Flying Between Obstacles with an Autonomous Knife-Edge Maneuver

by

Andrew J. Barry

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2012

Signature of Author .....
Department of Electrical Engineering and Computer Science
August 31, 2012

Certified by .....
Russ Tedrake
Associate Professor of Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Theses

Flying Between Obstacles with an Autonomous Knife-Edge Maneuver

by

Andrew J. Barry

Submitted to the Department of Electrical Engineering and Computer Science
on August 31, 2012, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

We develop an aircraft and control system that is capable of repeatedly performing a high speed (7m/s or 16 MPH) "knife-edge" maneuver through a gap that is smaller than the aircraft's wingspan. The maneuver consists of flying towards a gap, rolling to a significant angle, accurately navigating between the obstacles, and rolling back to horizontal. The speed and roll-rate required demand a control system capable of highly precise, repeatable maneuvers. We address the necessary control theory, path planning, and hardware requirements for such a maneuver, and give a proposal for a new system that may improve upon the existing techniques.

Thesis Supervisor: Russ Tedrake
Title: Associate Professor of Computer Science

Acknowledgments

I would like to thank my advisor, Russ Tedrake, for his never-ending support and encouragement to think bigger, broader, farther, and faster. His passion for robotics and flight has lead me to rekindle my love of flying and to learn more than sometimes seemed possible.

I am grateful to the students of the Robot Locomotion Group for providing one of the best environments to design, build, experiment, and learn. Special thanks to John Roberts, Ani Majumdar, Joe Moore, Jacob Steinheartt, Zach Jackowski, Mark Tobenkin, Hongkai Dai, Frank Permenter, Michael Posa, Mieke Moran, Mark Pearrow, and, of course, the wonderful Kathy Bates. I want to thank Ani in particular for many long nights spent discussing, improving, and critiquing ideas. No matter if we were in lab testing controllers or thinking hard over a (delicious) pizza from Bob at Emma's, Ani was supportive, encouraging, and always ready to entertain a new thought.

The undergraduates I have worked with, including Tim Jenks, Patricia Suriana, David Benhaim have all contributed to ensuring that this work could happen. Tim, in particular, was always there to fix the aircraft, add a new idea, and come up with some tiny, ingenious method of attaching two pieces of carbon, wood, or Delrin together. Without Tim I am sure that our aircraft would weigh twice as much and break three times as often.

Last, I must thank my family, starting with my sister Jenny, for all her support and guidance, from mathematics and robotics to life in general. I owe much to Melanie for providing loving support through sleepless nights and failure after failure. From clear to muddy water, Melanie was always there, be it standing on a mountain waterfall or making everything better by throwing mud into the sky. Finally, I owe a great deal of thanks to my parents, Dan and Sue, for their endless support, inspiration, vision, and perhaps most of all, enthusiasm. Without their tireless effort and love of life, I never would have made it.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Avian flight far exceeds our best aircraft control systems. Common birds routinely execute maneuvers well outside the bounds of our flight controllers, such as rapidly navigating through a forest, darting through extremely tight spaces, and recovering from large disturbances (Figure 1-1). Our goal is to understand how to make small aircraft achieve similar feats in equally challenging environments.

In this work, we focus on the control problem, and assume that our system is given sensing information about the world. The specific task we execute is a "knife-edge" maneuver, in which a 28-inch wingspan aircraft is launched at 7 meters per second (16 MPH) and must execute a dramatic roll to navigate through a gap that is smaller than its wingspan (Figure 1-2). This task forces our system to execute precise control in difficult flight regimes such as unsteady flow from fast roll-rates and disturbances from a high G-force launch. The aircraft flies in a motion capture setting and is connected wirelessly to off-board computers that perform the necessary computations.

Figure 1-1: Pigeon flying through an obstacle course. Image courtesy of Andrew Biewener, Concord Field Station / Harvard University [1].

1.2 Contributions

The contributions of this thesis include methods and results for building, modeling, and controlling a fixed-wing aircraft that can perform a knife-edge task including ultra-high roll-rates at high speed. The thesis is organized as follows: Chapter 2 discusses the system's hardware design and construction, Chapter 3 describes the system's planning, feedback control, and performance, Chapter 4 discusses a new method for modeling and control that improves upon these algorithms, and finally Chapter 5 offers concluding remarks.

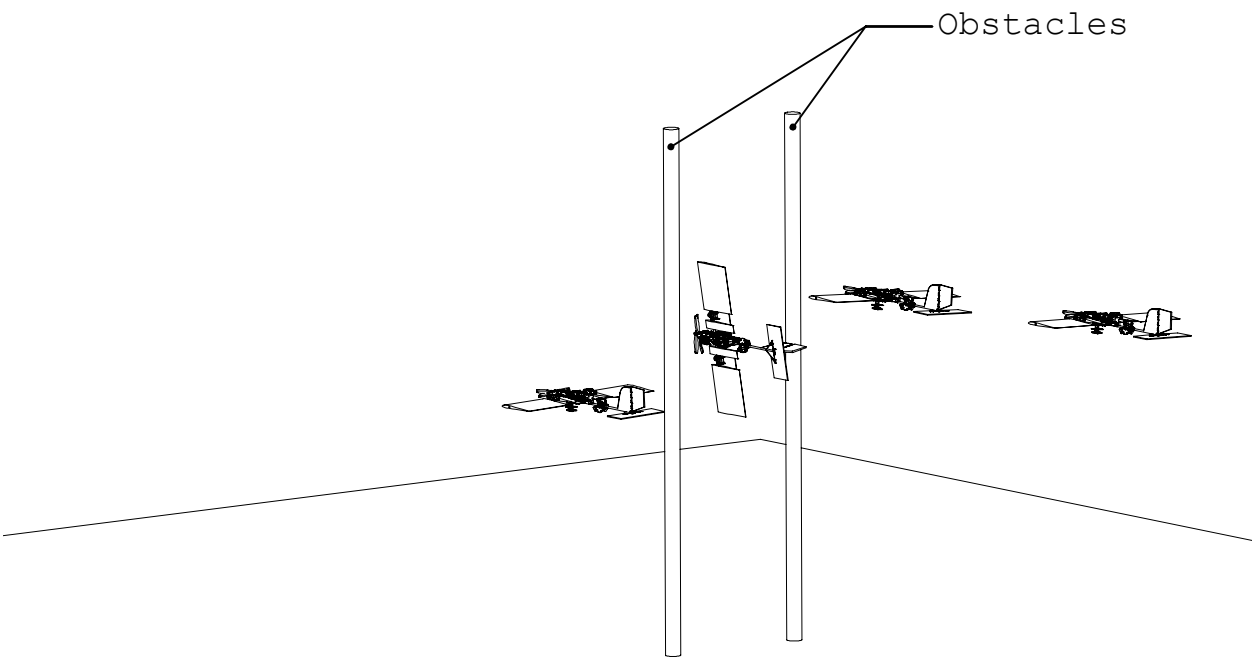


Figure 1-2: Sketch of the knife-edge task. The aircraft and obstacles' position and orientation in this figure are taken from flight data.

1.2.1 Notation

Table 1.1 introduces the notation used throughout this thesis for position, orientation, velocity, and control actions.

$\mathbf{x}$	state vector
$\mathbf{u}$	control vector
x	position on the x-axis
y	position on the y-axis
z	position on the z-axis
ψ	yaw
θ	pitch
ϕ	roll
$\dot{x}$	time derivative of x

Table 1.1: Notation

We denote the aircraft's body-centric position in 3-dimensional space and characterize

the aircraft dynamics using a 12-dimensional state as follows:

$$\mathbf{x} = \begin{bmatrix} x & y & z & \psi & \theta & \phi & \dot{x} & \dot{y} & \dot{z} & \dot{\psi} & \dot{\theta} & \dot{\phi} \end{bmatrix}^T$$

We control five inputs on the aircraft: throttle and the deflections of the elevator, rudder, left wingeron, and right wingeron. We approximate these inputs as instantaneous to avoid adding additional states to our system.

$$\mathbf{u} = \begin{bmatrix} \text{throttle} \\ \text{elevator} \\ \text{rudder} \\ \text{wingeronL} \\ \text{wingeronR} \end{bmatrix}$$

1.2.2 Collaborators

The author would like to acknowledge Tim Jenks for his significant work designing, constructing, and maintaining the aircraft. He further acknowledges both Tim Jenks and Anirudha Majumdar for their significant work on the aircraft model formulation, parameter identification, and testing procedures.

1.3 Related Work

Researchers have made substantial progress towards solving planning and control problems on systems with nonlinear dynamics and fast response times. Here we examine the most relevant works.

1.3.1 Agile Autonomous Flight

There has been substantial progress in flying robotics and control system in recent years: Abbeel et al. demonstrated a helicopter that could to perform a wide variety of aerobatic ma-

neuvers [2]. Mellinger and Kumar developed an impressive array of maneuvers for quadrotor vehicles, from flying through small gaps [3] to cooperative grasping, transport, and formation flight [4]. Cory developed a fixed-wing glider that is able to control through a stall and land on a perch [5]. Muller et al. present a system for quadrotor vehicles to juggle between themselves in flight [6]. Sobolic demonstrated a system for fixed-wing aircraft to automatically transition from hover to level-flight modes [7].

Most of these systems, however, either have simpler dynamics than a fixed-wing aircraft, or do not require the precision planning and control that a knife-edge maneuver demands. In general, quadrotor vehicles are more flexible in their control actions, since they are able to hover and to apply a torque in many more directions than an aircraft.

Flying our aircraft through a gap at speed is so difficult that we are unlikely to find a pilot able to repeatedly perform the task. Furthermore, the roll-rate of our system is fast enough that the reaction time required would demand an extremely able pilot. Abbeel's pilots did not need to consider obstacles in their maneuvers, but only attempted to remain stable, as did the resulting controllers.

Cory's fixed-wing glider used a two-dimensional model with a roll-stabilized design and thus was able to use a seven-state model, ignoring the effects of out-of-plane movement. Sobolic's system does not consider obstacles, so while he must account for the full complex dynamics of the aircraft, he does not require precision maneuvers in 3D space.

The knife-edge task is uniquely positioned relative to this previous work, requiring a more careful consideration of the system dynamics and the precision of the controllers to succeed. Here we examine methods for trajectory planning and control using both global and local models for robotic systems and discuss the unique advantages and disadvantages of each for our task.

