A Closed Loop Research Platform that Enables Dynamic Control of Wing Gait Patterns in a Vertically Constrained Flapping Wing - Micro Air Vehicle

Hermanus Van Botha
Wright State University

Follow this and additional works at: http://corescholar.libraries.wright.edu/etd_all
Part of the Computer Engineering Commons

Repository Citation
http://corescholar.libraries.wright.edu/etd_all/1483
A CLOSED LOOP RESEARCH PLATFORM THAT ENABLES DYNAMIC CONTROL OF WING GAIT PATTERNS IN A VERTICALLY CONSTRAINED FLAPPING WING - MICRO AIR VEHICLE

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering

by

HERMANUS VAN NIEKERK BOTHA
B.S., Wright State University, 2013

2016
Wright State University

John C. Gallagher, Ph.D.
Thesis Director

Mateen M. Rizki, Ph.D.
Chair, Department of Computer Science & Engineering

Committee on Final Examination

John C. Gallagher, Ph.D.

Travis E.W. Doom, Ph.D.

Mateen M. Rizki, Ph.D.

Robert E.W. Fyffe, Ph.D.
Vice President for Research and Dean of the Graduate School
ABSTRACT

Botha, Hermanus. M.S., Department of Computer Science and Engineering, Wright State University, 2016.

Research in Flapping Wing - Micro Air Vehicles (FW-MAVs) has been growing in recent years. Work ranging from mechanical designs to adaptive control algorithms are being developed in pursuit of mimicking natural flight. FW-MAV technology can be applied in a variety of use cases such a military application and surveillance, studying natural ecological systems, and hobbyist commercialization. Recent work has produced small scale FW-MAVs that are capable of hovering and maneuvering. Researchers control maneuvering in various ways, some of which involve making small adjustments to the core wing motion patterns (wing gaits) which determine how the wings flap. Adaptive control algorithms can be implemented to dynamically change these wing motion patterns to allow one to use gait based modification controllers even after damage to a vehicle or its wings occur. This thesis will create and present a hardware research platform that enables hardware-in-the-loop experimentation with core wing gait adaptation methods.
## Contents

1 Introduction ................................................. 1  
   1.1 Description ........................................... 2  
      1.1.1 Flapping Wing Vehicles ............................. 2  
      1.1.2 Existing Insect-Like Air Vehicles .................. 5  
      1.1.3 Open Questions ................................. 7  
   1.2 Motivation ............................................ 8  
      1.2.1 Tasks ........................................ 8  
   1.3 Summary of work done ................................ 9  
   1.4 Outline of Thesis ................................... 9  

2 Background ................................................ 10  
   2.1 The Vehicle ......................................... 10  
      2.1.1 Operation .................................... 12  
      2.1.2 Wing Level Control ............................. 15  

3 Design & Implementation .................................. 19  
   3.1 Description ......................................... 19  

4 Hardware .................................................. 21  
   4.1 Low Level Controller Layer ......................... 22  
      4.1.1 Controllers .................................. 23  
      4.1.2 Multiplexing ................................ 28  
   4.2 High Level Controller ............................... 33  
      4.2.1 Computer on Module ......................... 35  
      4.2.2 COM Interfacing .............................. 38  

5 Software .................................................. 41  
   5.1 Initialization ....................................... 41  
   5.2 Flight Parameter Control ............................. 44  
   5.3 Flapping Pattern Control ............................. 47  
   5.4 Closing the Loop ................................... 48  
      5.4.1 Power Management .............................. 48
# List of Figures

1.1 Bird-like Ornithopter .................................................. 2
1.2 Commercial Bird-like Ornithopter .................................. 3
1.3 Harvard Robo-Bee ...................................................... 4
1.4 First Takeoff FW-MAV ................................................ 5
1.5 AFRL Aluminium FW-MAV ............................................. 7

2.1 3D printed FW-MAV .................................................... 10
2.2 Assembled printed FW-MAV ......................................... 11
2.3 Four Bar Linkage ....................................................... 12
2.4 Asymmetric Sinusoidal Waveform .................................. 14
2.5 AFRL-Control-Loop ................................................... 15
2.6 Split-Cycle With Bias: $\delta < 0$ ..................................... 18
2.7 Split-Cycle With Bias: $\delta > 0$ ..................................... 18

4.1 Closed-loop Control Flow Overview ............................... 21
4.2 Communication Interface Overview ................................. 24
4.3 Microcontroller P1 ...................................................... 25
4.4 Microcontroller P2 ...................................................... 25
4.5 Microcontroller P2 ...................................................... 27
4.6 Module Controller Multiplexer ...................................... 28
4.7 Controller Inverters ..................................................... 29
4.8 Communication Multiplexers ........................................ 30
4.9 Actuator Control & Feedback Signals ............................... 31
4.10 Low Level PCB - 1 ...................................................... 33
4.11 Low Level PCB - 2 ...................................................... 34
4.12 Gumstix Overo COM ................................................. 35

5.1 Power Management PCB ............................................. 49
5.2 Experimentation Environment ....................................... 51
5.3 Tracking Sample ....................................................... 52
List of Tables

2.1 Example table of four bar linkage lengths ................................. 13
4.1 Control Signal Multiplexing ............................................... 29
4.2 Communication data signals multiplexing ................................. 31
4.3 Actuator control and feedback multiplexing ............................. 32
4.4 The select signal GPIO pins on high level controller .................. 36
4.5 The GPIO Pins And Their Default Multiplexing Status For SPI Busses 37
Acknowledgment

I would like to thank Dr. Sanjay Boddhu and his family for his guidance and wisdom and for helping me debug the strangest issues in the middle of the night. A special thanks to Dr. John Gallagher for giving me the opportunity to work on his project and for his wisdom in all things Batman.

I would also like to thank my advisory committee for their support in approving this thesis and allowing me to gain my M.S. degree in Computer Engineering.

This material is based upon work supported by the National Science Foundation under grant numbers CNS-1239196, CNS-1239171, and CNS-1239229 Opinions expressed in this poster are those of the authors and do not necessarily reflect those of the National Science Foundation.
Dedicated to:

My family and friends that have supported me from the very beginning of this journey, that have never once doubted me. For standing with me and encouraging me every step of the way. Ek is lief vir julle.

Brooke, because without you I don’t know where I would be or what I would be doing.
Introduction

Flight is for the birds. That was until humans made airplanes and became superior at flying. Are fixed wing aircraft really better? Of course being able to transport hundreds of people, tons of cargo, for thousands of miles at near supersonic velocity is in some ways superior. However, at smaller scale such as the African swallow, or the slightly smaller European swallow, speed might not be as important as maneuverability. In the future one might ask what is the maneuverability of an unladen wing flapper. The answer should undoubtedly be: "What do you mean, natural or robotic?".

The general term for an air vehicle that flaps its wings to fly is ornithopter. Many different ornithopter designs and implementations exist, most of which are based on birds and insects. Ornithopters have been researched throughout history for various reasons. Some research attempts develop different forms of flight, other research desire the study of nature and ecology. Surveillance applications and robotic control theory are particularly attractive areas of study regarding ornithopters. Ornithopters can potentially be nearly silent and highly mobile indoors and in constrained spaces. Imagine a humming bird and how effectively it can navigate between leaves, twigs and branches. Exploiting similar capability in artificial ornithopters could be of high value.
1.1 Description

1.1.1 Flapping Wing Vehicles

Ornithopters generate lift by means of flapping wings. One tends to immediately imagine a bird when told about something that flaps wings to fly. One might also imagine an insect such as a bee, moth, or fly. Similarly, for the sake of discussion we assume that there are generally two major categories that ornithopters fall into namely bird-like and insect-like ornithopters. There are some distinct differences between bird-like ornithopters and insect-like ornithopters. The two categories are most notably different in terms of size, flight control, and force generation methods.

Bird-like ornithopters are configured to mimic bird flight, which is to say that it they attempt to fly in the direction that the front/head of the vehicle is pointing towards. Construction configurations tend to have large wings in the front to mid section, long and slender bodies, and have control surfaces similar to tail wings located at the rear of the vehicle. An example of a bird-like ornithopter is shown in Figure 1.1.

