

University of Nevada, Reno

A Spherical Aerial Terrestrial Robot

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

by

Christopher J. Dudley

Dr. Kam K. Leang/Thesis Advisor
August 2014



THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

CHRISTOPHER J. DUDLEY

entitled

A Spherical Aerial Terrestrial Robot

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Kam K. Leang, Ph.D., Advisor

Eric Wang, Ph.D., Committee Member

David Feil-Seifer, Ph.D., Graduate School Representative

Marsha H. Read, Ph.D., Associate Dean, Graduate School

August 2014

Abstract

This thesis focuses on the design of a novel, ultra-lightweight spherical aerial terrestrial robot (ATR). The ATR has the ability to fly through the air or roll on the ground, for applications that include search and rescue, mapping, surveillance, environmental sensing, and entertainment. The design centers around a micro-quadcopter encased in a lightweight spherical exoskeleton that can rotate about the quadcopter. The spherical exoskeleton offers agile ground locomotion while maintaining characteristics of a basic aerial robot in flying mode. A model of the system dynamics for both modes of locomotion is presented and utilized in simulations to generate potential trajectories for aerial and terrestrial locomotion. Details of the quadcopter and exoskeleton design and fabrication are discussed, including the robot's turning characteristic over ground and the spring-steel exoskeleton with carbon fiber axle. The capabilities of the ATR are experimentally tested and are in good agreement with model-simulated performance. An energy analysis is presented to validate the overall efficiency of the robot in both modes of locomotion. Experimentally-supported estimates show that the ATR can roll along the ground for over 12 minutes and cover the distance of 1.7 km, or it can fly for 4.82 minutes and travel 469 m, on a single 350 mAh battery. Compared to a traditional flying-only robot, the ATR traveling over the same distance in rolling mode is 2.63-times more efficient, and in flying mode the system is only 39 percent less efficient. Experimental results also demonstrate the ATR's transition from rolling to flying mode.

Acknowledgments

During my time at the University of Nevada, Reno I have learned plentifully, performed intriguing research, and have met extraordinary colleagues and friends. I am grateful to everyone who has contributed to my success. Specifically, I would first like to thank my advisor, Kam Leang, for his continued support for my endeavours. I have enjoyed the projects we worked on together and his guidance, and support through this process. Kam is a dedicated advisor, a talented researcher, and a good person and I am lucky to have had him as a mentor.

I express my thanks to my thesis committee members, Dr. Eric Wang and Dr. David Feil-Seifer, for taking the time to serve on my committee and providing guidance on my thesis work.

To my lab-mates, Alex Woods, Jim Carrico, Max Fleming, and Brian Kenton, and my friend Robby Liebherr. Thank you for your insights into my research, your efforts to cheer me up after a failed experiment or high-speed encounter with the road, and most of all, your friendship. I wish you all the best in your future endeavours. To Gary Erickson, the staff, and my teammates on Team Clif Bar Cycling, thank you for your support both on and off the bike, you've always kept me grounded and focused towards my goals.

I am indebted to my family for the love and support they have given me over the past 25 years. My father coerced me into the garage at a young age and gave me a head start in engineering for which I am forever thankful. My parents gave me every opportunity and encouraged me along the way — I cant thank them enough for everything they have done for me.

Last but certainly not least, I thank my girlfriend Amanda who always believes in me and always makes me smile. Through late nights in the lab and weekends away bike racing, you have stood by my side.

Dedication

To my parents, John and Kathryn Dudley.