1.3.2 Trajectory Generation and Optimization

Generating a feasible plan for a dynamical system can be a difficult task. In the case of an aircraft, trajectories must ensure that the system generates enough lift, does not saturate

the control surfaces, and is aware of the changing system models during stall. Moreover, the state space of a full 3D aircraft is large (at least 12-dimensional), so exhaustive search or even dynamic programming methods are prohibitively expensive [8]. To mitigate this issue, researchers resort to locally optimal methods that can optimize a given trajectory to satisfy dynamical constraints and minimize cost but cannot provide guarantees of global optimality. Here we discuss two methods for this task: multiple shooting and direct collocation.

In a general sense, shooting methods are optimizations that use a dynamics simulator to determine the evolution of a system and then optimize trajectories based on repeated, often short, simulations of the system's dynamics [9]. In contrast, direct collocation methods attempt to optimize trajectories by solving a nonlinear programming problem with the system's dynamics added as constraints to the optimization. This method requires the optimizer to minimize both the cost function and find a trajectory that satisfies the dynamics [10].

Probabilistic roadmaps (PRMs) attempt to solve the trajectory generation problem with a "learning" and subsequent "query" phase [11]. In the first, a planner samples the robot's configuration space, or in our case the full-dimensional state space, and attempts to build paths through that space. As it continues to sample, the algorithm attempts to connect the paths together to create "highways" that allow it to build long paths with only a small amount of new planning by attaching to previously known paths. In the query phase, these "highways" are used extensively and only small new paths to connect to the highways are required of the online planner.

Just as with the other optimizations, PRMs require a good dynamic model of the system. For aircraft however, PRMs face a more substantial issue. Aircraft system are nonholonomic, so an undirected graph of paths will not accurately represent the possible options for moving through state space. Some work on nonholonomic PRMs suggests that the algorithm can be adapted to these types of conditions, but these works often assume the ability to generate forward and backwards paths [12], which is difficult and less meaningful in our case (a loop or other dramatic maneuver would be required).

The rapidly exploring random tree (RRT) has gained significant attention for its ability to find an input tape that will take the system from one point in state space to another desired point, even in cases with challenging state space obstacles. This approach works by exploring the state space through random sampling and small extend operations [13]. We investigate using an RRT with local models in Chapter 4.

1.3.3 Feedback Control

For aircraft systems, open-loop trajectories are almost always unstable, so once given a trajectory from any of the above methods, one must chose an feedback solution to ensure that the system will stay near the given path. We examine a number of options for this feedback, including model predictive control (MPC) and time-varying linear quadratic regulators (TVLQR). We further note that in some cases, we can provide stability guarantees on these systems, under the assumption of the correctness of our models.

Model Predictive Control

At each timestep, model predictive control uses the plant model to predict trajectories out to a time horizon while solving an optimization program to find inputs that will bring the system close to the given reference trajectory. This process repeats at each timestep, giving, in the best case, optimal, results although often at high computational cost. MPC with linear systems is relatively computationally tractable, allowing for guaranteed convergence on some systems [14]. Recent work has also shown that linear MPC is capable of fast, realtime rates [15]. MPC for non-trivial nonlinear systems however, has no guarantees of convergence or even that it will produce feasible solutions at real-time rates.

Time-Varying Linear Quadratic Regulator

Recently there have been a number of successful, dynamic autonomous maneuvers based TVLQR systems [5, 2]. In this formulation, we use a standard linear quadratic regulator (LQR) controller but move the goal point at each instant in time, linearizing around the ref-

erence trajectory [16]. The algorithm is closely related to differential dynamic programming [17] with similar formulations for costs and results for linear systems.

Provably Safe Control

Despite the concentration on local linearizations and even local models (discussed in more detail below), it is still possible to generate rigorous statements about the stability and performance of some controllers using Lyapunov methods [18, 19, 20, 21]. To generate these statements, we develop a search for a Lyapunov function that, when restricted to certain classes of functions, can be cast as sums-of-squares (SOS) program. This program is in turn cast as a semidefinite optimization which makes the task computationally feasible. Drawbacks of this approach include the requirement for a good model of the dynamical system and the restriction of the model class allowed by those models. While the proofs generated are valid, actual systems may not perform as intended if the system's model is not accurate.

1.3.4 Building and Using Local Models

Identifying an accurate model for dynamical systems continues to be an area of interest. The methods we describe above use a global model to perform planning and control. Local models, however, have received significant treatment in the literature and we examine some of these as an alternative to our global model approach. Atkeson suggests using many local models to approximate a more complicated global function [22]. He first collects data about the function or system and stores all of the experiences. He then generates local models each time a query is made based on a weighted regression of past data, giving a higher weight to past experiences close to the queried point. In Chapter 4, we propose that this approach will benefit from using a richer local model built from a number of trail runs. In that way, instead of being forced to perform the regression at runtime, a system could perform a fast search for a good local model and then immediately adapt that model for control. This approach requires some study of appropriate search techniques, which we address below.

A more recent addition to the literature is the use of Gaussian Processes (GPs) for modeling and control. Deisenroth suggests that a GP formulation of the optimal control dynamic programming problem can achieve good results without discretization issues and with better scaling [23]. He demonstrates the successful control of a hardware-based cart-pole system using only 17.5 seconds of run-time on the real system as training data [24]. One drawback of this approach is the offline time required to perform the training update. In this case, the update requires $O(Tn^2D^3)$ time, where T is the prediction horizon, n is number of training samples, and D is the dimension of the state. On the cart-pole system, the offline update takes 10 minutes to complete on an average desktop computer. While the GP formulation holds substantial promise, there has been relatively little work on scaling it to systems as complicated as ours.

To perform local control (agnostic to model choice), Frazzoli's proposes a set of controllers that piece together maneuvers between *trim trajectories*, or stable regimes in state space to perform aggressive control [25]. In Chapter 4 we build on this approach, proposing similar maneuver automatons that are built with more flexible transitions in mind. We note, however, that regardless of any extension, the maneuvers concept has not been implemented on a system of this complexity.

1.3.5 Nearest Neighbor Searching

While perhaps not immediately apparent, we use results on nearest neighbor search in our proposed algorithm (Chapter 4). In the nearest neighbor problem, we attempt to find the closest point in a metric space to some query point. These algorithms have a wide variety of uses and have promising, fast solutions demonstrated in the literature.

KD-trees [26] are an obvious first approach, but we note that growth-restricted metrics allow us to perform very fast (log time) search in many more dimensions [27]. For example, the cover tree algorithm [28] provides substantial performance benefits even in the dimensionality equal to our aircraft's state space (12 dimensions). This encouraging body of literature indicates that we may be able to build a very substantial number of local models and search

them online.

Chapter 2

Aircraft Hardware

2.1 Requirements & General Design[1]

We have built an unmanned aerial vehicle (UAV) research platform to serve as the testbed for our algorithms and as the final metric of performance. The ideal platform to support our research is highly maneuverable, robust to impact, and supports state-of-the-art embedded computation. With these goals in mind, we have chosen a "wingeron" design that combines high roll rate, general maneuverability, and payload capacity with a number of impact-resistant features. The constraints of a limited motion capture volume require our aircraft to be small, having a 28-inch wingspan.

We use a "wingeron" design that does away with the traditional wing and aileron seen on most aircraft in favor of a wing that is completely actuated; in other words, the entire wing rotates to act as a control surface. This prevents the saturation of the aircraft's roll ability, when either the wing or the aileron is turned at a 90 degree angle to the surface, but complicates the flight dynamics due to the additional possibility of a turn-induced single-wing stall.

[1]The author acknowledges Tim Jenks for his insight in the design, construction, and maintenance of this system.

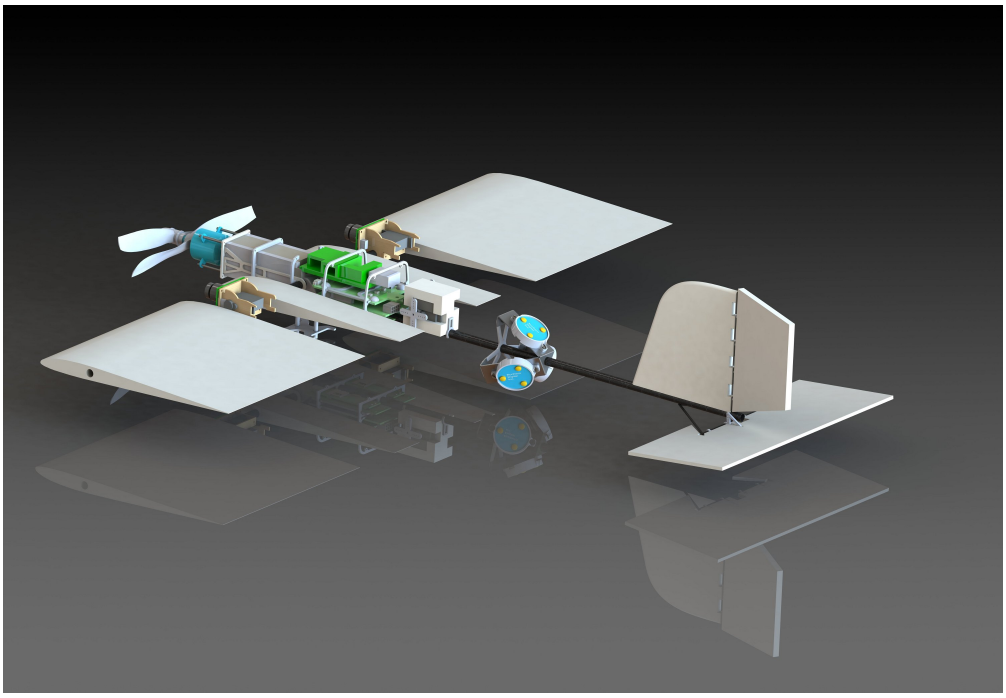


Figure 2-1: CAD rendering of the aircraft.