Figure 1.1: An example image of a bird-like ornithopter.\(^1\)

\(^1\)http://terpconnect.umd.edu/~skgupta/UMdBird/
ornithopter can be seen in Figure 1.1. These bird-like ornithopters flap their wings up-and-down to produce lift as well as forces that propel the device forward. The tail wing is responsible for the majority of flight control as well as assisting in gliding. The tail wing controls flight by introducing drag similar to a rudder on a boat which causes the directional change. By twisting or turning the tail wing, an imbalance in drag on one side will cause a corresponding change in heading in the front of the vehicle. Bird-like ornithopters are similar to fixed-wing aircraft partly in build and operation in that they have similar glide abilities. Similar to fixed-wing aircraft, bird-like ornithopters are common and commercially available. These ornithopters are cheap and easy to manufacture, even hobbyists can easily build their own models with wood, plastic, and rubber bands. More elaborate models exist that can have multiple control surfaces as seen in Figure 1.2.

Figure 1.2: An elaborate bird-like ornithopter with multiple control surfaces.

https://www.tintoyarcade.com/rubberband-ornithopter-bird.html?language=en&currency=USD&gclid=CjwKEAjwrOO3BRCX55-L9_WojHoSJAAPxcSPKPtWzEW7tK7xSlpaUAAj-BE0p10y6U6FUx5avtsxoCSj_w_wcB
Commercial insect-like ornithopters are more rare and more expensive. These ornithopters are generally implemented for research purposes. Many insect-like vehicles do not have tail-wing (or alternative) control surfaces. These models use only their wings to produce propulsion. Therefore the wings are responsible for all control and lift. These vehicles are configured in a ”pendulum stable” position. This is to say that the vehicle is oriented vertically instead of a horizontal positioning as seen in the bird-like models. The wings are located high, while the rest of the vehicle body hangs below the wings. An example insect-like ornithopter can be seen in Figure 1.3.

These insect-like ornithopter models flap their wings back-and-forth which result in propulsion generation. Similar to a person treading water in place and twisting their hands, the wings will move back-and-forth while pivoting in a scooping motion. The vehicle can hover in place on average by keeping a consistent symmetrical sinusoidal pattern of flapping back-and-forth. By

---

3https://www.hobbyking.com/mobile/viewproduct.asp?idproduct=77822
4http://www.smithsonianmag.com/arts-culture/biomimetic-design-means-well-all-be-living-a-bugs-life-1558896
making subtle changes to that sinusoidal flapping pattern, the vehicle can increase or reduce lift and also perform complicated maneuvers such as rotational, forward, and backward movement.

1.1.2 Existing Insect-Like Air Vehicles

Harvard researchers created an insect scale Flapping Wing - Micro Air Vehicle (FW-MAV) which successfully generates vertical lift and claimed to be the first to do so[1]. The first take-off that the Harvard FW-MAV performed was constrained with two guide wires attached to the vehicle shoulder/wing areas. The guide wires allowed the vehicle to move vertically freely. The demonstration served the purpose that engineering insect scale robotic flight was indeed possible. An example of the vehicle design can be seen in Figure 1.1.1. Consequently, the team made further progress toward controlled flight and demonstrated that they could hover vertically while still having guide wires to constrain horizontal movement[2]. The team made another breakthrough in
flapping wing control five years after the first flight of the Harvard FW-MAV. A new FW-MAV design was created and is smaller and more lightweight[3]. The vehicle generates sufficient forces by flapping its wings in unison according to a sinusoidal pattern[4]. As mentioned in subsection 1.1.1, the insect-like FW-MAV is modeled to be pendulum stable. The wings are located high on the air-frame while the rest of the body hangs below. The new Harvard FW-MAV scale can be seen in the Figure 1.4. The Harvard vehicle is manufactured by means of a specialized origami folding technique known as monolithic fabrication of printed circuit microelectromechanical systems (PC-MEMS)[5][6]. The technique involves the use of lasers on multiple layers of printed circuit board substrate on scales ranging from mm to μm. This complicated procedure allowed the Harvard team to create the very small and light weight FW-MAVs seen in Figure 1.3 and Figure 1.4. The most notable design decision in the Harvard model is that it is tethered for power management and control. The additional weight of adding a power source and control architecture would defeat the FW-MAVs ability to overcome gravitational pull.

Another research group from the Air Force Research Laboratory (AFRL), created a larger insect-like FW-MAV based on the Harvard model[7][8][9]. That vehicle model does not yet produce flight enabling vertical lift, but is situated on a flat circular puck-like structure which floats on a cushion of air and is capable of maneuvering across a air-hockey table-like surface. The FW-MAV on its platform mimics hover by constraining its vertical mobility. The FW-MAV therefore is constrained to two degrees of freedom horizontally. The model has independently controllable wings which both flap according to sinusoidal wing gait patterns which generates sufficient forces for movement and control. This model is pendulum stable in design with two wings situated high on the vehicle frame, while the body is attached to a circular puck. The vehicle is fabricated from aluminum stock which is milled to produce individual components which are mechanically attached to form the entire FW-MAV. The AFRL vehicle can be seen in Figure 1.5. Unlike the Harvard model, this model is not tethered and carries a power source and control architecture during operation.
1.1.3 Open Questions

It is clear that small scale FW-MAVs introduce a series of critical problems such as power supply, fabrication, weight management, calibration, and control methods. The examples mentioned in subsection 1.1.2 and many others address the critical problems over the course of years of experimentation. Experimenting with new vehicles and techniques is required as the aerodynamics of FW-MAV’s are difficult to model in simulation. When researchers choose to model and simulate FW-MAVs, they do so in idealized conditions since actual real world scenarios are incredibly difficult to model. While the critical problems such as power management, fabrication, and weight management are important, the problem of control is notably difficult. Control issues for FW-MAVs at small scale become particularly difficult and warrant special investigation. FW-MAV fabrication can introduce imperfections that might not conform to generic flight control models, which in turn could cause control imprecision or failure. Even if sufficiently precise fabrication
methods are developed, wing membranes and moving parts degrade or become damaged over time. FW-MAV operational damage could force critical deviation away from expected control models. Therefore, static controllers require validation in fragile robotic systems such as FW-MAVs before dynamic adaptive controllers can be experimented with.

1.2 Motivation

The primary purpose of this thesis is the design and creation of control electronics capable of supporting experimentation of both static and dynamic controllers for insect-like FW-MAVs. The existing AFRL control methods and hardware served as inspiration and will be incorporated into a redesigned control platform. This involves redesigning the low level static controllers, creating a high level controller for interfacing, producing an API for which enables closed loop control and learning capabilities.

1.2.1 Tasks

Existing embedded control circuitry is responsible for ensuring FW-MAV sinusoidal wing flapping patterns. The control interface enables a user to command a pair of parameters that introduce subtle differences to the fundamental sinusoidal flapping pattern. A user can therefore provide control input which would allow the FW-MAV to perform maneuvers such rotational, forward, and backward movement. The new low level control redesigns that controller to support an additional parameter that enables more detailed control of flapping patterns.

An adaptive controller could alter the fundamental flapping pattern in situations such as described previously, where the FW-MAV active operation might deviate too far from the low level control model. Adaptive controllers can attempt to restore the FW-MAV to the expected low level model, or find a new model that might restore some flight functions.

A closed-loop control environment is required for experimentation, in addition to enabling access of low and high level controllers.
1.3 Summary of work done

A modified FW-MAV research platform has been created which successfully dynamically transfers new commutation flapping patterns without interfering with normal active operation. The FW-MAV platform additionally serves as a research testbed for use in testing machine learning algorithms and flight controller implementations. The architecture circuitry is divided in two modules. One module is responsible for control and interfacing and the other is a power delivery module for the actuators and controller interface for the FW-MAV. Additionally, a low and high level API has been created for an end user to perform FW-MAV parameter commands such as flapping frequency and split cycle modulation adjustment. The platform also supports a tracking system which allows for autonomous closed loop control capability for a FW-MAV.

1.4 Outline of Thesis

The Background chapter 2, describes the FW-MAV prototype that is used in this research platform. The description includes the physical appearance, mechanical operation, and control methodology.

The Design & Implementation chapter 3, describes the components that complete the research platform. The first component is a hardware designs required to control, command, and interface with a FW-MAV. The second component is an API which provides user control and interfacing capabilities with FW-MAV controllers. The third component is a testing and experimentation environment that provides closes-loop control access.