Contents

1	Thesis Goal, Objectives, and Contribution	1
1.1	Introduction	1
1.2	Contribution	2
1.3	Organization	3
2	Background	4
2.1	Unmanned Autonomous Systems (UAS)	4
2.1.1	Autonomy	5
2.2	Unmanned Ground Vehicles (UGV)	5
2.3	Unmanned Aerial Vehicles (UAV)	6
2.3.1	Fixed-Wing vs. Rotor-Driven Aircraft	9
2.3.2	Aircraft Size	9
2.3.3	Summary of UAS	10
2.4	Quadcopter Aerial Robots	10
2.4.1	Quadcopter Challenges	11
2.5	Review of Aerial-Terrestrial Hybrid Robotics	12
2.6	Summary	14
3	ATR System Design and Fabrication	15
3.1	Quadcopter Platform Design	17

3.2	Exoskeleton Design	19
3.3	Fabrication	21
3.4	Control Hardware and Software	25
3.4.1	Quadcopter Electronics	25
3.4.2	Wireless Communication	26
3.4.3	Inertial Measurement Unit	26
3.4.4	Digital PID Control	27
3.5	Summary	28
4	System Modeling	30
4.1	Reference Frames	30
4.2	Coordinate Transformation	32
4.2.1	Euler Rotation Matrix	32
4.2.2	Quaternion Transformation	34
4.3	Vector Derivatives in Two Reference Frames	36
4.4	Newton's Second Law in Body Coordinates	36
4.5	Euler's Second Law in Body Coordinates	37
4.6	Newton-Euler Equations	38
4.7	Aerial Locomotion	38
4.8	Terrestrial Locomotion	41
4.9	Summary	48
5	Prototype Characterization	49
5.1	Motor Characterization	49
5.2	Physical System Characterization	51
5.3	Controller Performance	52

6	Simulation	54
6.1	Summary	56
7	Experimental Results and Discussion	58
7.1	Autonomous Hover	58
7.2	Open Loop Aerial Trajectory Tracking	60
7.2.1	Terrestrial Locomotion	63
7.3	Energy Analysis	65
7.4	Summary	66
8	Conclusions and Future Work	67
8.1	Conclusions	67
8.2	Future Work	68
A	Motor Characterization	73
B	MATLAB .m Files	75
B.1	GUI	75
B.2	Simulation Videos	84
B.2.1	Create Patches	84
B.2.2	Animation	86
C	Simulink Block Diagram	91
C.1	System Block Diagram	91
D	ATR Control Code	95

List of Figures

1.1	Concept of the aerial terrestrial robot (ATR). The robot can be hand launched and operate in either flying or rolling mode. Rolling mode is convenient for energy efficient locomotion and maneuvering through tight spaces and challenging terrain.	2
2.1	State of the art in UGV and UAV systems. a) Oshkosh TerraMax UGV capable of waypoint, lead, and follow tracking. b) Prototype Google UGV for civilian use. c) Plug-and-play UGV TerraMax unit for implementation in any tactical vehicle for military use. d) Predator UAV by General Atomics, Inc. engaging a target. e) Insitu ScanEagle surveillance UAV. f) AeroVironment hand-launchable Raven UAV for surveillance applications.	8
2.2	Various size UAVs and their applications such as military target engagement, surveillance, mapping, and entertainment with respective masses ranging from 15,000 lb to 1.5 lb.	10
2.3	Commercially available quadcopter from 3D Robotics with onboard GPS, sonar sensor, barometer, and external video camera.	11

2.4	Quadcopter system and actuation principles given by a) thrust and moment from angular velocity of actuators. b) Roll action is achieved by increasing the angular velocity of motor 4 while decreasing the angular velocity of motor 3. c) Pitch action is achieved by increasing the angular velocity of motor 2 while decreasing the angular velocity of motor 1. d) Yaw action is achieved by increasing the angular velocity of motor pair 1 and 2 while decreasing that of motor pair 3 and 4. . .	12
2.5	Existing hybrid aerial-terrestrial designs. a) Micro air-land vehicle, b) hexapedal flapping winged robot, c)ultra-light jumping and gliding robot d) HYTAQ robot.	13
3.1	Spherical aerial terrestrial robot: (a)key components, (b)rolling mode, and (c)perching mode.	16
3.2	Micro quadcopter used to create the ATR.	17
3.3	Spherical exoskeleton design: (a) mode of operation, (b) design geometry, and (c) turn radius corresponding to d_1 , d_2 , d_3 for varying γ . . .	20
3.4	Fabrication of the spherical exoskeleton: (a) completed half sphere on polypropylene mold, (b) structural solder joining the structure, (c) combining of completed half spheres, and (d) completed ATR with carbon fiber axle, low-friction Delrin journal and micro-quadcopter. .	22
3.5	Exoskeleton design iterations: (a)3D printed PLA, (b)lightweight 3D printed PLA (c)carbon fiber composite , (d)titanium spring wire (e)steel welding wire 0.30 in diameter, and (f)0.20 in spring steel wire (music wire).	24
3.6	Circuit diagram for the micro-quadcopter platform with Wireless XBee radio	25