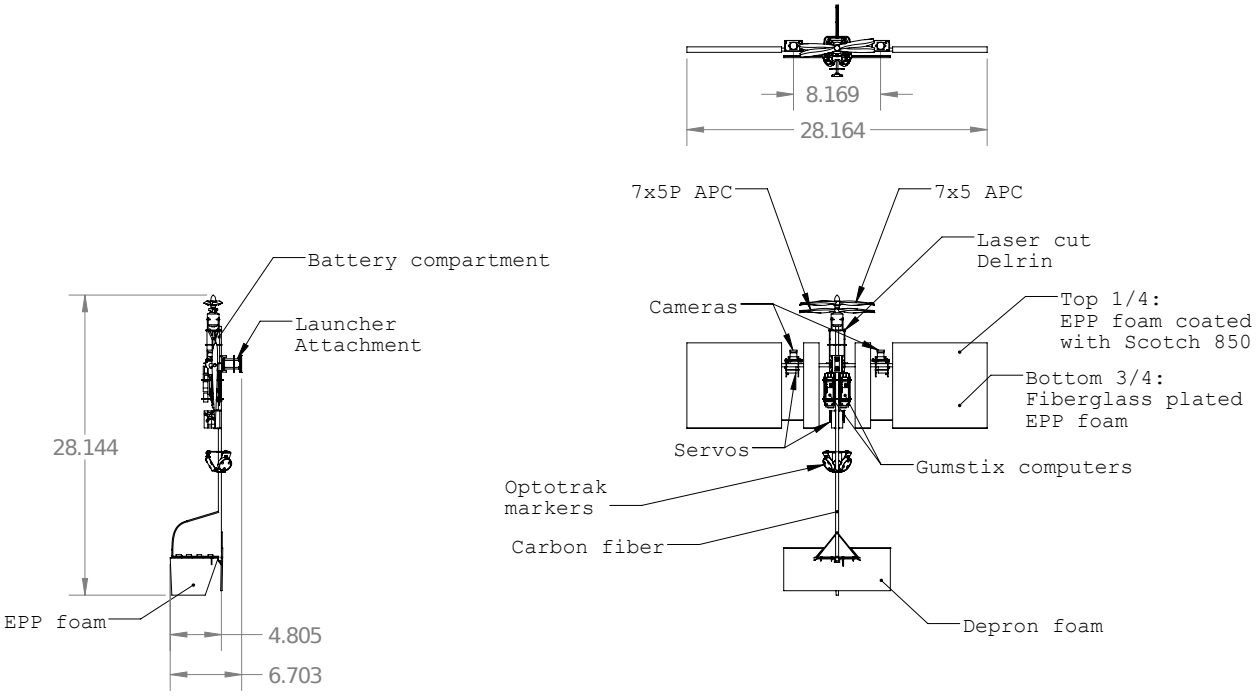


Figure 2-2: Scaled and annotated drawing of our research platform. All dimensions are in inches.

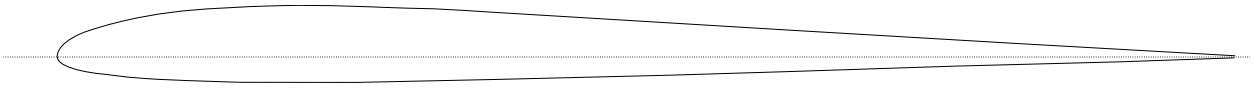


Figure 2-3: Airfoil profile, designed by Dr. Mark Drela to optimize lift at our intended flight speed of 5-15 m/s.

Figure 2-4: EPP foam wing with fiberglass applied, shown while drying in a vacuum. The vacuum technique is used to remove excess epoxy from the wing, creating a light, strong trailing edge.

2.2 Wing Design

We use an asymmetric wing that is optimized for flight at 5-15 meters per second. We cut the wing out of expanded polypropylene (EPP) foam, chosen for its ability to absorb kinetic impacts without permanent deformation. The wings are cut with a computer-controlled hot wire to produce the appropriate airfoil and then strengthened with light-weave fiberglass in the trailing three quarters to ensure that the wing does not deform when subjected to airflow.

2.3 Motor & Energy Storage

For propulsion, we use small outrunner brushless DC motors and APC propellers commonly found on this type of aircraft. We use a two-motor, counter-rotating propeller approach that

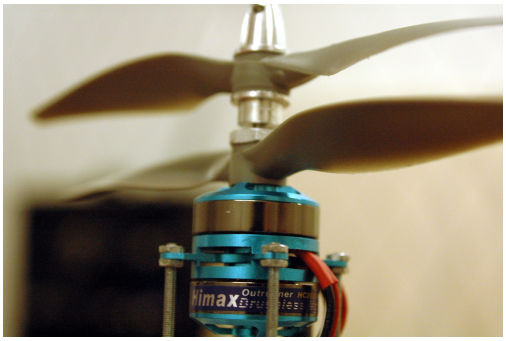


Figure 2-5: Counter-rotating propellers and brushless DC motor provides propulsion for our aircraft. Note the difference between the propeller blades which are designed to produce forward thrust when turned in different directions. We use APC 7x5 and APC 7x5P propellers.

reduces the roll effect of changing throttle speeds at a minor efficiency cost. By doing this, our dynamical models can safely ignore the torque effects of the derivative of throttle speed, reducing the model's state space by one dimension.

The aircraft is powered by a single 900mAh 3-cell (11.1V) lithium polymer battery that can provide up to 22.5A of current continuously. In practice, this battery can run the aircraft in a hovering mode for approximately 5 minutes and for much longer in a forward-flight.

The motor is mounted to the front of the aircraft with laser-cut Delrin plastic that is designed for snap-in mounting. This allows the motor mount to be completely replaced in approximately 30 minutes.

2.4 Fuselage & Electronics

The fuselage and main wing mount consists of a carbon fiber tube, connected with a semi-flexible rubber mount, custom molded to hold the rods tightly. The rubber mount holds the tubes at a 90 degree angle unless there is excessive force on the wing, in which case it flexes to prevent the rods or mount from failing.

The aircraft's internals are surrounded by four thin carbon fiber rods attached to 1/16th inch laser-cut Delrin. Each electronics board is mounted with a snap-in connector, rigidly

Figure 2-6: Rubber mount shown with the carbon fiber tubes inserted. The rubber mount holds the main spars securely but provides flexibility in the event of a large force on either the fuselage or wing.

holding the boards in place while allowing for fast servicing (Figure 2-7).

Our on-board electronics package is based around a ARM Cortex-A8 running Linux connected to a Atmel microcontroller that is in turn connected to the motor speed controllers, actuator servo motors, 3 axis gyroscope, 3-axis accelerometer, and 3-axis magnetometer.

2.5 On-board Video

We use two Point Grey Firefly cameras, one mounted on each wing, to capture stereo on-board video. The cameras use a USB 2.0 interface to deliver 320x240 grayscale images at 120 frames per second to an on-board processor (currently a second Gumstix). We use 2.1mm wide-angle lenses that provide a 150 degree field of view for spacial awareness. For the purposes of this experiment, we only used the right camera to capture on-board video. The primary use of the cameras is to collect video to understand how we might eventually determine where obstacles are using on-board video capture.

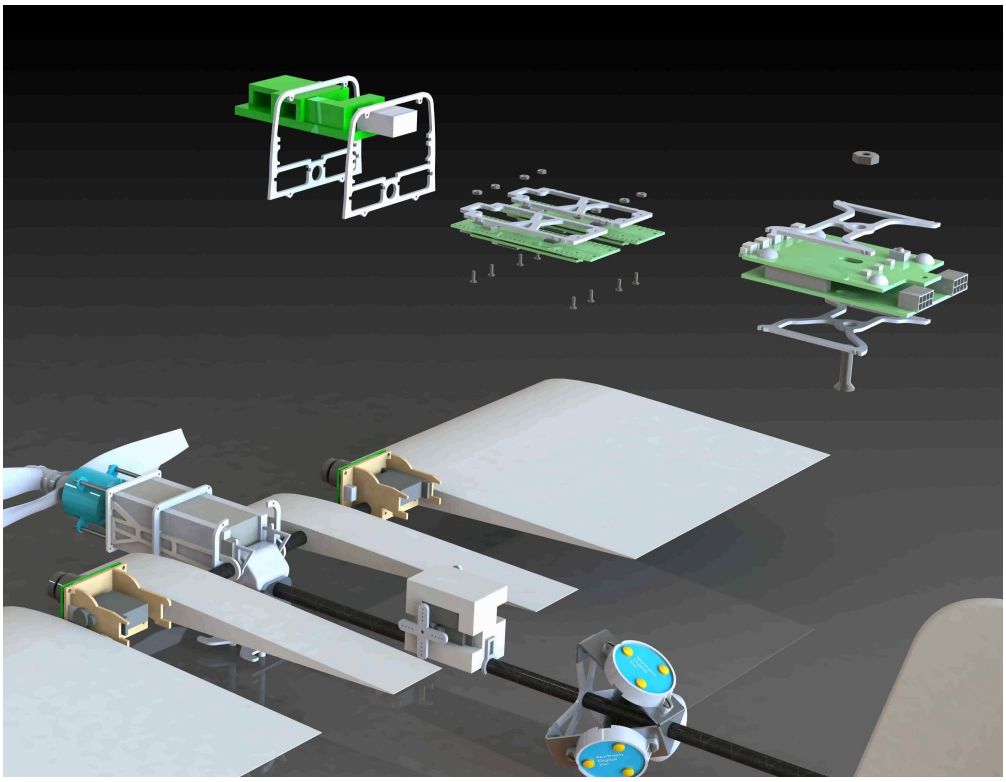


Figure 2-7: CAD rendering of the electronics mounting system.

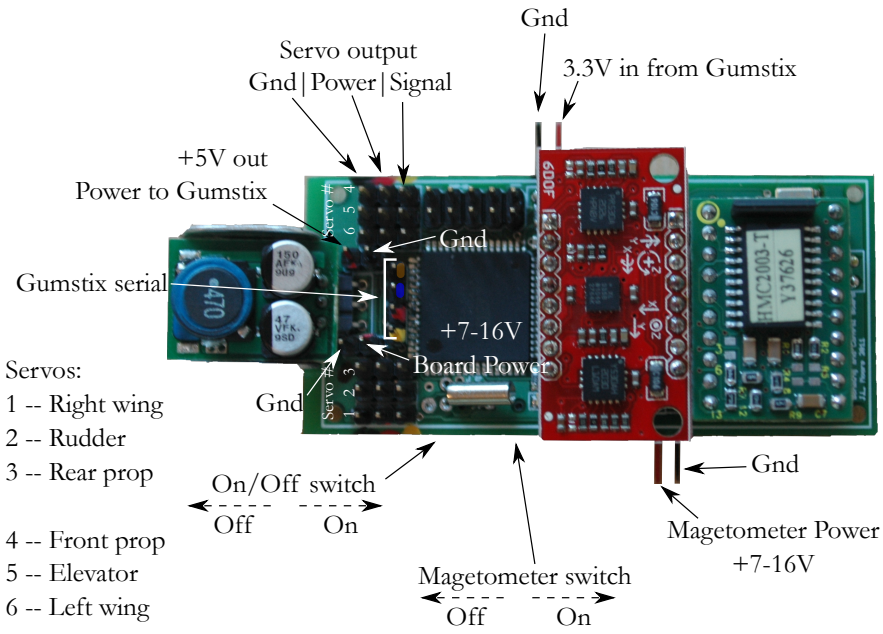


Figure 2-8: Sensor and power management board for flight hardware. The red section contains the gyroscopes and accelerometers. The Atmel microcontroller is placed center-left near the servo/motor output pins. A switched DC regulator protrudes on the left for voltage conversion from the 11.1V lithium-polymer battery.