The Conclusion chapter 6, reiterates the work done which includes the 3 components which complete the research platform.
Background

2.1 The Vehicle

This section will discuss the basic operation of the AFRL FW-MAV prototype. This includes information regarding the mechanical design, components, and control methods. The vehicle prototype seen in 2.1 is a pendulum stable four bar minimally actuated insect-like FW-MAV. The body and linkages are 3D printed and designed to hold two actuators and a wing per flapping arm. The
motors rotate the linkages that connect to the arms with wings attached which then move back and forth in a flapping motion. The actuators connect to control circuitry as shown in Figure 2.2. All vehicle components are hand assembled which includes hand finishing, fitting, and attaching printed components. A wing consists of carbon fiber spars super epoxied to lightweight membranes. The fragile wings can take up to 3 days to manufacture. The manufacturing process can be expensive and difficult to accomplish. The FW-MAV prototype is also sensitive to damage in a number of ways. The linkages and motor housings tend to break as the plastic becomes brittle over time and with temperature fluctuations. The FW-MAV is housed on a puck-like structure which constrains the prototype vertically. When the puck and FW-MAV are placed on the testing environment surface, a cushion of air will cause the prototype to hover.

Figure 2.2: Assembled 3D printed FW-MAV including motors and excluding wings[11].
2.1.1 Operation

The FW-MAV flaps each wing individually by linking an actuator to each wing[7]. Each wing is eventually coupled to an actuator via a four bar linkage as seen in Figure 2.3. Rotational motion generated by each actuator is then transformed into reciprocating motion in the form of wing flaps.

A Brushless DC Motor (BLDC) is coupled to the crank as pictured in Figure 2.3. The BLDC rotates the crank which is coupled to a coupler bar which is linked to a rocker arm. The rocker arm sways back-and-forth as the crank rotates. Four bar linkage kinematics are non-linear and therefore produce non-symmetric motion patterns on output linkages such as the rocker arm. In other words, with constant rotation of the crank, the back-and-forth velocities of the rocker will not be equal and opposite in magnitude[7].

A flapping cycle is defined as a combination of two half strokes namely a forward stroke and an backward stroke. Assuming an actuator rotates at a constant rate, the resulting rocker/wing linkage movement over time will be some asymmetric kinematic sinusoid such as shown in Figure

Figure 2.3: An example of a four bar linkage[7].
2.4. The relationship between rocker and crank angles in a four bar linkage is expressed in 2.1:

\[
\phi_l = 2 \arctan 2 \left( \frac{-K_1(\theta_l) \pm \sqrt{K_1(\theta_l)^2 - K_3(\theta_l)^2 + K_2(\theta_l)^2}}{K_3(\theta_l) - K_2(\theta_l)} \right) \tag{2.1}
\]

Where \( \phi_l \) is the rocker angle, \( \theta_l \) is the crank angle, \( \arctan 2 \) is a 4 quadrant arctangent function that is dependent on 2.2, 2.3, and 2.4:

\[
K_1(\theta_l) = -2l_1l_3\sin\theta_l \tag{2.2}
\]

\[
K_2(\theta_l) = 2l_3(l_0 - l_1\cos\theta_l) \tag{2.3}
\]

\[
K_3(\theta_l) = l_0^2 + l_1^2 - l_2^2 + l_3^2 - 2l_0l_1\cos\theta_l \tag{2.4}
\]

Where \( l_0 \) is the distance between line of centers, \( l_1 \) is the crank length, \( l_2 \) is the coupler length, and \( l_3 \) is the rocker length[12]. The linkage length parameters for the example can be found in Table 2.1.

<table>
<thead>
<tr>
<th>Linkage</th>
<th>Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_0 )</td>
<td>24.2</td>
</tr>
<tr>
<td>( l_1 )</td>
<td>5.76</td>
</tr>
<tr>
<td>( l_2 )</td>
<td>25.75</td>
</tr>
<tr>
<td>( l_3 )</td>
<td>8.38</td>
</tr>
</tbody>
</table>

Table 2.1: Example table of four bar linkage lengths

The asymmetric sinusoid that is produced by the non-linear four bar linkage system can be seen in Figure 2.4. It is clear that the first half(downstroke) is faster than the second half(upstroke). The asymmetric cycle pictured in Figure 2.4 is problematic to the operation of the FW-MAV. The wing will experience higher dynamic pressure during the faster moving first half stroke than the slower moving second half stroke. This will lead to a net gain in cycle averaged horizontal force. This is an uncontrolled instance of force production. In order to control the force production of the FW-MAV, the full flapping cycle must be controlled. As mentioned previously, the asymmetrical
sinusoid is produced when the crank is rotated with constant velocity. The crank can be rotated with varied velocities such that the sinusoid is forced to become symmetrical. The process in which the crank velocities are computed to output a desired rocker motion is called kinematic inversion. The AFRL team generated a desired symmetric sinusoidal rocker waveform. Then applied $\phi_l$, Equation 2.1 to the waveform to generate a lookup table of $\theta_l$ crank angles. They also generated another lookup table which maps the crank velocities required at each crank position to generate the desired symmetric sinusoidal wave. The velocity lookup table is generated for each index position $i$ by taking the difference between $i-1$ and $i+1$ in the crank position lookup table.

Each wing actuator has a dedicated microcontroller responsible for the position lookup and control of motor commutation via a proportional integral derivative (PID) controller. The onboard PID controller tracks crank position via an encoder contained within the BLDC wing actuator. The
controller compares the crank position and velocity with the previously mentioned desired crank position and velocity lookup tables. The PID controller generates cumulative error/deviations from the difference between desired and current crank positions and velocities. The deviation signals are then applied to adjust actuator control signals to match actual with desired crank locations and velocities. Each actuator is controlled independently by a small microcontroller and some amplifying control circuitry. An additional third microcontroller is responsible for collecting and parsing controller commands from an RF receiver and relaying the parameters to wing controllers.

2.1.2 Wing Level Control

![Open Loop Control Flow](image)

Figure 2.5: Open Loop Control Flow.

The AFRL FW-MAV low level PID controllers work directly with the commutation tables...
which contain the reference positions and velocities required maintain a symmetrical sinusoidal flapping pattern. The PID controllers generate an error value from the difference between desired commutation elements to current commutation elements. The error is then applied to adjust the Pulse Width Modulation(PWM) values that energize the coils within its BLDC actuators which generate velocity and torque. The commutation tables divide a full rotation of a crank linkage into 1024 segments. Each element in the velocity commutation table represents the amount of time that the crank linkage should take to move an incremental index position from the position reference table. That is to say that if the crank rotates from position \( i-1 \) to \( i+1 \), the \( i'th \) position in the velocity table represents the time it should take to rotate that distance.

The symmetrical sinusoidal flapping pattern generates equal amounts of force in both flapping directions (back-and-forth) and therefore cancel out. When the FW-MAV flaps its wings according to the symmetrical pattern, the first half a wing flap is represented by the first half of the sinusoidal waveform and is equal to the second half of the waveform which represents the second half of the wing flap. In theory, when both halves of a complete wing flap are equal, then the FW-MAV should hover in place. One can then imagine that if the first half of the wing flap is slightly faster and the returning half of the wing flap is slightly slower, on average the FW-MAV would generate more force in one direction than the other. This is called split-cycle constant period frequency modulation(SCCPF)[13][14]. One can apply a bias to the sinusoidal wing flap waveform in such a way as to keep the period constant, but the first half of the sinusoidal waveform has a smaller frequency and the second half is a larger frequency. The Equations 2.5 and 2.6 represent the forward and backward wing strokes positions:

\[
\phi_F = \cos(\omega t), \quad 0 < t < \frac{\pi}{\omega}
\]  
(2.5)

\[
\phi_B = \cos(\omega t), \quad \frac{\pi}{\omega} \leq t < \frac{2\pi}{\omega}
\]  
(2.6)

Where \( \omega \) is the symmetrical sinusoidal wing flapping waveform. The SCCPFM control model is responsible for the production of force during a flapping cycle by altering wing flapping charac-
teristics to have a higher magnitude of velocity in one half and equal lower magnitude in the other half. The Equations 2.7 and 2.8 represent the forward and backward wing stroke positions when SCCPFM is applied:

\[ \phi_F = \cos((\omega - \delta)t), \quad 0 < t < \frac{\pi}{\omega - \delta} \]  \hspace{1cm} (2.7)

\[ \phi_B = \cos((\omega + \sigma)t + \xi), \quad \frac{\pi}{\omega - \delta} \leq t < \frac{2\pi}{\omega} \]  \hspace{1cm} (2.8)

Where \( \delta \) is the split-cycle bias parameter. The parameter \( \xi \) is responsible for phase shifting 2.6 to be compatible with the frequency change in the waveform:

\[ \xi = \frac{-2\pi\delta}{\omega - 2\delta} \]  \hspace{1cm} (2.9)

The parameter \( \sigma \) ensures that the split-cycle waveform ends with the correct period, which should be the same period as if the sinusoidal waveform was symmetrical:

\[ \sigma = \frac{\delta\omega}{\omega - 2\delta} \]  \hspace{1cm} (2.10)

An SCCPFM example is shown in Figure 2.6 where a bias \( \delta < 0 \) is applied to Functions 2.7 and 2.8. Alternatively, if \( \delta > 0 \) the resulting waveform looks like the Figure 2.7.
Figure 2.6: Split-Cycle With Bias: $\delta < 0$.