3.7	Custom designed graphic user interface with options for joystick input and autonomous commands. The ground control station communicates with the ATR at 50 Hz via wireless 2.4 GHz XBee radio.	27
3.8	Attitude controller design for stabilization of the robot orientation, ϕ , θ , ψ	28
4.1	Rigid body in space with body-fixed accelerating reference frame B . An additional reference frame, E , is defined and is inertial. The velocity of the body can be expressed in both reference frames B , and E , such that a transformation between B and E exists for all orientations.	31
4.2	Experimental micro-quadcopter platform with inertial reference frame E , defined by \hat{e}_x , Northward, \hat{e}_y , Eastward, and \hat{e}_z , directed to the center of the earth. The quadcopter body frame is defined by \hat{b}_x , forward, \hat{b}_y pointed right, and \hat{b}_z downward.	39
4.3	The ATR system in terrestrial locomotion mode, powered by the brushed motors propelling and directing the rolling exoskeleton. The ATR is capable of smooth turning by positioning itself onto a set of its rings and rolling in a circular pattern.	42
4.4	The ATR system in perching mode. The system can achieve passive stability by rolling onto the flat surfaces located at the ends of the robot.	48
5.1	Motor thrust characterization test stand with tachometer, power meter, gram scale for thrust measurement, and serial communication link for data acquisition and throttle commands to the platform.	50
5.2	a) Motor thrust and b) moment constant characterization with second order polynomial fit.	51

5.3	Measured system response due to: a) Step response and b) input tracking. The solid green line represents the desired pitch angle, dashed line represents the simulated model response, and the solid black line represents the experimental response on 1-DOF test stand.	53
6.1	Simulation block diagram.	54
6.2	Simulation of the ATR rolling on the ground and transitioning to aerial mode.	55
6.3	Simulation of the ATR performing a vertical take-off, flying forward, correcting its angle to minimize its velocity, and rolling on the ground after landing.	56
6.4	Simulation of the ATR rolling on the ground to maneuver through obstacles and then transitioning to flying mode through a small opening.	57
7.1	The ATR can be thrown into the air and self-stabilize to hover. The duration of this test is approximately 14 seconds.	59
7.2	Open loop trajectory tracking for a desired square pattern.	61
7.3	Flying behavior of the ATR into a tube, then rolling through the tube and out.	62
7.4	Open loop experimental results for a desired roll, stop, fly trajectory.	63
7.5	Open loop experimental results for a desired roll and turn maneuver	65

List of Tables

3.1	Micro-quadcopter specifications	18
5.1	ATR system parameters	52

Chapter 1

Thesis Goal, Objectives, and Contribution

1.1 Introduction

Aerial robots that can hover and maneuver quickly and accurately in tight urban and indoor spaces are well suited for applications that include search and rescue [1]–[3], mapping [4], [5], surveillance [6], environmental sensing [7], [8] and disaster remediation [9]. Interest in aerial robots has grown at a rapid pace, but relatively short flight time, limited control of maneuvers, self-localization, sensing, and end user safety pose significant challenges [10]. This thesis presents the development of a new hybrid robot that combines aerial and terrestrial (hybrid) locomotion to address the challenges of efficiency and limited functionality of ground and aerial vehicles. The objectives of this thesis are to design, build, and model an ultra-small spherical ATR, then utilize the model to generate control parameters and inputs to demonstrate hybrid locomotion. Herein, the design, modeling, simulation, characterization, and control of a new spherical aerial terrestrial robot (ATR) is presented.