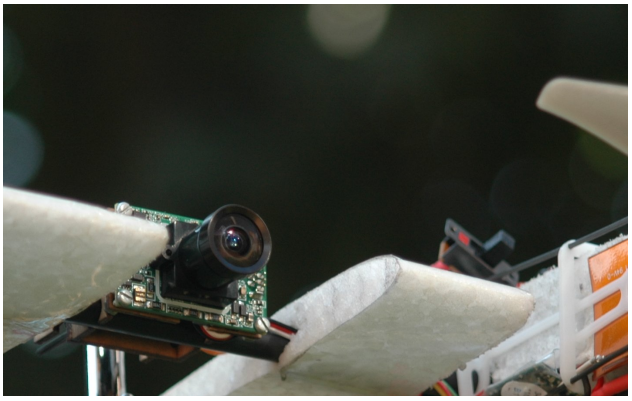


Figure 2-9: Camera mounted on the right wing of the aircraft with a wide-angle lens.

Figure 2-10: Launching apparatus. The aircraft sits on the top rail of the carriage and is held in place against its own thrust by the bar that can be seen pivoting in this photograph. The launcher accelerates the aircraft up at rates up to 9-Gs and can place the aircraft in flight at 7-8 meters per second.

2.6 Launching System

Due to the limited space in our motion-capture environment, we require a system to accelerate the plane to full speed very rapidly. We use a large piece of elastic tubing to accelerate a carriage down an approximately meter-long (3.2 ft) rail, with a 9-G peak force (Figure 2-10). A bar in front of the aircraft holds the plane in place, even at full throttle, allowing us to release the aircraft while the motors are running. The carriage is tall enough to give clearance for the propellers.

2.7 Airframe Robustness

The hardware system we have designed here has proven to have impressive performance and to be robust to multiple failures. We have flown hundreds of flights, each ending with an impact into a net, with only minor, repairable hardware failures. The airframe is robust to conditions created from sensing, control, and computational failure, as well as loss of power and unintended impact with the launching system.

Chapter 3

Control

3.1 Aircraft Model

3.1.1 Aerodynamic Model Identification[1]

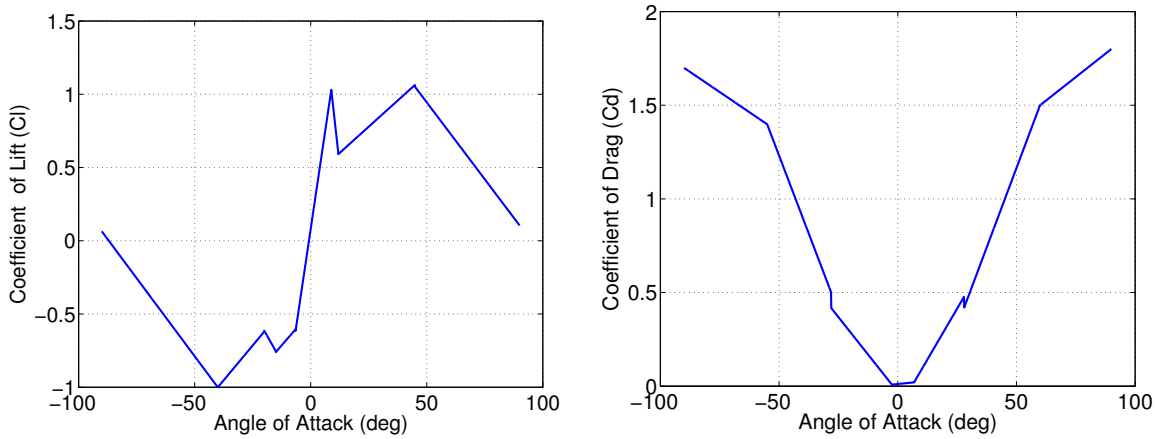
The knife-edge task requires advanced, high performance planning and control. Here, we use a model-based formulation for both, so we require a high-fidelity model of our aircraft. Stevens and Lewis [29] and Sobolic [7] provide proven aircraft models that we use as a basis. To build our model we modified a standard aircraft model to account for our wingeron design and split wings. These modifications are relatively insignificant and straightforward to do: we set the area of the wing to zero and set the aileron's area to be the full size of our wing.

Ideally, one could fit the parameters of an aerodynamic model using only the airfoil shape, the CAD model, and measurements on the airframe, but without wind tunnel testing, it is difficult to obtain a good estimate of the thrust and drag forces from the propeller and on the body respectively. To identify these parameters, we fly the aircraft repeatedly over a range of conditions and fit the following parameters: prop-wash velocity, thrust, body component of drag, and body component of drag in the vertical axis (Z). Table 3.1 shows the parameters, values, and identification methods for this model.

[1]The author would like to acknowledge Tim Jenks and Anirudha Majumdar for their significant work on this model formulation and parameter identification.

Parameter	**Value**	**Identification Method**
Mass	0.49 kg	Measured
Outboard wing area	0.04631 m^2	Measured
Inboard wing area	0.0102 m^2	Measured
Outboard wing moment arm	0.243 m	Measured
Inboard wing moment arm	0.055 m	Measured
Elevator area	0.0217 m^2	Measured
Elevator moment arm	0.49 m	Measured
Vertical stabilizer area	0.0092 m^2	Measured
Vertical stabilizer moment arm	0.40 m	Measured
Rudder area	0.00850 m^2	Measured
Rudder moment arm	0.49 m	Measured
Moment of inertia (X)	0.00515 $kg \cdot m^2$	CAD
Moment of inertia (Y)	0.00881 $kg \cdot m^2$	CAD
Moment of inertia (Z)	0.00390 $kg \cdot m^2$	CAD
Prop-wash velocity	0.025 * throttle	Fit
Body component of drag ($c_d \cdot A$)	0.0048	Fit
Body component of drag (Z) ($c_d \cdot A$)	2.424	Fit
Coefficient of lift	Figure 3-1(a)	XFOIL simulation
Coefficient of drag	Figure 3-1(b)	XFOIL simulation

Table 3.1: Model parameters. Parameters are either measured directly on the airframe, computed using a Solid Works CAD model, computed using an airfoil simulation, or fit from flight data. Note that the body component of the drag is much higher in the Z direction, since that direction is relative to the aircraft's coordinate frame. As expected, there is much more drag to move down than there is to move forward. The "throttle" value is the throttle input on a range of 0-255.



(a) Coefficient of lift

(b) Coefficient of drag

Figure 3-1: Coefficients of lift and drag used in the model at varying angles of attack. At low angles of attack, the curves are fit using piecewise polynomials to data from airfoil simulations using the XFOIL [30] package. At high angles of attack, we use flat plate theory as suggested by [31, 5]. The plots look especially sharp because we plot them for a full 180 degrees and do not smooth the transitions between computation methods.

3.2 Open-Loop Planning

3.2.1 Planning with Direct Collocation

We use a direct collocation method for our open-loop trajectory generation system. We do not require modifications to the standard direct collocation method as given by Hargraves and Paris [32], so we will only describe it briefly here. The method uses cubic polynomials to represent the system's trajectory and uses collocation to satisfy the dynamics. Here we examine our customized cost and constraint functions.

Our cost function focuses on the control actions of the aircraft. The one-step cost is $g = \mathbf{u}^T R \mathbf{u}$, where $\mathbf{u}$ is the control vector introduced earlier. We choose R initially based on

intuition and finally hand-tuning during experiments:

$$R = \begin{bmatrix} 0.001 & 0 & 0 & 0 & 0 \\ 0 & 0.003 & 0 & 0 & 0 \\ 0 & 0 & 0.005 & 0 & 0 \\ 0 & 0 & 0 & 0.0001 & 0 \\ 0 & 0 & 0 & 0 & 0.0001 \end{bmatrix}$$

We let the system's final cost simply equal to the final time, promoting faster solutions: $h = t_f$.

To ensure that our trajectory does not collide with obstacles, we add constraints based on the distance to each obstacle. With the cost function and constraints, we simultaneously balance avoiding obstacles and limiting control actions. We use a polygonal obstacle representation with infinitely tall obstacles, allowing us to safely compute the distance to the obstacle by projecting the aircraft's image and the obstacles' image onto the XY plane. We then compute the minimum distance from the aircraft's "shadow" to the obstacles' polygon. This method allows us to easily incorporate the aircraft's roll configuration into the computation, since we can modify the craft's projection accordingly.

If we let d be the minimum distance from the perimeter of the obstacle's polygon to the obstacle, we use the following function for the constraint's cost:

$$\phi = \tanh(-d) \tag{3.1}$$

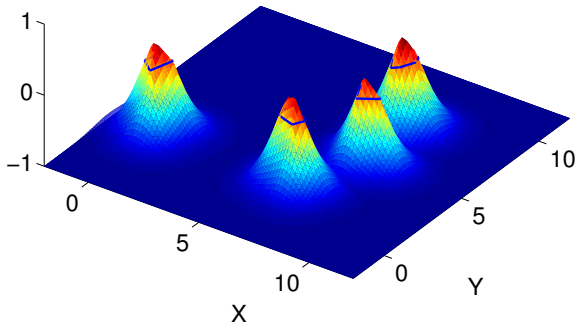


Figure 3-2: Constraint function value for a 2D slice of space (roll is set to zero). Values greater than one are inside the obstacles (shown as blue polygons). Values less than one are outside the obstacles.

and the derivatives of equation 3.1 are:

$$\frac{d\phi}{d\mathbf{x}} = \begin{bmatrix} \frac{d\phi}{dx} \\ \frac{d\phi}{dy} \\ \frac{d\phi}{dz} \\ \frac{d\phi}{d\psi} \\ \frac{d\phi}{d\theta} \\ \frac{d\phi}{d\phi} \\ \frac{d\phi}{d\dot{x}} \\ \frac{d\phi}{d\dot{y}} \\ \frac{d\phi}{d\dot{z}} \\ \frac{d\phi}{d\dot{\psi}} \\ \frac{d\phi}{d\dot{\theta}} \\ \frac{d\phi}{d\dot{\phi}} \end{bmatrix}^T = \begin{bmatrix} \frac{d\phi}{dx} \\ \frac{d\phi}{dy} \\ \frac{d\phi}{dz} \\ \frac{d\phi}{d\psi} \\ \frac{d\phi}{d\theta} \\ \frac{d\phi}{d\phi} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}^T \tag{3.2}$$

which we compute using numerical differentiation. Figure 3-2 visualizes the constraint function with a slice in two dimensions.

