit solutions for law enforcement.

References

[1] APU101: All about AMD Fusion Accelerated Processing Units and how they're changing, well, everything. Technical report, AMD Inc., 2011.

[2] Programming guide, AMD Accelerated Parallel Processing, OpenCL. Technical report, AMD Inc., 2011.

[3] S. A. Al-Hiddabi and N. H. McClamroch. Tracking and maneuver regulation control for nonlinear nonminimum phase systems: application to flight control. *IEEE Transactions on Control Systems Technology*, 10(6):780–792, 2002.

[4] P. Babenko and M. Shah. MinGPU: a minimum gpu library for computer vision. *Journal of Real-Time Image Processing*, 3(4):255–268, 2008.

[5] M. E. Campbell and W. W. Whitacre. Cooperative tracking using vision measurements on SeaScan UAVs. *Control Systems Technology, IEEE Transactions on*, 15(4):613–626, July 2007.

[6] J. Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(6):679–698, Nov. 1986.

[7] G. Chowdhary and R. V. Jategaonkar. Aerodynamic parameter estimation from flight data applying extended and unscented kalman filter. *Aerospace Science and Technology*, 14(2):106–117, 2010.

[8] M. M. Chu. Marketing Presentation: GPU computing: past, present and future with ATI Stream Technology, March 2010.

[9] NVIDIA Corporation. NVIDIA CUDA compute unified device architecture programming guide. `http://developer.nvidia.com/cuda`, Jan. 2007.

[10] S. Devasia, D. Chen, and B. Paden. Nonlinear inversion-based output tracking. *IEEE Transactions on Automatic Control*, 41(7):930–942, 1996.

[11] G. Ducard and H. P. Geering. Airspeed control for unmanned aerial vehicles: a nonlinear dynamic inversion approach. In *Proceedings of 16th Mediterranean Conference on Control and Automation*, pages 676–681, June 2008.

[12] B. Edwards, J. Archibald, W. Fife, and D. J. Lee. A vision system for precision mav targeted landing. In *Proceedings of International Symposium on Computational Intelligence in Robotics and Automation*, pages 125–130, June 2007.

[13] B. Etkin. *Dynamics of atmospheric flight*. John Wiley & Sons Inc, 1972.

[14] J. P. Farrugia, P. Horain, E. Guehenneux, and Y. Alusse. GPUCV: A framework for image processing acceleration with graphics processors. In *Proceedings of IEEE International Conference on Multimedia and Expo*, pages 585–588, July 2006.

[15] W. Forstner and B. Moonen. A metric for covariance matrices. Technical report, Dept. of Geodesy and Geoinformatics, Stuttgart University, 1999.

[16] B. A. Francis and W. M. Wonham. The internal model principle for linear multivariable regulators. *Applied Mathematics & Optimization*, 2(2):170–194, 1975.

[17] G. F. Franklin, J. D. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems*. Prentice Hall, Upper Saddle River, NJ, 5th edition, 2008.

[18] J. Fung and S. Mann. Using graphics devices in reverse: GPU-based image processing and computer vision. In *Proceedings of IEEE International Conference on Multimedia and Expo*, pages 9–12, April 2008.

[19] G. H. Golub and H. A. van der Vorst. Eigenvalue computation in the 20th century. *Journal of Computational and Applied Mathematics*, 123(1-2):35–65, 2000.

[20] D. M. Halford. *Aerodynamic Data for the BAC 221 up to a Mach Number of 0.955 as Measured in Wind Tunnel Tests*. Her Majesty's Stationery Office, London, 1972.

[21] P. G. Hamel and R. V. Jategaonkar. The role of system identification for flight vehicle applications - revisited. In *Proceedings of RTO SCI Symposium on "System Identification for Integrated Aircraft Development and Flight Testing"*, March 1998.

[22] M. J. Harvey and G. D. Fabritiis. Swan: A tool for porting CUDA programs to OpenCL. *Computer Physics Communications*, 182(4):1093–1099, 2011.

[23] J. Hauser and R. Hindman. Aggressive flight maneuvers. In *Proceedings of 36th IEEE Conference on Decision and Control*, pages 4186–4191, Dec. 1997.

[24] J. Huang, S. P. Ponce, S. I. Park, Y. Cao, and F. Quek. GPU-accelerated computation for robust motion tracking using the CUDA framework. In *Proceedings of 5th International Conference on Visual Information Engineering*, pages 437–442, Aug. 2008.

[25] K. Ilif. Aircraft parameter estimation. Technical Report NASA TM 88281, Ames Research Center, National Aeronautics and Space Administration, Edwards, California 93523, 1987.

[26] K. Ilif and R. Maine. Practical aspects of using a maximum likelihood estimation method to extract stability and control derivatives from flight data. Technical Report NASA TN D-8209, Dryden Flight Research Center, National Aeronautics and Space Administration, Edwards, California 93523, April 1976.

[27] AMD Inc. Heterogeneous computing, OpenCL and the ATI Radeon HD5870 ("Evergreen") architecture, 2010.

[28] AMD Inc. OpenCL and the AMD APP SDK. `http://developer.amd.com/documentation/articles/pages/OpenCL-and-the-AMD-APP-SDK.aspx`, April 2011.

[29] A. Isidori. *Nonlinear Control Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1995.

[30] A. Isidori and C. I. Byrnes. Output regulation of nonlinear systems. *IEEE Transactions on Automatic Control*, 35(2):131–140, 1990.

[31] F. James, S. Mann, and C. Aimone. Openvidia: Parallel GPU computer vision. In *Proceedings of ACM Multimedia*, Nov. 2005.