The ATR consists of an airborne (e.g., multi-rotor) platform encased in a spherical exoskeleton which can be exploited for energy efficient rolling without the use of additional actuators, preserving the system’s mechanical simplicity. Figure 1.1 shows the ATR and the various modes of locomotion. The developed ATR is smaller than

6 inches in diameter, thus the user can easily hand launch the robot like a ball into flying mode, and depending on the situation, the robot can fly, or enter rolling mode to traverse over the ground surface or through pipes and air ducts.

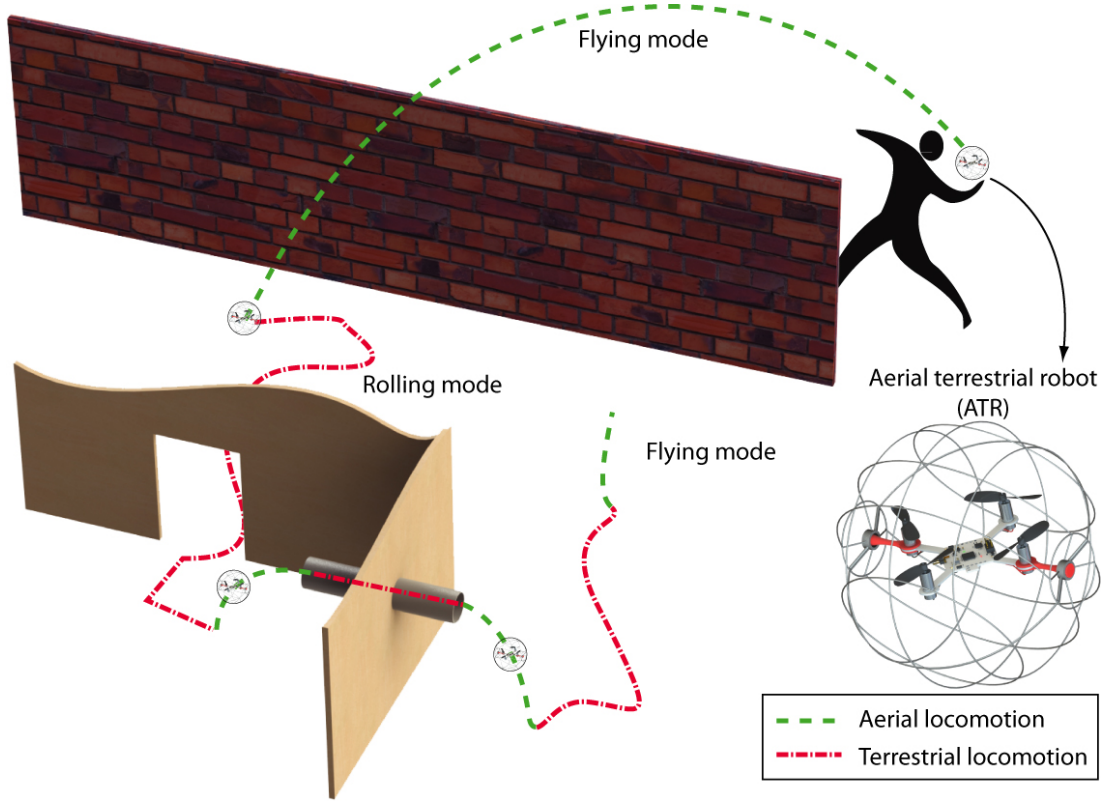


Figure 1.1: Concept of the aerial terrestrial robot (ATR). The robot can be hand launched and operate in either flying or rolling mode. Rolling mode is convenient for energy efficient locomotion and maneuvering through tight spaces and challenging terrain.