To generate trajectories, we being with a tape that starts at x_0 and flies straight to the goal. We then minimize the cost over over the control action while ensuring that the dynamics are satisfied and that we do not impact an obstacle:

$$\begin{aligned} \min_{\substack{u_0 \ldots u_n \\ x_0 \ldots x_n}} \quad & h + \sum_{n=1}^{N} g(x[n], u[n]), \end{aligned}$$

subject to:

$$\begin{aligned} x[0] &= x_0, \\ x[n+1] &= f(x[n], u[n]) \\ \phi_i(x) &< 0 \end{aligned}$$

where ϕ_i is included for each obstacle.

3.3 Feedback Control with a Time-Varying Linear Quadratic Regulator

We stabilize the open-loop trajectories using a time-varying linear quadratic regulator (TVLQR). We do not require modifications to the standard TVLQR algorithm, so we refer the reader to [33, 5] for its derivation.

The TVLQR controller requires some hand-tuning of the cost matrix before it will track the open-loop trajectory adequately. We performed this hand-tuning based on repeated flights and comparisons between the system's actual and desired paths. The final costs for

the knife-edge trajectory are as follows:

$$Q = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 42 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix}$$

$$R = \begin{bmatrix} 0.2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0.6 & 0 \\ 0 & 0 & 0 & 0 & 0.6 \end{bmatrix}$$

3.3.1 Experimental Setup

We placed two poles 0.7 meters (27.5 inches) apart 2.75 meters (9 feet) in front of our launching system. Figure 3-3 gives an overview of the experimental setup.

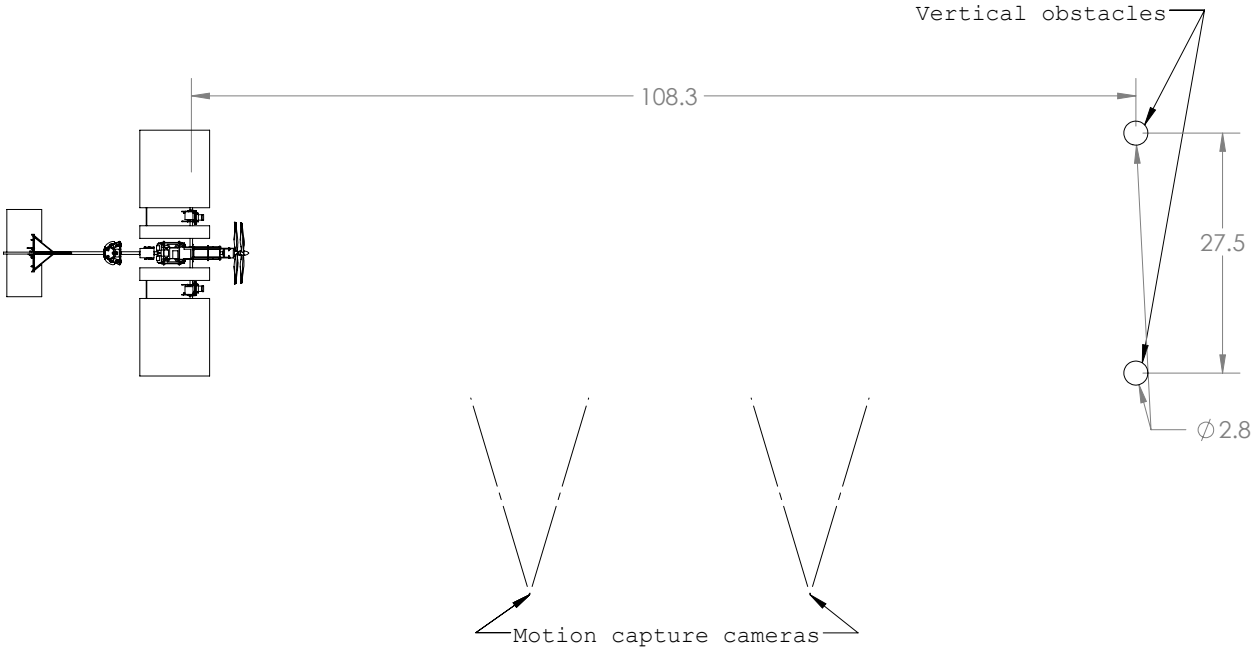


Figure 3-3: Scaled diagram of the experimental setup. All dimensions are in inches.

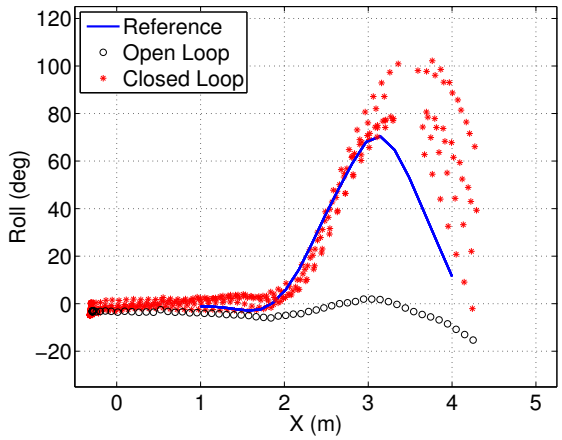
3.3.2 Knife-Edge Results

Tracking Performance

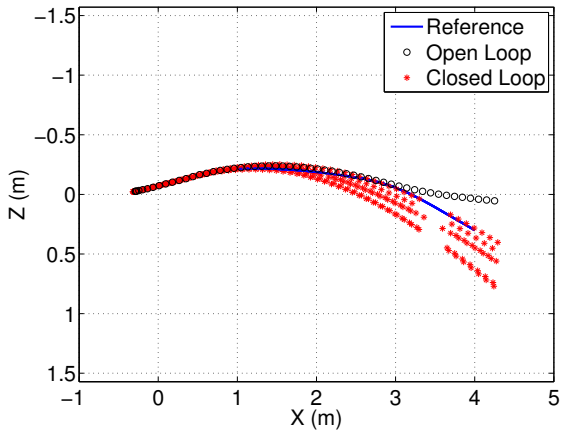
Our open loop trajectory requires the aircraft to perform a 70 degree roll in under two body-lengths, while moving at over 10 body-lengths per second. We require this maneuver to be performed with extreme precision, demanding an advanced control system with tight tracking. Here we present the results of the system described above with tuned LQR costs. The tracking is sufficient for repeated flights through our obstacle field without collision. Figure 3-4 shows the results six repeated flights with this system, the desired trajectory, and the result of running the system without feedback. Figure 3-9 gives snapshots of the aircraft performing this trajectory in our test environment.

Tuning

Our system requires hand-tuning of LQR gains. We found that each trajectory requires approximately 40-60 flights to tune, although we note that this is a rough measurement



(a) Aircraft flight data for roll vs. forward flight. Red shows flights with the closed-loop controller running and black shows flights running only the open-loop tape.



(b) Comparison of forward flight (X-axis) vs. height (Z-axis). Note that the Z axis is plotted in reverse to give a more intuitive view of the aircraft's height (in our coordinate system increasing Z moves towards the ground).

Figure 3-4: Motion capture data from the aircraft in flight compared with the planned knife-edge maneuver. We show both open loop (black) and closed loop (red) flights. The gaps in the data occur when the plane's markers pass by the obstacles, causing the motion capture system to lose sight of the plane for a small period of time.

based only on a few scenarios. The hand-tuning required is relatively straight forward and an intuitive knowledge of the trade-offs that each cost makes is enough to provide a good idea of what modifications are required. The main focus of our tuning was the trade-off between cost on action (ensuring that the aircraft would not over-compensate) and cost on errors in roll (ensuring that the plane would roll enough).

There are two primary considerations that allowed us to perform this tuning rapidly. The first is that our aircraft is robust to impacting a net repeatedly without damage. This allowed us to remove the obstacles, and use the motion-capture system to inform us of the system's performance. Secondly, our launching system allowed us to achieve relatively consistent initial conditions which significantly reduced the tuning challenge.