Figure 2.7: Split-Cycle With Bias: $\delta > 0$. 
Design & Implementation

As mentioned in the introduction, the purpose of this thesis is the design and construction of a research platform which enables the ability to perform experiments with static and adaptive controllers on a FW-MAV. This is done by incorporating and redesigning the existing FW-MAV prototype controller systems described in the Background chapter. This work implements the existing control software and adds an API for accessibility. This work also redesigns the existing hardware controller into a low level control layer. Existing hardware controller designs have shown to have functioning low level wing control capabilities and have maneuvered a vertically constrained FW-MAV across an air table experimentation surface. The pre-existing hardware controller designs provide the necessary groundwork to build upon and implement this research platform. The added benefit of incorporating the existing control software and hardware designs is the avoidance of pitfalls such as rewriting code, debugging control algorithms, and testing and debugging new microcontroller or hardware implementations.

3.1 Description

Three components that complete this research platform are as follows:

- A low level hardware control layer which contains design implementations that are different from the pre-existing hardware controllers. Additionally, a high level controller is introduced.

- A software API, which is accessible through the newly added high level controller and is
required to interface with the redesigned low level hardware control layer. An interfacing library which is hidden from user access is also added and is responsible for flight control parameter conversion as well as coordinating communication between the high level controller and low level controller layer. The API provides simplified access to FW-MAV flight control parameters in addition to controlling wing flapping patterns.

• A testing environment which closes control and learning loops. The testing environment includes a FW-MAV flight experimentation environment, position tracking implementation, and a remote access application.
Hardware

The research platform makes use of a working implementation of the prototype FW-MAV described in the Background chapter. The FW-MAV is capable of producing forces with the abil-
ity to propel the vehicle across an experimentation environment similar to the one described in the Background chapter. The research platform provides a more robust and modular capability of dynamically altering wing flapping patterns. Moreover, the research platform provides closed loop autonomous functionality essential for adaptive control and learning. The Figure 4.1 shows a high level overview containing the primary components and sub-components which complete the research platform. The control flow is as follows: A user controller algorithm generates FW-MAV commands based on tracking feedback. The commands are transferred over a wireless connection to the high level controller. The high level controller containing the user accessible API, sends translated commands to the low level hardware control layer via the interfacing library. The low level hardware control layer interprets the commands and adjusts the FW-MAV flight characteristics accordingly. The motion tracking system captures new up to date FW-MAV locations and forwards them to the user controller algorithm for feedback.

4.1 Low Level Controller Layer

The low level hardware control layer incorporates elements from the hardware controllers discussed in the Subsection 2.1.2. The pre-existing hardware controllers were originally created with the intent of demonstrating split-cycle cosine waveform control. The work provided an interface to dynamically adjust FW-MAV flapping frequency, Omega and split-cycle cosine bias, Delta. In addition to being able to dynamically change the Omega and Delta parameters, in this work the capability of dynamically altering wing flapping trajectory characteristics for the FW-MAV via a third parameter is provided. The commutation table which discussed in Subsection, 2.1.1 which defines the crank reference and rotational velocity characteristics is the third controllable parameter.

Future work anticipates a closed-loop autonomous system where all control computation should occur on the FW-MAV. This is to say that in the future, adaptive controllers will be stored and executed from the FW-MAV platform, not from a remote computer location. The FW-MAV
research platform therefore must contain a computational resource with adequate memory and processing capabilities to perform adaptive learning algorithms. The computational resource serves as the high level controller mentioned earlier. The high level controller interfaces with the low level controller via a communication bus in order to transfer flight parameters and commands.

4.1.1 Controllers

The low level controller layer incorporates some elements from the pre-existing hardware controller designs while opting for alternative communications solutions. The RF module and its accompanying microcontroller mentioned in 2.1.2 and shown in Figure 2.5 is removed. The low level hardware control layer is designed with the following additions:

- Existing SPI channels redirected and software updated to interface with a high level controller.
- Capability of transferring a commutation table parameter is added.

It is important to understand the limitations of the low level microcontroller architecture to understand the design decisions made in this work. Firstly, the low level microcontrollers support 8KB of static random access memory (SRAM), 4.324KB of which is already occupied with the existing control software\(^1\). The section of code responsible for taking up the largest amount memory space is the commutation table containing 1024 elements of 32bit long elements. This amounts to 4KB of space required for a single actively accessed table. Transferring a new commutation table requires that the old table be overwritten. However, if one were to attempt to replace the contents of the commutation table during active operation of the FW-MAV, one stands the risk of causing PID control issues. Some elements that are new and some elements that are original might be present in the table which is currently being referenced commutated by the PID controller. One can imagine that in this scenario that can be a significant problem. If the PID controller passes

from original to new elements of the table, it might cause the actuators to perform in unpredictable ways which stands the risk of damaging the vehicle. A potential solution to this problem is to introduce an additional array table on the microcontroller which can be accessed and written to while the PID controller references from the original table. Unfortunately the additional 4KB of storage required for an additional array table would exceed the available SRAM space that the microcontroller supports. There are also no other microcontrollers with the same architecture as the existing microcontroller which support adequate amounts of SRAM. Logically, the solution to this problem should bypass the microcontroller memory limitations as well as not interfere with active vehicle performance. The microcontroller limitations to transferring a new commutation array table are listed as follows:

- Pre-existing control software cannot be changed and a new microcontroller architecture cannot be selected.
• Transferring a new commutation table interferes with normal FW-MAV operation.

• Microcontroller memory is too little to contain an additional array table as a buffer.

Figure 4.3: The first of the two identically configured microcontrollers - P1.

Figure 4.4: The second of the two identically configured microcontrollers - P2.
A multiplexed control and communication system is designed to avoid interfering with active operation and to bypass microcontroller memory limitations[15]. Two copies of the microcontroller exist simultaneously and toggle responsibilities to control wing actuators and communicate parameters respectively. This is to say that where there used to be one microcontroller per wing actuator, there are now two microcontrollers per wing actuator. Two identically configured and programmed microcontrollers as seen in Figure 4.3 and Figure 4.4 coordinate to control an actuator and to communicate with the high level controller respectively. The high level coordinator is responsible for sending coordination signals. All input and output channels that are required for PID commutation control and actuator feedback are configured exactly as those described in the Background section. This solves the first problem, as the control software and microcontroller architecture are kept intact.

Each microcontroller and its supporting control components can be treated as a module. One control module functions as a communication module, while the other functions as an actuator controller. During normal active FW-MAV operation, there is a controller module responsible for actuator control communication module responsible for SPI message transfers. The communication module cannot be in control of a wing actuator while controller module is in control of its wing actuator. If one were to transfer a new commutation table to the low level control layer, all of the SPI data transfer would occur between the high level controller and the communication module. During SPI data transfer, the controller module is left undisturbed for the entirety of the commutation table transfer. The PID control interference as described above is therefore avoided, which addresses second problem.

Both modules are identical and can perform identical tasks. As mentioned previously, the high level controller sends coordination signals between which toggle module responsibilities. When a commutation table is transferred, the modules can perform a control switch as shown in 4.5. This is to say that the controller module will no longer be in control of the wing actuator and instead it will become the communicator module. Simultaneously, the communicator module will no longer be primarily responsible for communications and instead will become the controller module. The
controller module is capable of communication of small data transfers if the correct control signals are generated from the high level controller. However, large continuous message transfers such as a commutation table via the controller module could impede the PID controller which could lead to FW-MAV malfunction or damage and should only be performed via the communication module.