1.2 Contribution

The contribution of this work is the design, modeling, simulation, trajectory generation, fabrication, and demonstration of an aerial terrestrial robot. Hybrid modes of locomotion enable the ATR to negotiate challenging obstacles and terrain while maintaining the benefits of each respective mode of locomotion. In addition to yawing

about the ground contact point, a novelty of the ATR is its capability to turn while rolling in a method similar to how a railcar navigates curves in its tracks. The ATR can position itself on varying diameter rings of its exoskeleton in contact with the ground while rolling, causing the platform to turn in a circular manner which allows the ATR to follow complex curved ground trajectories. Compared to flying, rolling mode can be used to avoid detection, and since the ATR does not need to support its own weight in rolling mode, it can be more energy efficient, thus preserving battery power.

1.3 Organization

This thesis is organized as follows. Chapter 2 presents a background of existing unmanned autonomous systems including aerial, terrestrial, and hybrid locomotive systems. Chapter 3 presents the details of the ATR system design, along with the specification and fabrication of the ATR. Chapter 4 presents a model of the experimental system and Chapter 5 presents the characterization of the experimental system. Specifically, the synthesized controller performance is evaluated, motor parameters are measured, and physical system parameters are evaluated. Chapter 6 presents a simulation of the prototype ATR based on the formulated model and system characterization. Chapter 7 presents autonomous aerial and terrestrial locomotion results and discussion. Finally, conclusions are presented in Chapter 8.

Chapter 2

Background

This chapter serves to provide a background on unmanned autonomous systems and their applications in society. Section 2.1 explores the various unmanned autonomous systems, and their applications to modern-day society are discussed in Sections 2.2 and 2.3. Section 2.4 gives the history of the quadcopter aerial robot, as well as quadcopter basic principles of operation and accompanying challenges. Finally, Section 2.5 explains aerial-terrestrial hybrid robots and reviews previous hybrid aerial-terrestrial robotic systems.

2.1 Unmanned Autonomous Systems (UAS)

Research in unmanned robots has increased rapidly since the invention of piloted aircraft and machinery. By eliminating a human pilot from the system, unmanned autonomous systems (UAS) can perform tasks that are possibly dangerous, undesirable, or difficult for a human operator to perform. Additionally, since UAS do not require a human pilot, designers are able to reduce system mass, size, and complexity, thereby increasing overall efficiency. UAS are classified by their operating medium. For example, autonomous systems that operate on the ground are defined as unmanned ground vehicles (UGV), systems that fly as unmanned aerial vehicles (UAV), and systems that operate in aqueous environments are classified as autonomous un-

derwater vehicles (AUV). Unmanned autonomous systems, when applied to industries such as agriculture, transportation, homeland security, and defense, can provide significant contributions to society.

2.1.1 Autonomy

An important differentiation between unmanned autonomous systems and traditional hobby-like radio controlled systems is autonomy. That is, UAS are capable of executing tasks independently without user input, whereas radio controlled systems have the obvious restriction that a remote pilot must control all aspects of radio controlled system operation. Unmanned autonomous systems drastically reduce the need for user input and can allow a single operator to control a fleet of UAS with high level guidance, increasing the system throughput.

2.2 Unmanned Ground Vehicles (UGV)

Unmanned ground vehicles (UGV) are a classification of UAS that operate on the ground in applications that may be dangerous, require stealth and small form factor, or endurance that can fatigue a human operator. Recently, the U.S. Armed Forces announced a goal to have one third of its operational ground combat vehicles unmanned by 2015, freeing up valuable human resources and reducing the inherent risk of humans in the battlefield. One such example of unmanned ground combat vehicles is the TerraMax UGV by Oshkosh Defense, shown in Fig. 2.1(a), which can be controlled remotely or programmed to lead or follow a manned vehicle in combat scenarios [11]. The same company has developed a plug-and-play system, shown in Fig. 2.1(c) that can transform traditional tactical vehicles into UGVs. Civilian applications for UGVs include autonomous automobiles, a field that began in the

1980's when Carnegie Mellon University presented its Navlab vehicles that operated autonomously in structured environments [12] and when the University of the Bundeswehr Munich showed early results in high-speed motorway driving [13]. Recently, tech giant Google has developed vehicles that can drive themselves. The vehicles, shown in Fig. 2.1(b) use artificial-intelligence software that can sense anything near the car and mimic the decisions made by a human driver [14]. Autonomous controlled vehicles react faster than humans, have 360-degree perception and do not get distracted, sleepy or intoxicated. This technology could drastically increase the capacity of roads by allowing cars to drive safely while closer together, and because the robot cars would eventually be less likely to crash, they could be built lighter, thus reducing fuel consumption.