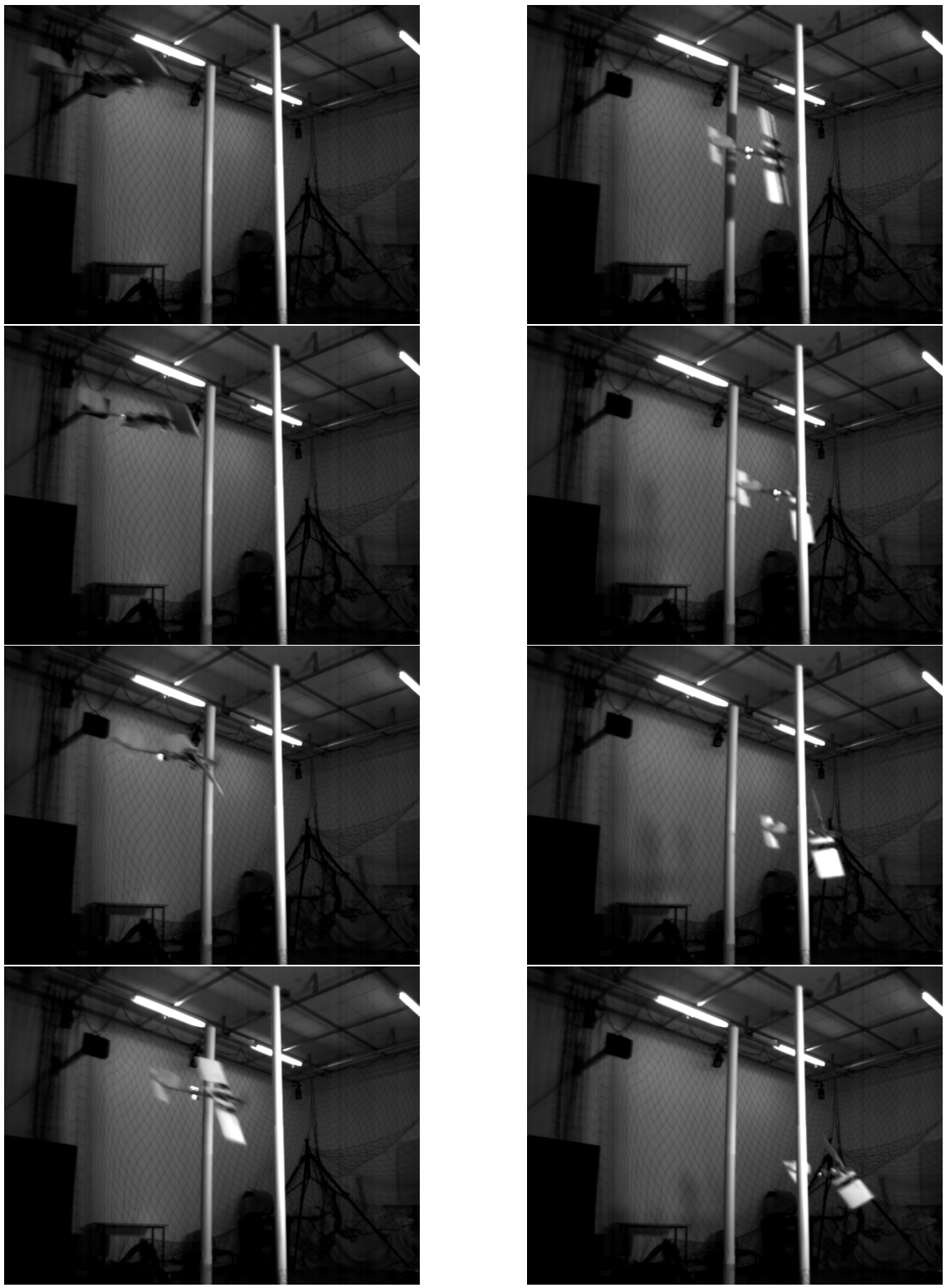
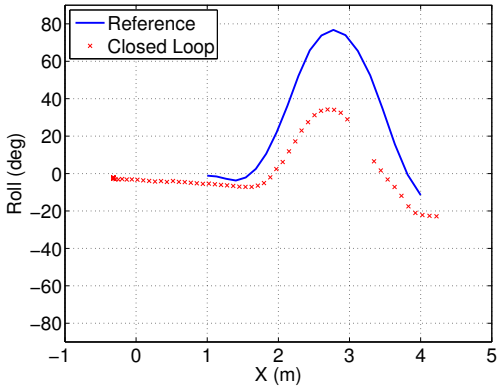
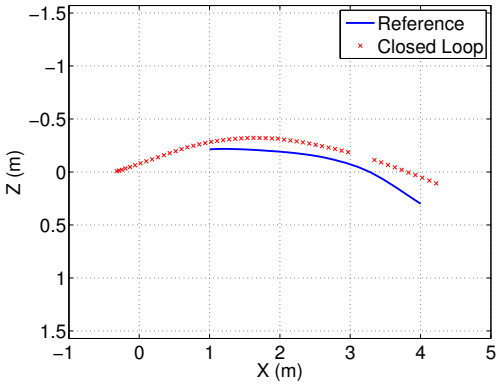


Figure 3-5: Sequence of stills from a knife-edge maneuver. Each image is 0.0417 seconds (41.7ms) after the previous. The entire set captures a total of 0.292 seconds.

Figure 3-6: Sequence of stills from an onboard camera during the knife-edge maneuver. Each image is 0.083 seconds (83ms) after the previous. The entire set captures a total of 0.583 seconds.



(a) Aircraft flight data for roll vs. forward flight during the four-obstacle maneuver.



(b) Comparison of forward flight (X-axis) vs. height (Z-axis). Note that the Z axis is plotted in reverse to give a more intuitive view of the aircraft's height.

Figure 3-7: Motion capture data from the aircraft in flight compared with the planned four-obstacle knife-edge maneuver. The gap in the data occurs when the plane's markers pass by the vertical obstacles (horizontal obstacles do not cause an occlusion).

Extension to Horizontal Obstacles

While the knife-edge maneuver demonstrates exceptional tracking and system performance, it does not require the system to perform more than one maneuver during a flight, as is demonstrated by the worse tracking on the trailing edges of Figure 3-4. Here we show that the same system can be used to perform multiple maneuvers by adding two horizontal obstacles after the two vertical obstacles. This requires the aircraft to quickly roll back to level flight and maintain altitude tracking to succeed in avoiding both obstacles.

We used our existing model to plan a new trajectory with horizontal obstacles, described in the system in a very similar way to our vertical barriers. We then tuned the LQR costs for that trajectory to improve tracking. Figure 3-7 shows the trajectory of a successful flight and Figure 3-8 demonstrates snapshots of the flight.

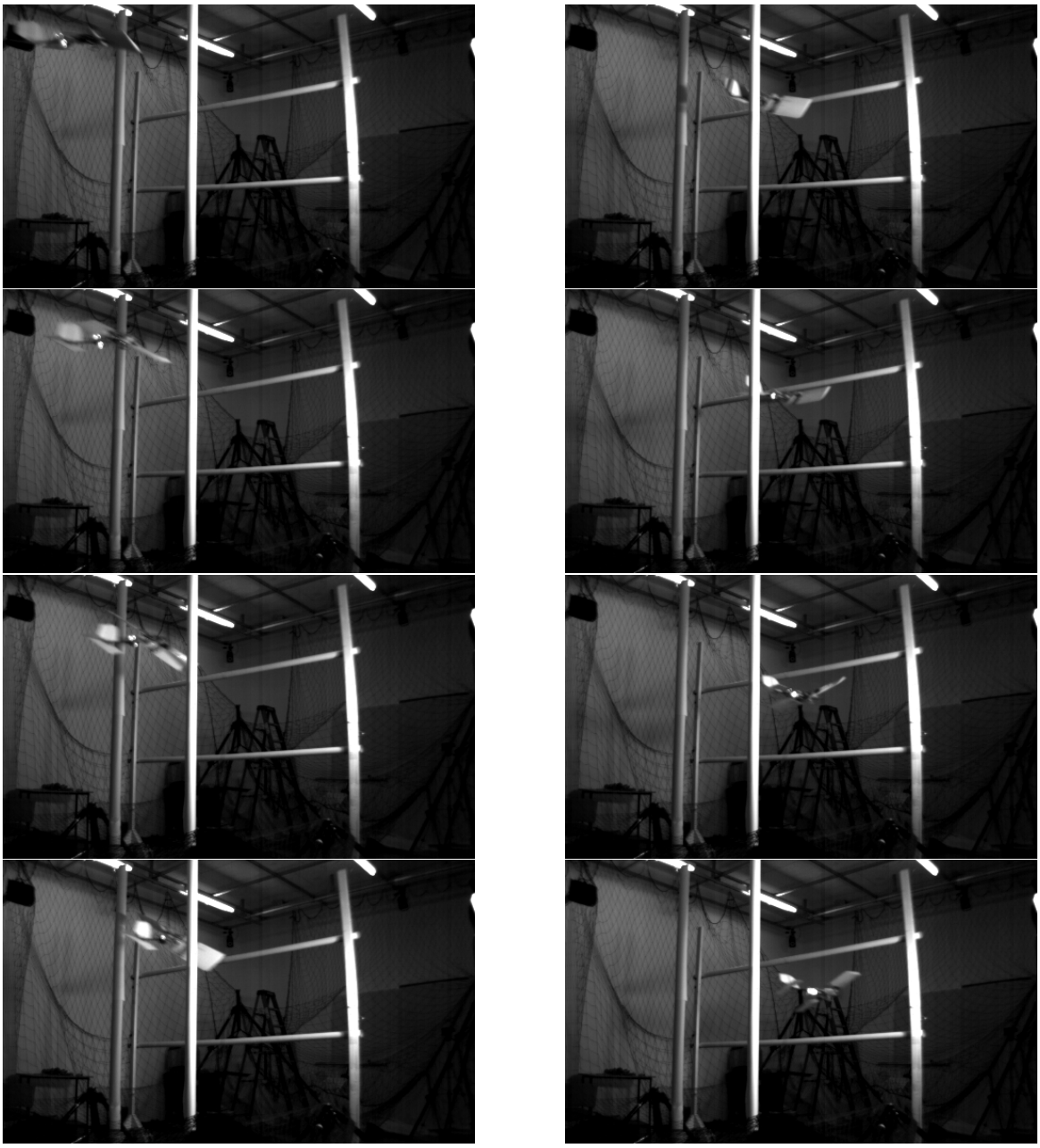


Figure 3-8: Sequence of stills from a four-obstacle maneuver. As before, each image is 0.0417 seconds (41.7ms) after the previous. The entire set captures a total of 0.292 seconds.