In a scenario mentioned above, a new array table was transferred to the communication module which then performed a control switch. The switch causes the communication module to become the controller module and therefore is controlling the wing actuator with a new commutation table. This successfully bypasses the third problem, which is the microcontroller memory limitations.
4.1.2 Multiplexing

The controller modules are named $P1$ and $P2$ for routing purposes [11]. Both modules have their own distinct set of input and output channels. The channels eventually lead to the same components, such as the wing actuator, amplifier, and SPI interface. All of the channels do so after passing through a range of multiplexers. The above mentioned switch between controller and communication modules is performed through a variety of multiplexers. Multiplexing select signals are the coordination signals controlled from the higher level controller. There are two distinct operations that occur for the switching transition namely, the transfer of control and a transfer of communication. Both transfers occur simultaneously. The high level controller determines the select signals required for switching control and communication. The high-level controller will select two binary signals, one for to select the control module ($IS2(TABLE)$) and one for selecting the communication module ($IS1(SPI)$) and can be seen in Figure 4.6. Each signal from the high level

![Figure 4.6: The multiplexer responsible for maintaining controller and communication modules.](image)

- $1S1(SPI)$
- $1S2(TABLE)$
- $U1-A$
- $U1-B$
- MAX333ACUP
- $+3.3V$
- $1P2-CS$
- $1P1-TS$
controller is used as a multiplexer input select line which routes an output signal to 3.3V(\textit{HIGH}) or GND(\textit{LOW}).

The input select signal \textit{IS1(SPI)} multiplexes to an output signal \textit{1P2-CS}, which determines the module that will be the communication module. The input select line \textit{IS2(TABLE)} multiplexes to an output signal \textit{1P1-TS}, which determines the module that will be the control module. The output signal for \textit{1P1(TS)} and \textit{1P2(CS)} route to module \textit{P1} and \textit{P2} by default respectively. Additionally the multiplexer output signals are also routed to a respective logic level inverter which then routes to the respective opposite module. The inverter stage can be seen in Figure 4.7. The signal line \textit{1P2-CS} is routed to module \textit{P2} as well as to the inverter which produces a signal \textit{1P1-CS} which is routed to \textit{P1}. Similarly, the signal line \textit{1P1-TS} is routed to module \textit{P1} as well as to the inverter which produces a signal \textit{1P2-TS} which is routed to \textit{P2}. Table 4.1 shows the relationship between control signals and module modes(Controller and Communication).

<table>
<thead>
<tr>
<th>\textit{(IS1(SPI), IS2(TABLE))}</th>
<th>\textit{(1P2-CS, 1P1-TS)}</th>
<th>Control Module</th>
<th>Communication Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0)</td>
<td>(1, 1)</td>
<td>P1</td>
<td>P1</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>(1, 0)</td>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td>(1, 0)</td>
<td>(0, 1)</td>
<td>P2</td>
<td>P1</td>
</tr>
<tr>
<td>(1, 1)</td>
<td>(0, 0)</td>
<td>P2</td>
<td>P2</td>
</tr>
</tbody>
</table>

Table 4.1: Control Signal Multiplexing

![Inverter Diagram](image)

Figure 4.7: The inverters that ensure that only one module can be controller and the other be communicator at the same time.
Whenever the signal $1P1-TS$ is $HIGH$, the module $P1$ is set to be the controller and automatically the signal $1P2-TS$ is simultaneously set to $LOW$ and routed to the alternate module $P2$ ensuring that it cannot be a controller module. Similar to the signal $1P1-TS$, whenever $1P2-CS$ is $HIGH$, the module $P2$ is set to be the communication module and automatically the signal $1P1-CS$ simultaneously set to $LOW$ and routed to the alternate module $P1$ ensuring that it cannot be a communication module. It is clear that inverting both control and communication module select signals is to ensure that both modules cannot be in equivalent states at the same. A module can however be both controller and communicator at the same time.

![Figure 4.8: The multiplexer responsible for Multiplexing SPI data signals to the communicator module.](image)

To communicate via SPI, the communication module requires access to three signal lines namely, Data In($1SDI$), Data Out($1SDO$), and SPI Clock($1SCLK$). Multiplexers are again used to ensure that the SPI communication signal lines are routed to the communication module. The multiplexers use communication control signal $1P2-CS$ as a select signal as seen in Figure 4.8.
The data signal ISDI is multiplexed to either 1P1-SDI or 1P2-SDI and the data signal ISDO is multiplexed to either 1P1-SDO or 1P2-SDO. Therefore, when the communication signal select line 1P2-CS is toggled, all communication data from ISDI and ISDO will be routed to the alternative module. The Table 4.2 shows the data SPI data signal lines that are multiplexed to either P1 or P2.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Multiplexed to</th>
</tr>
</thead>
<tbody>
<tr>
<td>1SDI</td>
<td>1P1-SDI or 1P2-SDI</td>
</tr>
<tr>
<td>1SDO</td>
<td>1P1-SDO or 1P2-SDO</td>
</tr>
</tbody>
</table>

Table 4.2: Communication data signals multiplexing

Figure 4.9: The multiplexers responsible for Multiplexing PID feedback and control signals to the controller module and actuator.

All encoder input and actuator control and feedback channels used for PID control for the controller module are also multiplexed. When the multiplexer control signal select line 1P1-TS toggles, all actuator control and feedback channels are routed from one module to the alternative
module. The exact logic described in the communication multiplexing above is applied in this phase. The table of multiplexed actuator control and feedback signals is shown in Table 4.3.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Multiplexed to</th>
<th>Signal</th>
<th>Multiplexed to</th>
<th>Signal</th>
<th>Multiplexed to</th>
</tr>
</thead>
<tbody>
<tr>
<td>1AHI</td>
<td>1P1-AHI or 1P2-AHI</td>
<td>1IHI</td>
<td>1P1-CHI or 1P2-CHI</td>
<td>1HC</td>
<td>1P1-HC or 1P2-HC</td>
</tr>
<tr>
<td>1ALO</td>
<td>1P1-ALO or 1P2-ALO</td>
<td>1CLO</td>
<td>1P1-CLO or 1P2-CLO</td>
<td>1CHA</td>
<td>1P1-CHA or 1P2-CHA</td>
</tr>
<tr>
<td>1BHI</td>
<td>1P1-BHI or 1P2-BHI</td>
<td>1HA</td>
<td>1P1-HA or 1P2-HA</td>
<td>1CHB</td>
<td>1P1-CHB or 1P2-CHB</td>
</tr>
<tr>
<td>1BLO</td>
<td>1P1-BLO or 1P2-BLO</td>
<td>1HB</td>
<td>1P1-HB or 1P2-HB</td>
<td>1CHZ</td>
<td>1P1-CHZ or 1P2-CHZ</td>
</tr>
</tbody>
</table>

Table 4.3: Actuator control and feedback multiplexing

The above discussed control and communication multiplexing system which includes microcontrollers, multiplexers, amplifiers, supporting components, and circuitry are for one half of the vehicle which is the control and command of one wing. A completely identical copy of all of the above also exists for the other wing of the FW-MAV. Figure 4.10 shows a red line that splits the low level control layer PCB into two halves. The purple boxes show the low level controller module pairs. Each purple box has two microcontrollers, one for control, and one for communication. The yellow boxes show the multiplexers that route the actuator control and feedback channels. The orange boxes contain the SPI communication data channel multiplexers. The red boxes contain the amplifiers that power the motors. The high level controller plugs into the PCB which physically connects to the low level hardware layer PCB shown in 4.11. The green box contains the actuator ribbon connectors. The blue box contains ribbon connectors that can be used if an alternative interface input for the high level controller. The salmon box shows the connectors that the high level controller plugs into for interfacing.
4.2 High Level Controller

A new high level controller is introduced which interfaces with the low level control layer via two SPI busses as seen in 4.2. This high level controller provides an interface to send commands and flight characteristics to the low level control layer via an API. A user can therefore create closed-loop flight controller software. As mentioned previously, a computational resource is contained on the FW-MAV. The high level controller platform of choice is a Gumstix Overo Firestorm-Y.
computer-on-module (COM)\textsuperscript{2} for the following reasons:

- Significant computation power and memory for scale.
- Supports Wifi communication.

\textsuperscript{2}\url{https://store.gumstix.com/coms/overo-coms/overo-firestorm-y-com.html}
• Interfaces with PCB mountable connectors for expandability.

• Has a small form factor with minimal footprint size and weight.

• Linux Operating System and development environment.

4.2.1 Computer on Module

The Gumstix shown in Figure 4.12 and the connector pins that plug into the low level control layer PCB are visible on the right side. The Overo Com has 1GB of flash memory, expandable storage via a micro SD-card slot. A wireless antennae enables wifi connectivity to the Gumstix Overo COM. The Overo COM is roughly the size of a stick of gum, hence the aptly named *Gumstix*. The exact dimensions of the Overo COM are 58mm in length, 17mm in width, and 4.2mm in height. The Gumstix Overo COM processor is a Texas Instruments DaVinci DM3730.