[32] R. V. Jategaonkar. *Flight Vehicle System Identification - A Time Domain Methodology*. American Institute of Aeronautics and Astronautics, 2006.

[33] R. V. Jategaonkar and E. Plaetschke. Estimation of aircraft parameters using filter error methods and extended kalman filter. Technical Report DFVLR-FB 88-15, Institute of Flight Systems, German Aerospace Center, Lilienthalplatz, Braunschweig, March 1988.

[34] R. V. Jategaonkar and E. Plaetschke. Algorithms for aircraft parameter estimation accounting for process and measurement noise. *Journal of Aircraft*, 26(4):360–372, April 1989.

[35] H. K. Khalil. Robust servomechanism output feedback controllers for feedback linearizable systems. *Automatica*, 30(10):1587–1599, 1994.

[36] H. K. Khalil. *Nonlinear Systems*. Prentice Hall, 1996.

[37] J. Kim, M. Hwangbo, and T. Kanade. Realtime affine-photometric klt feature tracker on GPU in CUDA framework. In *Proceedings of 12th International Conference on Computer Vision Workshops*, pages 886–893, Oct. 2009.

[38] M. Lalonde, D. Byrns, L. Gagnon, N. Teasdale, and D. Laurendeau. Real-time eye blink detection with GPU-based sift tracking. In *Proceedings of Fourth Canadian Conference on Computer and Robot Vision*, pages 481–487, May 2007.

[39] S. H. Lane and R. F. Stengel. Flight control design using non-linear inverse dynamics. *Automatica*, 24(4):471–483, 1988.

[40] P. Li and L. Xiao. Mean shift parallel tracking on GPU. In *Proceedings of 4th Iberian Conference on Pattern Recognition and Image Analysis*, pages 120–127, June 2009.

[41] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[42] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of DARPA Image Understanding Workshop*, pages 121–130, April 1981.

[43] Y. Luo and R. Duraiswami. Canny edge detection on NVIDIA CUDA. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8, June 2008.

[44] L. Meng, L. Li, and S. M. Veres. Aerodynamic parameter estimation of an unmanned aerial vehicle based on extended kalman filter and its higher order approach. In *Proceedings of 2nd International Conference on Advanced Computer Control (ICACC)*, volume 5, pages 526–531, March 2010.

[45] M. Naruoka, T. Hino, R. Nakagawa, T. Tsuchiya, and S. Suzuki. System identification of small UAVs with mems-based avionics. In *Proceedings of AIAA Infotech@Aerospace Conference*, Seattle, Washington, April 2009.

[46] F. Porikli. Achieving real-time object detection and tracking under extreme conditions. *Journal of Real-Time Image Processing*, 1(1):33–40, 2006.

[47] F. Porikli and O. Tuzel. Covariance tracking using model update based on lie algebra. Technical Report TR2005-127, Mitsubishi Electric Research Laboratories, June 2006.

[48] F. Quek and R. Bryll. Vector coherence mapping: A parallelizable approach to image flow computation. In Roland Chin and Ting-Chuen Pong, editors, *Computer Vision ACCV'98*, volume 1352 of *Lecture Notes in Computer Science*, pages 591–598. Springer Berlin / Heidelberg, 1997.

[49] S. Saripalli, J. F. Montgomery, and G. S. Sukhatme. Visually guided landing of an unmanned aerial vehicle. *Robotics and Automation, IEEE Transactions on*, 19(3):371–380, June 2003.

[50] J. Shi and C. Tomasi. Good features to track. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 593–600, June 1994.

[51] T. Shimobaba, T. Ito, N. Masuda, Y. Ichihashi, and N. Takada. Fast calculation of computer-generated-hologram on AMD HD5000 series GPU and OpenCL. *Opt. Express*, 18(10):9955–9960, May 2010.

[52] S. Sinha, J-M. Frahm, M. Pollefeys, and Y Genc. GPU-based video feature tracking and matching. Technical Report TR 06-012, Dept. of Computer Science, University of North Carolina Chapel Hill, May 2006.

[53] J. E. Slotine and Weiping Li. *Applied Nonlinear Control*. Prentice-Hall, Inc., 1991.

[54] C. Teuliere, L. Eck, and E. Marchand. Chasing a moving target from a flying UAV. In *Proceedings of International Conference on Intelligent Robots and Systems*, pages 4929–4934, Sept. 2011.

[55] P. Theodorakopoulos and S. Lacroix. A strategy for tracking a ground target with a UAV. In *Proceedings of International Conference on Intelligent Robots and Systems*, pages 1254–1259, Sept. 2008.

[56] C. Tomlin, J. Lygeros, L. Benvenuti, and S. Sastry. Output tracking for a non-minimum phase dynamic ctol aircraft model. In *Proceedings of 34th IEEE Conference in Decision and Control*, pages 1867–1872, Dec. 1995.

[57] Tom's Hardware. Charts, benchmarks desktop CPU charts 2010, raw performance. http://www.tomshardware.com/charts/desktop-cpu-charts-2010/Raw-Performance-SiSoftware-Sandra-2010-Pro-GFLOPS,2409.html, 2010.

[58] O. Tuzel, F. Porikli, and P. Meer. Region covariance: A fast descriptor for detection and classification. Technical Report TR2005-111, Mitsubishi Electric Research Laboratories, May 2006.

[59] A. Wald. Note on the consistency of the maximum likelihood estimate. *The Annals of Mathematical Statistics*, 20(4):595–601, 1949.

[60] W. Whitacre, M. Campbell, M. Wheeler, and D. Stevenson. Flight results from tracking ground targets using SeaScan UAVs with gimballing cameras. In *Proceedings of American Control Conference*, pages 377–383, July 2007.