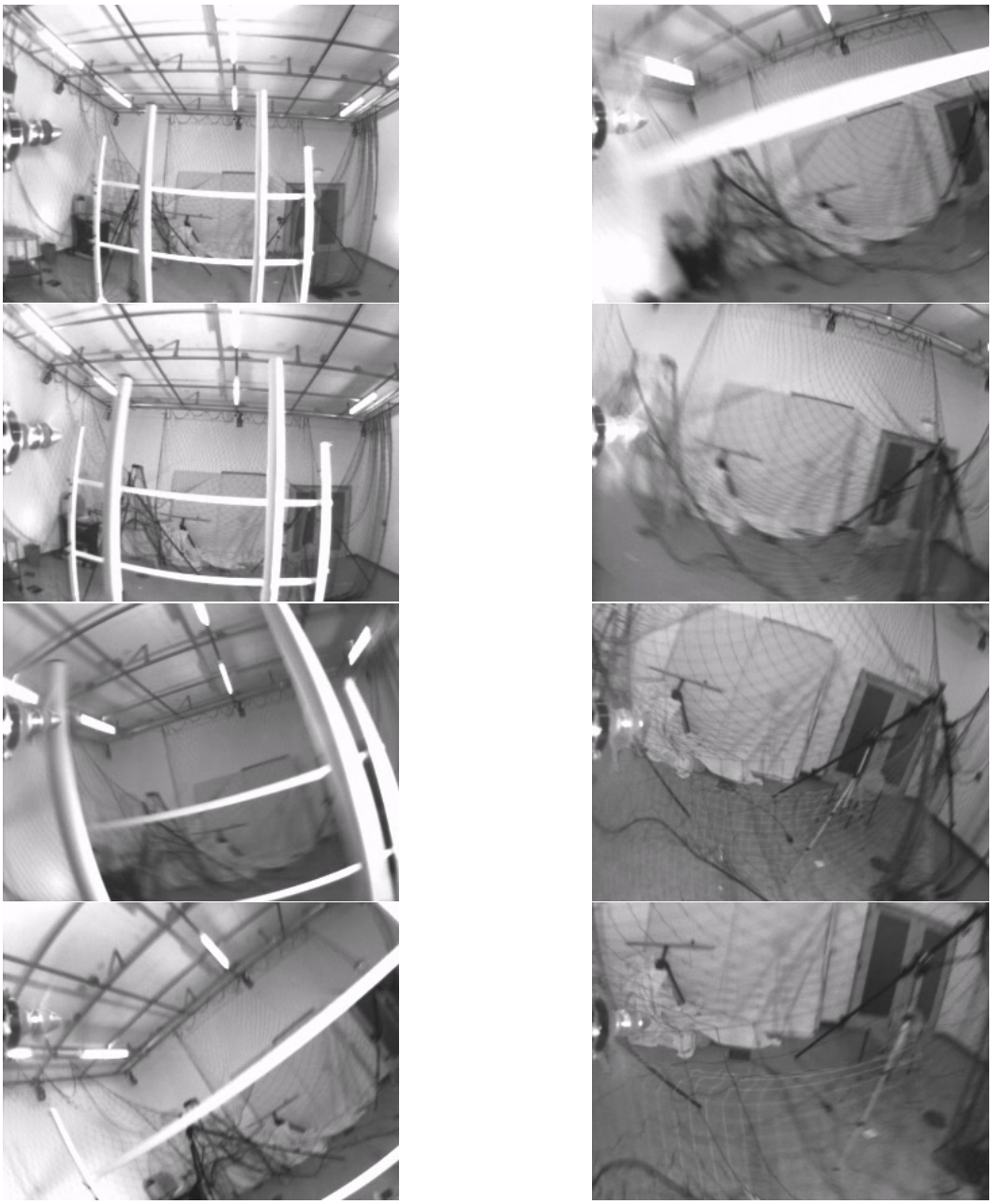


Figure 3-9: Sequence of stills from an onboard camera during the four-obstacle knife-edge maneuver. Each image is 0.083 seconds (83ms) after the previous. The entire set captures a total of 0.583 seconds.

3.4 Discussion

3.4.1 Aircraft Model

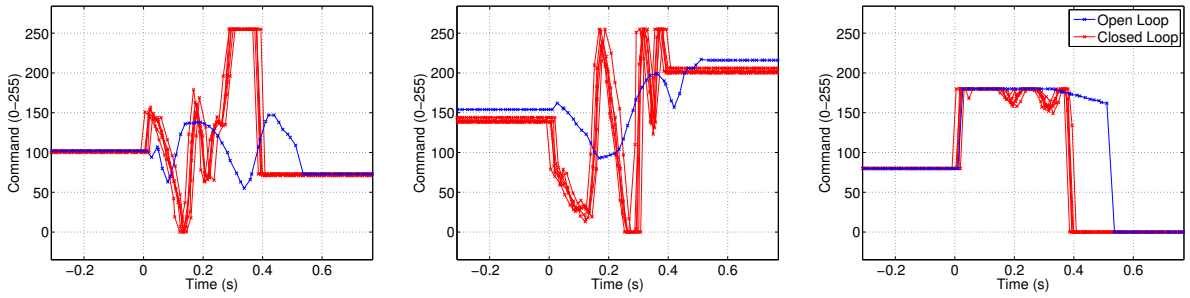
While our technique for using a standard aerodynamic model of our aircraft works well enough to succeed in the knife-edge maneuver, it is unsatisfying that building such a model takes substantial effort and data to refine the model's accuracy to an acceptable level. Should we want to extend these control techniques to more general systems and systems outside of laboratory environments, better modeling tools are required. These tools must require less "hand-holding" and repeated refinement to achieve the ability to generate reasonable action-tapes with some accuracy.

Perhaps more unsettling is that the nominal action-tapes used for the knife-edge maneuver do not necessarily reflect the actual control actions the aircraft takes over repeated flights (Figure 3-10). This, to some extent, explains why the open-loop trajectories, like that in Figure 3-4 perform so poorly. The discrepancy indicates that our feedback system is primarily accounting for failures in our model, not disturbances during flight. We expect that if we can fly using better initial tapes, our performance will increase as the feedback system's task becomes easier.

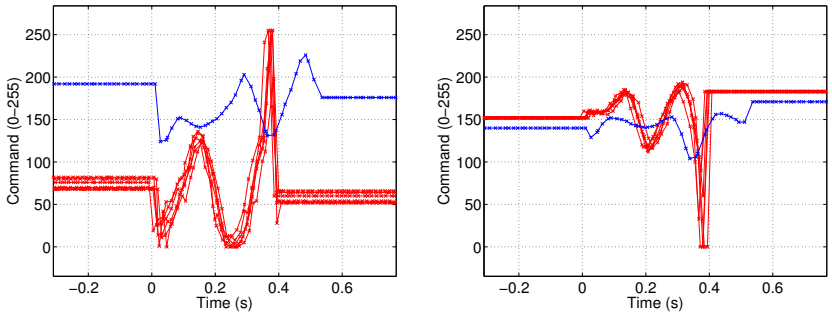
3.4.2 Using TVLQR

To our knowledge, time-varying linear quadratic regulators have not been applied to dynamical systems that require as much precision in flight control as in our task. It is important to note that we have found this feedback system to work repeatedly, albeit with hand-tuning. While the performance of the feedback system is impressive, it is disappointing how sensitive it is to the costs.

To succeed at the knife-edge task we required a system that could be flown repeatedly, with similar initial conditions, so that we could hand-tune the TVLQR cost matrices. This means that systems that do not have repeatable initial conditions or cannot withstand significant use outside their nominal operating regime are beyond our ability to control with



(a) Left wingeron (b) Right wingeron (c) Throttle



(d) Elevator (e) Rudder

Figure 3-10: Comparison between open-loop control tapes (blue) and closed-loop tapes (red) for the knife-edge trajectory. Note the similarity between six successful runs of the knife-edge trajectory and the striking dissimilarity between those and the open-loop tape. This indicates that while our model is good enough to perform the task, our feedback system is primarily correcting for its inaccuracies, not in-flight disturbances.

these techniques.

Both the deficiencies in our model-building and the required tuning of our control system point to an algorithm that we think can improve control systems. It is disappointing that despite flying an aircraft through a reasonable trajectory, a model will, without careful tuning, produce widely incorrect answers. A more data-driven approach in which the dynamics are characterized directly from data, can potentially alleviate this issue. Furthermore, a similar argument can be made for a control system that uses past trajectories to develop future actions of the system. We discuss the beginnings of one technique that attempts to solve these issues below.

Chapter 4

A Proposal for Fast and Cheap Modeling and Control Design

Building globally accurate models for systems of this complexity is a challenging task. The 12-dimensional state space of our system is so large that exhaustive experimentation is nigh impossible. The dynamics are often difficult to model from first principles and change rapidly in regimes such as stalled and post-stalled flight.

We propose a data-driven local model approach for designing an aircraft control system. We will record a significant number of trajectories of the aircraft with a variety of control inputs at our desired speed. We will focus these inputs on the maneuvers we are interested in performing (such as rapid roll and roll-out). With these data we will build a large number of small local models around the regions in state space explored by these trajectories. We will place each local model in a cover tree [28] so that we can perform nearest neighbor search very rapidly.

Importantly, we assume that the system's dynamics are repeatable and that we can rely on past experience to give us a robust indication of the future dynamics. While the true system has far more states then we account for (dynamics history, air flow along the wings, incoming gusts, propeller imbalance, etc.) we hypothesize that by connecting local models in appropriate ways, we can minimize the effect of these states, most notably the dynamics

history, and provide a controller that is both highly dynamic and robust. Our knife-edge work has provided evidence for this assumption, notably that the action-tapes between different successful flights are similar (Figure 3-10).

To plan a trajectory with this data structure, we propose to use a reversed rapidly exploring random tree (RRT) [13] in which the extend operation is implemented by querying nearby local models for valid points and entirely rejects samples outside the regions near good data (or equivalently does not sample from those regions). In this way, our system will not plan a path into a region where the dynamics are not well understood, but instead will ensure that the system is confident in its characterization of the dynamics before venturing into that region of state space.

While online, the system will have two levels of feedback control. At the lower level, the local model is used to control to a desired trajectory. Our models will be small, however, and disturbances may quickly push the system outside of the region in which its current local model is valid. When that occurs, the system will perform a logarithmic-time lookup of nearby local models using the cover tree data structure to quickly identify a different model of the dynamics for which to use for control. By ensuring that we have many small local models, the system will be flexible to both small and large disturbances.

We note that this system will be limited in its ability to operate in large regions of state space since it requires good data before it will plan to enter new regions. We hypothesize that this limitation will not severely constrain our operation, even for highly dynamics maneuvers, since while the regions of state space we are interested in are dynamic, they still represent a small region of the entire space.

4.1 Building Models from Data

While it would be ideal to have a proven, flown path through every point in state space, generating that dataset from experiments is prohibitively expensive. In fact, exhaustive trajectories in just a tiny region of state space are still prohibitively expensive to collect, since each path is measure zero. Thus, to fill the space, we resort to building models around

data, that can capture how the dynamics change in a local region.

Optimally determining where to split our trajectories (where one model ends and another begins) is a difficult task beyond the scope of this thesis, so we offer a simple constant-time metric to stand in its place: split models after t_c time. We hypothesize that this metric will have sufficient performance to allow us to use the models in many interesting and dynamic cases.

We further note that using all available data to determine the linear regression for each model is likely to be less useful than using only nearby data, so we propose a weighted regression approach, although we leave its implementation for future work.