---

<table>
<thead>
<tr>
<th>GPIO Pin</th>
<th>Select Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO - 144</td>
<td>2S1(SPI)</td>
</tr>
<tr>
<td>GPIO - 145</td>
<td>2S2(TABLE)</td>
</tr>
<tr>
<td>GPIO - 146</td>
<td>1S1(SPI)</td>
</tr>
<tr>
<td>GPIO - 147</td>
<td>1S2(TABLE)</td>
</tr>
</tbody>
</table>

Table 4.4: The select signal GPIO pins on high level controller

The Overo COM operating system is based off of an embedded Linux distribution. A platform named, the *Yocto Project*[^4] is an open source embedded Linux build system. The Yocto Project provides frameworks and templates that assist in the compilation and cross-compilation of necessary libraries for embedded systems such as the Gumstix Overo COM. The Yocto build system not only assists in creating custom Linux builds, but also in generating cross-compiled binaries for various software suits written for those embedded systems. Embedded COMs usually have bootloaders which assist in the initialization of essential hardware and software such as memory management and file systems. One of the hardware components which the Overo COM supports is the SPI. As mentioned in Subsection 4.1, the FW-MAV consists of two halves, one for each wing actuator. Each half interfaces with the high level controller via a SPI bus. The Overo COM processor supports up to four SPI busses, only two of which are available for access according to the Overo COM design specification[^5]. Furthermore, only one of the two SPI busses is enabled during boot by the bootloader by default. Additionally, the high level controller select signals used to determine the controller and communicator modules in the low level control layer are channeled via GPIO pins shown in Table 4.4. In order to gain access to the second SPI bus the Overo COM bootloader must be customized. There are two tasks to complete in order to customize the bootloader namely:

- Download and compile the Gumstix Overo Yocto Project framework.
- Patch Yocto build files

[^4]: https://www.yoctoproject.org/
The framework is available from github as a repository which contains the steps required to make and build the framework\(^6\). The first build downloads all required packages and libraries and compiles a functioning base Linux distribution as well as supporting bootloader images. Once the compilation is complete, the Gumstix Overo build header files are altered in order to force the bootloader to multiplex GPIO pins that are required for the second SPI bus to function which are not accessible by default. The Table 4.5 shows the GPIO pins that are used by each SPI bus as well as the default GPIO multiplexing status. The board/overo/overo.h header file contains the GPIO multiplexing tables, which are patched as shown in Listing 4.1. The desired changes are

\(^6\)https://github.com/gumstix/yocto-manifest

<table>
<thead>
<tr>
<th>Pin</th>
<th>Purpose</th>
<th>Default Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO - 171</td>
<td>SPI1 - CLK</td>
<td>Enabled</td>
</tr>
<tr>
<td>GPIO - 172</td>
<td>SPI1 - MOSI</td>
<td>Enabled</td>
</tr>
<tr>
<td>GPIO - 173</td>
<td>SPI1 - MISO</td>
<td>Enabled</td>
</tr>
<tr>
<td>GPIO - 174</td>
<td>SPI1 - CS0</td>
<td>Enabled</td>
</tr>
<tr>
<td>GPIO - 175</td>
<td>SPI1 - CS1</td>
<td>Enabled</td>
</tr>
<tr>
<td>GPIO - 88</td>
<td>SPI2 - CLK</td>
<td>Disabled</td>
</tr>
<tr>
<td>GPIO - 89</td>
<td>SPI2 - MOSI</td>
<td>Disabled</td>
</tr>
<tr>
<td>GPIO - 90</td>
<td>SPI2 - MISO</td>
<td>Disabled</td>
</tr>
<tr>
<td>GPIO - 91</td>
<td>SPI2 - CS0</td>
<td>Disabled</td>
</tr>
<tr>
<td>GPIO - 92</td>
<td>SPI2 - CS1</td>
<td>Disabled</td>
</tr>
</tbody>
</table>

Table 4.5: The GPIO Pins And Their Default Multiplexing Status For SPI Busses
made in the `board/overo/overo.h` file and a patch is generated by the Yocto framework which is then included in the build configuration files. The GPIO pins that are repurposed for SPI2 are used for the display data pins by default. Therefore, the display port will not function as intended. This does not affect the usability of the Gumstix COM, as a user interacts with it wirelessly via SSH. Additionally, drivers must be binded to the newly multiplexed SPI2 bus. The `arch/arm/mach-omap2/board-vero.c` file is patched as shown in the Listing 4.2. The Yocto build is performed again which includes the patches. The build generates a Linux image and Bootloader images, which are stored on the Gumstix COM micro-SD storage card.

### 4.2.2 COM Interfacing

The SPI drivers `SPIDEV`\(^7\) that the are packaged into the Overo COM Linux distribution conveniently implement the `IOCTL`\(^8\) abstraction layer. In order for SPI devices to communicate success-

---


static int spi_init(const char *dev) {
    int fd = open(dev, O_RDWR);
    ioctl(fd, SPI_IOC_WR_MODE, &mode); // write Mode
    ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits); // bits per word
    ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed); // max speed hz
    return (fd); }

Listing 4.3: SPI init code

static const char *device1 = "/dev/spidev1.0"; //left side
static const char *device2 = "/dev/spidev2.0"; //right side
static uint8_t mode = SPI_MODE_3;
static uint8_t bits = 16;
static uint32_t speed = 8000000; // 8MHz
static uint32_t delay;

Listing 4.4: SPI Device Declarations

fully, their protocols must be set up identically. This includes setting their control modes (SPI clock polarity), message word sizes, and clock frequencies. The required Linux device configurations are set via IOCTL programatically in the Function 4.1 as demonstrated by the Listing 4.3.

\[private\ spi\_init(dev)\] (4.1)

The static SPI configuration values: \textit{mode}, \textit{bits}, and \textit{speed} are set according to the Listing 4.4. The SPI clock frequency is set to 8MHz, the message word size is 16bits and the SPI is set to \textit{MODE\_3}. The \textit{SPI\_MODE\_3} sets clock idle state at high level and active state at low level. In order to send messages via SPI from the high level controller (Gumstix Overo COM) to the low level control layer a \textit{spi\_ioc\_transfer} struct is created and passed with the IOCTL abstraction layer with the Function call 4.2 as demonstrated in listing 4.5.

\[private\ spi\_tx\_16b(dev, data)\] (4.2)
Listing 4.5: SPI send code

It is important to note that the above is example code for interacting with the SPI IOCTL interface. The *ioctl()* call will return based on success or failure. The code in listing 4.3 and 4.5 are interfacing demonstrations.
Software

The high level controller contains two software packages, first is a library which is responsible for interfacing with the low level control layer via SPI and IOCTL layers, the second is a user facing API. The API provides a user with simple flight characteristic commands described in the Background section as well as the additional commutation table altering parameter discussed above.

The most two most important parameters that are required in order to control the flight of the FW-MAV are $\Delta$ and $\Omega$. Without the ability to control $\Delta$, the vehicle would not be able to produce cycle-averaged forces which would result in a vehicle which does not move. $\Omega$ is important as without it, no flapping frequency would be controllable which further impedes the ability to control vehicle movement. Furthermore, the ability to programmatically alter the flight parameters is essential for the purpose of realizing an autonomous and closed loop controlled system.

5.1 Initialization

The following are public calls that are visible and available for a user. Each function call potentially makes calls to private functions are required for interfacing with the low level control layer.

\[ public\ init\_Wings() \]  

(5.1)

A user must for any given experimentation session initialize the API. The function 5.1 call will perform a variety of initialization procedures which are essential for the operation of the FW-
int gpio_export(int pin) {
    char buffer[3];
    ssize_t bytes_written;
    int fd = open("/sys/class/gpio/export", O_WRONLY);
    bytes_written = snprintf(buffer, BUFFER_MAX, "%d", pin);
    write(fd, buffer, bytes_written);
    fsync(fd);
    close(fd);
    return; }

Listing 5.1: GPIO File Descriptor Export Procedure

MAV. The first sequence that the function performs is the low level control layer signal generation. The signals discussed in 4.1, are produced by the high level controller which are generated from Linux GPIO file descriptors. The GPIO file descriptors must initially be exported via the function 5.2 which in turn enables user level access as seen in Listing 5.1.

\[\text{private } \text{gpio_export}(\text{pin})\] (5.2)

The function call 5.2 call takes a pin parameter which describes a GPIO pin which can be made user accessible. When the pin number is written to the "/sys/class/gpio/export" Linux file descriptor that pin is made available for read or write by a user. The four select signal GPIO pins shown in Table 4.4 that are responsible for selecting the wing actuator controller module and communication control module on the lower level controller are exported. Pins 146 and 147 are configured to be output pins as they are one directional pins while pins 144 and 145 are bidirectional.