4.1.1 Planning with Many Models

As described above, we use a trivially modified reverse-RRT to plan actions. We seed the RRT with every trajectory that we have collected data for, generating a forest with varying initial conditions. While we could plan to connect each of these points, we take the simplifying assumption that we will be able to start at any of the given initial conditions.

For each RRT extend operation, we find the closest model to our query point and use those dynamics to extend a single dt towards the query point. We add the result of each extend operation to a second cover tree, allowing for fast lookups of all paths and all models. We repeat the query-extend loop until the distance to any known trajectory is less than a threshold.

4.1.2 Preliminary Results

We implement this system on a simulated, torque-limited pendulum, and attempt to generate an action-tape for the swing-up to a different point. We do not implement a weighted regression and use a Euclidean distance metric for the RRT. Figure 4-1 shows the initial plan in dark black.

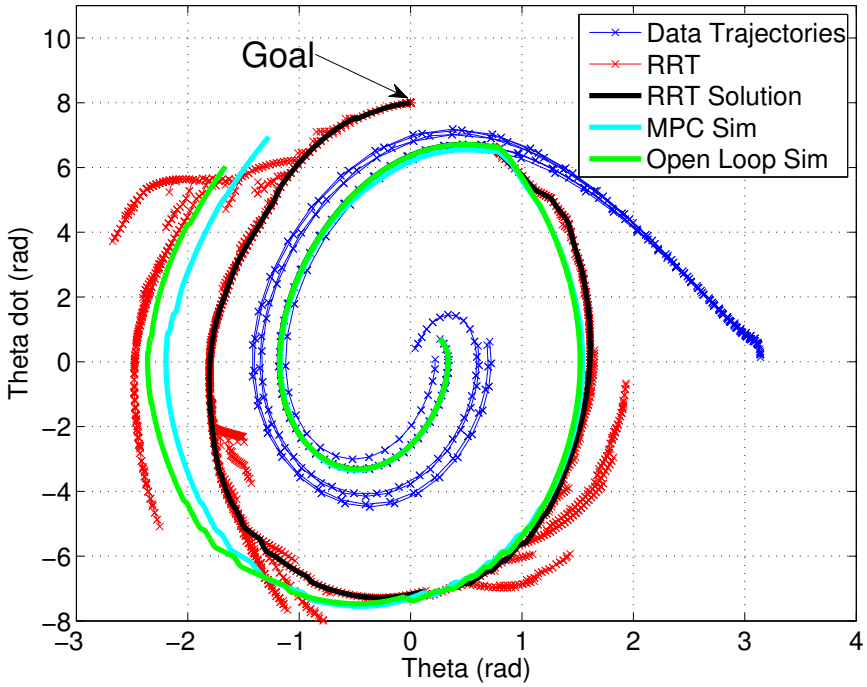


Figure 4-1: Trajectories in the state space of the torque-limited pendulum. The task is to move the pendulum to the point $(0, 8)$ when given the data from 8 full swing-up trajectories. The data are show in blue, marked by xs at each break in a tiny model. The RRT's plan is show in red, marked at each dt. A converging trajectory is show in dark black, followed by a simulation of that trajectory using MPC feedback (cyan) and an open-loop simulation (green). Note that we expect the system's performance would improve if we simulated beyond the end of the action-tape's time.

4.2 Control Implementation

We propose to use a model predictive control (MPC) framework for control using these many local models. In this framework we will be given a reference trajectory. At each timestep, we search for the closest model to our current location in state space and use that model to perform MPC. To ensure that we are using a relevant model throughout the MPC horizon, we simulate the system forward using the nominal action-tape and note the best model at each dt. Figure 4-1 shows a comparison between the MPC feedback (cyan) and an open-loop simulation (green).

Chapter 5

Conclusions and Future Work

5.1 Future Work

5.1.1 Scaling Local Model Framework

Our preliminary implementation of the local model framework described in Chapter 4 has not yet been expanded to complicated dynamics or real data. In addition to the nessasary implementation details for those systems, we intend to add support for symetrical dyanmics, states that are dynamics-invariant (such as the position states for the aircraft), and other methods for trajectory generation. It is an open question if an RRT or a direct collocation method will perform better for that framework.

5.1.2 FPGA Vision and Outdoor Navigation

Recent work on FPGA vision systems has provided encouraging results that our platform may be able to collect and process information about its surroundings at 120 frames per second [34]. We are considering moving our platform out of a motion capture environment, adding an FPGA processor, and determining where obstacles are in real time.

Moving out of a motion capture environment will require a substantial sensing and estimation effort, which will extend our sensors to airspeed, barometric altitude, GPS, and

others. Our aircraft is capable of carrying these sensors and we look forward to UAVs that can fly at high speed, identifying and avoiding obstacles faster than any human pilot.

5.2 Conclusions

We have demonstrated a system that is capable of performing a knife-edge maneuver, rolling 70 degrees in under two body-lengths, while moving at over 10 body-lengths per second. Our system, using an aerodynamic model, direct collocation based trajectory optimization, and TVLQR is capable of performing the maneuver robustly. We have demonstrated that the system is able to extend to other trajectories and obstacle configurations and have given a proposal for a modeling and control system that is better able to address some of the practical and theoretical concerns with the state of the art system.

Bibliography

[1] HT. Lin, I. Ros, and A. Biewener. Through the eyes of a bird: A vision-based model for pigeon obstacle avoidance flights. In preparation.

[2] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Proceedings of the Neural Information Processing Systems (NIPS '07)*, volume 19, December 2006.

[3] D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. In *Proceedings of the 12th International Symposium on Experimental Robotics (ISER 2010)*, 2010.

[4] Mellinger, D., Shomin, M., Michael, N., Kumar, and V. Cooperative grasping and transport using multiple quadrotors. In *Proceedings of the international symposium on distributed autonomous robotic systems*, 2010.

[5] Rick Cory. *Supermaneuverable Perching*. PhD thesis, Massachusetts Institute of Technology, June 2010.

[6] Muller, M., Lupashin, S., D'Andrea, and R. Quadrocopter ball juggling. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 5113–5120. IEEE, 2011.

[7] F. M. Sobolic. Agile flight control techniques for a fixed-wing aircraft. Master's thesis, Massachusetts Institute of Technology, June 2009.

[8] M. Diehl, H.G. Bock, H. Diedam, and P.B. Wieber. Fast direct multiple shooting algorithms for optimal robot control. *Fast Motions in Biomechanics and Robotics*, pages 65–93, 2006.

[9] Bock, H.G., Plitt, and K.J. A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings 9th IFAC World Congress Budapest*, pages 243–247. Pergamon Press, 1984.

[10] Oskar von Stryk. Numerical solution of optimal control problems by direct collocation. In *Optimal Control, (International Series in Numerical Mathematics 111)*, pages 129–143, 1993.

[11] L.E. Kavraki, P. Svestka, JC Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, August 1996.

[12] P. Švestka and M.H. Overmars. Motion planning for carlike robots using a probabilistic learning approach. *The International Journal of Robotics Research*, 16(2):119–143, 1997.

[13] S. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98–11, Iowa State University, Dept. of Computer Science, 1998.

[14] Alberto Bemporad and Manfred Morari. Robust model predictive control: A survey. In A. Garulli and A. Tesi, editors, *Robustness in identification and control*, volume 245 of *Lecture Notes in Control and Information Sciences*, pages 207–226. Springer Berlin / Heidelberg, 1999. 10.1007/BFb0109870.

[15] Y. Wang and S. Boyd. Fast model predictive control using online optimization. *Control Systems Technology, IEEE Transactions on*, 18(2):267–278, 2010.

[16] Philipp Reist and Russ Tedrake. Simulation-based LQR-trees with input and state constraints. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2010.

[17] B.D. Mayne. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control*, 3(1):85–95, 1966.

[18] Pablo A. Parrilo. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. PhD thesis, California Institute of Technology, May 18 2000.

[19] Zachary Jarvis-Wloszek, Ryan Feeley, Weehong Tan, Kunpeng Sun, and Andrew Packard. Control applications of sum of squares programming. In Didier Henrion and Andrea Garulli, editors, *Positive Polynomials in Control*, volume 312 of *Lecture Notes in Control and Information Sciences*, pages 3–22. Springer Berlin / Heidelberg, 2005.

[20] Mark M. Tobenkin, Ian R. Manchester, and Russ Tedrake. Invariant funnels around trajectories using sum-of-squares programming. *Proceedings of the 18th IFAC World Congress, extended version available online: arXiv:1010.3013 [math.DS]*, 2011.

[21] Andrew J. Barry, Anirudha Majumdar, and Russ Tedrake. Safety verification of reactive controllers for UAV flight in cluttered environments using barrier certificates. In *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[22] C.G. Atkeson. Using locally weighted regression for robot learning. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 958–963. IEEE, 1991.

[23] Deisenroth, M.P., Peters, J., Rasmussen, and C.E. Approximate dynamic programming with gaussian processes. In *American Control Conference, 2008*, pages 4480–4485. Ieee, 2008.

[24] MP Deisenroth and CE Rasmussen. Efficient reinforcement learning for motor control. *10th International PhD Workshop on Systems and Control*, Sep 2009.

[25] Frazzoli and E. *Robust Hybrid Control for Autonomous Vehicle Motion Planning*. PhD thesis, Massachusetts Institute of Technology, June 2001.

[26] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[27] D.R. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 741–750. ACM, 2002.

[28] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, pages 97–104. ACM, 2006.

[29] B.L. Stevens and F.L. Lewis. *Aircraft Control and Simulation*. John Wiley & Sons, Inc., 1992.

[30] M. Drela and H. Youngren. Xfoil 6.94 user guide. 2001.

[31] R.E. Sheldahl and P.C. Klimas. Aerodynamic characteristics of seven symmetrical airfoil sections through 180-degree angle of attack for use in aerodynamic analysis of vertical axis wind turbines. Technical report, Sandia National Labs., Albuquerque, NM (USA), 1981.

[32] C. R. Hargraves and S. W. Paris. Direct trajectory optimization using nonlinear programming and collocation. *J Guidance*, 10(4):338–342, July-August 1987.

[33] H. Kwakernaak and R. Sivan. *Linear optimal control systems*, volume 172. Wiley-Interscience New York, 1972.

[34] D. Honegger, P. Greisen, L. Meier, P. Tanskanen, and M. Pollefeys. Real-time velocity estimation based on optical flow and disparity matching. *To appear, Intelligent Robots and Systems (IROS) 2012*, October 2012.