Two SPI device configurations are defined globally as shown in Listing 4.4 and are initialized by passing the devices into the function 4.2, called in Listing 4.3. After the SPI devices are initialized and the GPIO ports exported, the API is ready to initialize the controller modules. The initialization procedure for the controller modules are essential as each microcontroller must complete a startup phase which sets initial parameter values and slowly increases the flapping frequencies of each wing. The select signal GPIO pins are forced to be a LOW signal. The microcontroller
for both the left wing and the right wing halves is therefore configured to be the control module as well as the communication module. This means that while the module is in control of its wing actuators, it is also capable of SPI communication. A *READ_ENC* command is transferred to the module which forces the microcontroller to read the stationary actuator crank position. This is so that the controller module has a starting reference crank position.

Commands that are sent to the low level modules might arrive out of order or experience noise during transfer. One way to confirm and ensure that the correct command was successfully received by the low level modules is to have the controller respond messages back to the high level controller API. The returning command can be compared with the sent command and if they are identical, a *COMMIT* command can be sent which notifies the low level control modules to execute the command procedures. Commands are sent via the function call 4.2 described in Listing 4.5 while wrapped in another function with the purpose of repeatedly checking if the commands are sent correctly, and repeating transfers if not. The function call 5.3 procedure can be seen in Listing 5.2.

\[\text{private transfer\_loop}(\text{dev, data})\]

A garbage message is constructed which is used to collect full-duplex SPI communication. The low level control modules are configured to always send a buffer of information with each SPI transfer. Which means that any old data that is in the microcontroller buffer, will be sent when new data is sent to the microcontroller for the high level controller. Therefore, whenever a command is sent to the low level control module, another garbage message must be sent in order to receive that last data message that the low level control module received. After the *P1* module has received and executed the *READ_ENC* command, the same startup phase is repeated for *P2* by setting the select signal GPIO pins to a HIGH value. The select signals cause the *P2* module to become the controller and also have communication capabilities. The *READ_ENC* command is then also sent to the module to record initial crank reference positions. Control is then returned to the *P1* module by setting the select signal GPIO pins to LOW. The low level control modules should be ready.
```c
int transfer_loop(int dev, uint16_t DATA){
    uint16_t GARBAGE = 2000; // Garbage message
    uint16_t cmd = 0;
    while(cmd != DATA){
        cmd = SPItx16b(dev, DATA); // send command for 16 bit transfer, ...
        // receive cmd which will be the previous sent message
        cmd = SPItx16b(dev, GARBAGE); // send garbage to check if last ...
        // transfer was success
    }
    return; }
```

Listing 5.2: SPI Transfer Loop

to slow start wing flapping at this point. The function call 5.1 will lastly call the function 5.4 to slowly initialize wing flapping.

5.2 Flight Parameter Control

The set function 5.4, is a public user facing function call that receives two parameters namely, wing and omega. The wing parameter provides the ability to set the flapping frequency of one wing out of the two available wings. The set function is responsible for assuring that not impossible parameter cases occur. For instance, if a Delta value is larger than an Omega value, this is impossible as explained in the Background section. Additionally, the function will determine a Delta value that scales up with the Omega value. If Omega were to increase, but Delta did not, then the Delta bias would increase. It is only when the user calls to adjust Delta, that a bias can be made in the low level controller module. Depending on if the previously set Deltas and Omegas, the Delta bias would either be positive or negative. A sign parameter is also calculated and used to command low level control modules. The actpas parameter determines which low level control module to send the command to. If no actpas parameter is passed, the function 5.4, will determine based on an internal parameter the controller module and communicator module and send the command to the appropriate module. The function finally determines appropriate parameters based on user
```c
int set_short_params(int Wing, int TableSelect, int SpiSelect, uint16_t ... Omega, uint16_t Delta, uint16_t Sign)
{
    uint16_t COMMAND = 2004;
    uint16_t COMMIT = 2002;
    spiDevice = findSpidev(Wing, TableSelect, SpiSelect);
    transferLoop(spiDevice, COMMAND)
    transferLoop(spiDevice, Omega)
    transferLoop(spiDevice, Delta)
    transferLoop(spiDevice, Sign)
    transferLoop(spiDevice, COMMIT)
    return; }
```

Listing 5.3: Send Short Parameter Function

```c
set_Omega(0, 12.56, 0);
set_Omega(0, 12.56, 1);
set_Omega(1, 12.56, 0);
set_Omega(1, 12.56, 1);
```

Listing 5.4: Example code to send frequencies to both controller and communicator modules for both wings

requests calls the function 5.5 which is shown in 5.3.

```
private set_short_params(wing, tableSelect, SpiSelect, Omega, Delta, Sign) (5.5)
```

The function 5.5, simply sends each command or parameter via the function 5.3 to the low level controller module. The Wing parameter determines which wing and therefore which SPI device to use to communicate with the low level module. The parameters TableSelect and SpiSelect are the parameters which, if needed, dynamically adjust the control signals to the correct low level module.

The default Omega parameter upon wing initialization is $4\pi rad/s$. The final initialization phase sets the frequency for both control module and the communication module for both wings. The Listing 5.4 shows Omega frequencies being set for both wings. The Function 5.6,
functions very similarly to the Function 5.4.

\[ \text{public } \text{set}_\text{Delta}(\text{wing}, \text{delta}, \text{actpas}) \quad (5.6) \]

The Function 5.6, ensures that a new \textit{Delta} value is not an impossible value that could cause the low level modules to malfunction. This includes ensuring that \textit{Delta} must be smaller than \textit{Omega}/2 as described in the Background chapter. After the function 5.1 completes, the FW-MAV including its low level controller modules should be completely initialized and ready to perform experiments. Additionally, the Functions 5.7, 5.8, 5.9, and 5.10 are also made available for user access.

\[ \text{public } \text{get}_\text{Omega}(\text{wing}, \text{actpas}) \quad (5.7) \]

\[ \text{public } \text{get}_\text{Delta}(\text{wing}, \text{actpas}) \quad (5.8) \]

The get Functions 5.7 and 5.8 return the current \textit{Omega} or \textit{Delta} that is set for the requested wing control module via the \textit{actpas} parameter which is either a controller or a communicator. If no reference control module is requested via the \textit{actpas} parameter, then the current controller module \textit{Omega} or \textit{Delta} value is returned.

\[ \text{public } \text{set}_\text{Direction}(\text{dir}, \text{wing}) \quad (5.9) \]

The Function 5.9, will send a toggle command to the current controller to invert the rotation of actuator rotation. This in turn reverses the direction of movement or force production for that particular wing.

\[ \text{public } \text{switch}_\text{Control}() \quad (5.10) \]

The Function 5.10, will toggle the control select signals which determine which low level controller module is controller or communicator, effectively reversing their roles.
5.3 Flapping Pattern Control

The final set of functionality that the API provides which is the interface to transfer a new commutation table to a low level module. The Function call 5.11 first allocates the required memory for the new table, then calls the Function 5.11.

\[
\text{public set\_Table(table, size, wing)} \quad (5.11)
\]

\[
\text{private send\_commutation(wing)} \quad (5.12)
\]

The Function 5.12, ensures a critical procedure which is required for the uninterrupted active operation of the FW-MAV low level controller modules. A commutation table is a 1024 element array of 32 bit unsigned integers. It has been established that the word size is 16bit, therefore one 32bit unsigned integer will be split into two sections and each sent as an individual message. Additionally, as shown in Listing 5.3, a message transfer requires a command, data, and commit parameter to be passed in order for the protocol to function. Each 5.3 call requires two message passes to confirm correctness. The Function 5.12, configures the control signals so that the controller module has no communication capability. A commutation table can therefore be transferred to the communicator module without interrupting the controller module. Once the transfer is complete, the communicator module contains the new and up to date commutation table and is ready to switch to become the controlling module. The Function call 5.12, calls the Function 5.13 which can be seen in Listing 5.5.

\[
\text{private set\_long\_params(wing, tableselect, spiselect, table[])} \quad (5.13)
\]

The Function 5.13, first notifies the low level control layer that it intends to transmit a commutation table. The low level control layer then prepares for the routines of receiving array elements and index values. The Function 5.13, then splits a 32bit value from the 1024 element array into two
```c
int set_long_params(int Wing, int Tab1Sel, int SpiSel, uint32_t* Table){
    int i = 0;
    for (i = 0; i < 1024; ++i) {
        transfer_loop(spiDevice, COMMAND)
        //split the 32bit value in 2 16 bit values to send.
        uint16_t temp = 0;
        int j = 0;
        for(j = 0; j < 2; j++) {
            temp = (uint16_t) (((int)(Table[i]) >> (16 * j));
            transfer_loop(spiDevice, temp)
        }
        transfer_loop(spiDevice, (uint16_t)i) \ send index
    }
    return; }
```

Listing 5.5: Send Long Parameter Function

16bit values and passes each via the Function 5.3, and thereafter passes an index value. After the completion of a commutation table transfer. A user can then call the Function 5.10, which will then toggle the control signal commands which switches wing actuator control to the communicator module (making it the controller module).

5.4 Closing the Loop

5.4.1 Power Management

Both the low and high level control modules function with a nominal DC voltage input of 3.3V\(^1\). The Gumstix Overo COM recommends a voltage regulator design which receives 5V and regulates that to 3.3V. The Traco TSR 1-2450\(^2\) DC/DC step-down converter is implemented in this work to take convert unregulated battery voltage and drop it to a stable 5V, however any voltage regulator will suffice. Each actuator requires a nominal voltage of 6V\(^3\). The UBEC DC/DC Step-Down\(^4\) converter is selected as the regulator for each wing actuator.

\(^1\)http://media.gumstix.com/docs/gs3703fy.pdf  
\(^2\)https://cdn-shop.adafruit.com/datasheets/tsr1.pdf  
\(^3\)http://www.micromo.com/media/pdfs/1028_B_FMM.pdf  
\(^4\)https://www.adafruit.com/products/1385
Due to the high cost of each actuator, the manufacturer recommends to never exceed the nominal voltage specifications of the Micromo Series 1028 006B brushless servomotor to ensure that the actuators are not damaged. The UBEC step-down converter ensures this by providing a stable 5V signal which is adequate to power the actuators. The Figure, 5.1 shows a PCB that provides battery connectors as well as the above mentioned components that regulate the power supplies for the FW-MAV. The red box contains the two UBEC step-down converters, one for each motor. The yellow box shows the Traco step-down converter for the 5.5V production. The purple boxes show additional safety circuitry intended for thermal protection.

Early in this project development, it was assumed that the 3D printed plastic body of the FW-MAV might not be able to withstand the temperatures generated from FW-MAV actuators and linkage friction. Thermal couplers combined with voltage dividers detect high temperatures on the FW-MAV. If a high enough temperature is detected the a kill-switch is thrown which cuts all power to the FW-MAV. After initial testing, it was observed that the actuators did not get warm enough to melt the plastic nor was the friction generating enough heat to cause damage. Therefore no testing was performed on the thermal safeties. The designs are set in place for future implementation with alternate actuators or FW-MAV plastics.
The Gumstix Overo COM logic level input and output signals function on a 1.8V DC voltage while the low level control modules function on 3.3V DC voltage. In order to interface the command signals that determine the control and communication modules in the low level controller, signals must be amplified or stepped up. The Bi-Directional SparkFun Logic Level Converter is implemented to convert the control select signals to and from the high and low level controllers respectively. The logic level converter can be seen in the green box in Figure 5.1.

### 5.4.2 Testing Environment

The Figure 5.2, shows the experimentation environment. The surface upon which the FW-MAV propels itself is water. The FW-MAV puck is capable of floating on water which constrains the vehicle vertically. Tracking cameras are attached to the top of the frame which continually locate the vehicle coordinates and forward them to a user algorithm. The Figure 5.3, shows colour tracking markers that are used to determine the center, front, and side of the FW-MAV. The tracking algorithm determines the location and forward facing direction based on the location markers. The user algorithm that captures location data as well as computes new control parameters interfaces with the Gumstix Overo COM high level controller via wifi. All API functions discussed in Section 5 are wrapped intro an application written in Python which connects over SSH to the high level controller. The application is responsible for maintaining closed-loop control by continually collecting location feedback and sending control commands through the control flow. In order to start working with the application a user must run the client application on the high level controller, then simply has to call the Function 5.14:

\[
\text{init()}
\]  

\[5.14\]  

---


6[https://www.sparkfun.com/products/12009](https://www.sparkfun.com/products/12009)
The Function 5.14 will create a secure shell connection to the Gumstix Overo COM perform all required initialization. Finally, the application provides a programmatic interface to collect tracking data, command FW-MAV frequency, delta bias, flapping direction, and flapping pattern tables via the following Functions:

\[ \text{collectTrack}() \]  
\[ \text{sendParams}(lDelta, rDelta, lOmega, rOmega) \]
changeDirection(wing) \hspace{2cm} (5.17)

sendTable(table[], wing) \hspace{2cm} (5.18)

enableTable() \hspace{2cm} (5.19)
Conclusion

There are 3 components that have been created and tested to work that complete this research platform as well as this thesis and are listed as follows:

• A new redesigned low level hardware control layer which contains design implementations from previous work as well as a new high level controller that interfaces with the low level controller and supports in closed-loop FW-MAV experimentation. The low level control layer introduces additional microcontrollers which alleviate memory and performance constraints. The constraints are alleviated by each microcontroller assuming a dedicated task of motor control and data communication respectively. The high level controller replaces existing wireless communication hardware and also supports more advanced computational features. The high level controller features wireless communication capabilities as well as user level control over flight control parameters and low level control tasks.

• A software library and API, which is accessible through the high level controller and is enables interfacing with the low level hardware control layer. An interfacing library which performs flight control parameter conversion as well as coordinating communication in the background between the high level controller and low level controller layer. The API provides simplified access to FW-MAV flight control parameters such as Omega and Delta and additionally enables the capability to control wing flapping patterns. The API and library perform transitional tasks for data transmission to and from low and high level controllers and keep track of flight control variables for vehicle safety.
• A testing environment which closes control and learning loops by providing tracking solutions and a remote access application. Tracking cameras and software are implemented which continually locate a FW-MAV on a flat two dimensional surface and transmit the data to a user algorithm over a network connection. The remote access application uses secure sockets to transmit user API calls to and from high level controller and user algorithm. Finally, the testing environment supports a floating puck in enclosed surface which contains water. The FW-MAV sits on the puck while floating on water, simulating hover.

6.1 Future Work

When FW-MAVs experience in flight failure or damage, their flight characteristics deviate from expected models in static controllers. Researchers want to apply dynamic control algorithms to correct the model behavior. Existing control platforms provided limited control over flight characteristics. Future work intends to apply adaptive control algorithms on FW-MAV wing gait patterns and this research platform provides that support. A genetic algorithm can be used in order to adapt, mutate, and learn new flapping wing patterns. A genome can represent two wing flapping patterns, one for each wing. Then various mutation functions can be applied which alter the wing gaits, which can then be passed to a FW-MAV via the platform and finally be tested.

6.2 Alternative Solutions

There are 2 different methods of attack for this problem.

• The first is a complete redesign of hardware and porting of software. A more powerful and feature rich microprocessor(such as an ARM Cortex M*) can provide the capabilities required to actuate multiple motors, generate new flapping patterns, and interface wirelessly. The microcontroller can replace the low and high level controllers as an all-in-one package. This would be the most effective course of action but also the most time consuming and
incompatible. If new flight control methods are developed by research groups designed for the original microchip microcontrollers mentioned in the Background section, then extensive porting and debug testing would be required.

- Second is to introduce new microcontrollers that are designed for the purpose of actuating PWM driven motors. The microcontrollers would be selected with memory capacity necessary for minimum as well as future capabilities. In this case some rewrite or porting of flight control code would be required for pin compatibility. Potentially creating a library which abstracts pins and channels could reduce future need to port newly developed flight control methods.

In each case it is important to include support to synchronize wing flaps. This can be done in a number of ways, one of which is by keeping a channel between each microcontroller open to regulate the start of a wing flap from a dominant microcontroller. An alternative method is to keep a channel open between the two low level wing controllers and the high level controller and to determine correct flapping frequencies.

Finally if this research platform is kept intact, then some alterations and improvements can be made. The SPI bus does not have to be passed through a multiplexer. SPI chip select pins are routed from the high level controller to the low level controller in this platform already. The chip select signal can be used to determine which microcontroller will be communicated to, the SPI data channels can therefore be directly routed to each microcontroller. This is important since the multiplexer potentially adds propagation delays which cause timer inconsistencies during SPI communication. Weight can also be reduced by removing the multiplexer ICs. To further save space and reduce potential noise issues, the multiplexers for motor control and feedback can be removed. The control signals from the high level controller can be used to simply notify each microcontroller of their state and to stop being in control. A small hand off algorithm can perform the transition.
Bibliography