2.3 Unmanned Aerial Vehicles (UAV)

Currently, general applications of unmanned aerial vehicles (UAV) utilizing existing aircraft include freight, human transportation, surveillance, and military applications. Unmanned aerial vehicles have the additional potential of impacting society in fields such as agriculture, law enforcement, mapping, and logistics due to their highly configurable nature. During the past decade, UAV development has increased rapidly and has resulted in fully controllable systems capable of surveillance and various military applications. One such example of this type of UAV is the Predator by General Atomics, shown in Fig. 2.1(d), which can be controlled from a remote ground control station (GCS) to identify potential targets and engage a target of interest with a laser guided missile [15]. The recently developed ScanEagle by Instil, shown in Fig. 2.1(e), is a smaller example of commercially available UAV that is capable of automatically tracking a target and relaying encrypted digital video and command and control datalinks over 55 nautical miles to a GCS [16]. The ScanEagle

has been continuously deployed since 2005 in land and maritime environments with a 99 percent mission success rate. AeroVironment has developed a small 4.5 foot wingspan, hand-launchable UAS called the Raven, that can be operated manually or pre-programmed for an autonomous surveillance mission via GPS waypoints [17]. The Raven, shown in Fig. 2.1(f), is the most widely operated UAV in the world today. Robust technology, longer operating life, and enhanced sensing capabilities have enabled a greater use of small UAVs. In addition, improved communication technologies has allowed operation of UAVs from far away ground control stations. Advanced systems have expanded military goals to include possibilities such as UAVs delivering cargo and supplies to soldiers on the battlefield in remote locations or performing complex search and rescue missions.



Figure 2.1: State of the art in UGV and UAV systems. a) Oshkosh TerraMax UGV capable of waypoint, lead, and follow tracking. b) Prototype Google UGV for civilian use. c) Plug-and-play UGV TerraMax unit for implementation in any tactical vehicle for military use. d) Predator UAV by General Atomics, Inc. engaging a target. e) Insitu ScanEagle surveillance UAV. f) AeroVironment hand-launchable Raven UAV for surveillance applications.

2.3.1 Fixed-Wing vs. Rotor-Driven Aircraft

Although the broad category of UAS are highly capable, different aircraft structures are better suited for certain applications. One such distinction is between fixed-wing and rotor-driven aircraft. Fixed-wing aircraft generally have one fixed airfoil-like geometry with a separate propulsion actuator that propels the aircraft forward. Airflow over the wings of a fixed-wing aircraft helps create lift that elevates the aircraft. In contrast, rotor-driven aircraft obtain lift by direct actuation of a rotor perpendicular to the earth. In some applications, this distinction is negligible, but rotor-driven aircraft are much more capable in applications that require vertical take-off and landing (VTOL), or the ability to hover in place. Rotor-driven VTOL aircraft, such as multi-rotor helicopters, do not require long runways for take-off or landing and are capable of hovering and flying at low velocities, making them well suited for urban applications.

2.3.2 Aircraft Size

UAS size has a direct impact on the types of tasks that can be performed; larger UAS can travel long distances and have a high payload capacity, but are restricted in their operating environments by their large size. Smaller UAS are more capable of urban tasks, such as indoor navigation, but their small size and limited payload decrease the overall system efficiency. An especially miniature subset of UAS, called micro air vehicles, is defined by systems smaller than 50 cm in wingspan and are capable of maneuvering in very tight spaces such as inspection ducts, damaged buildings, and environmental sensing in inaccessible environments. Due to their small size and flexible capabilities, micro scale UAS are especially attractive in the research community [18].

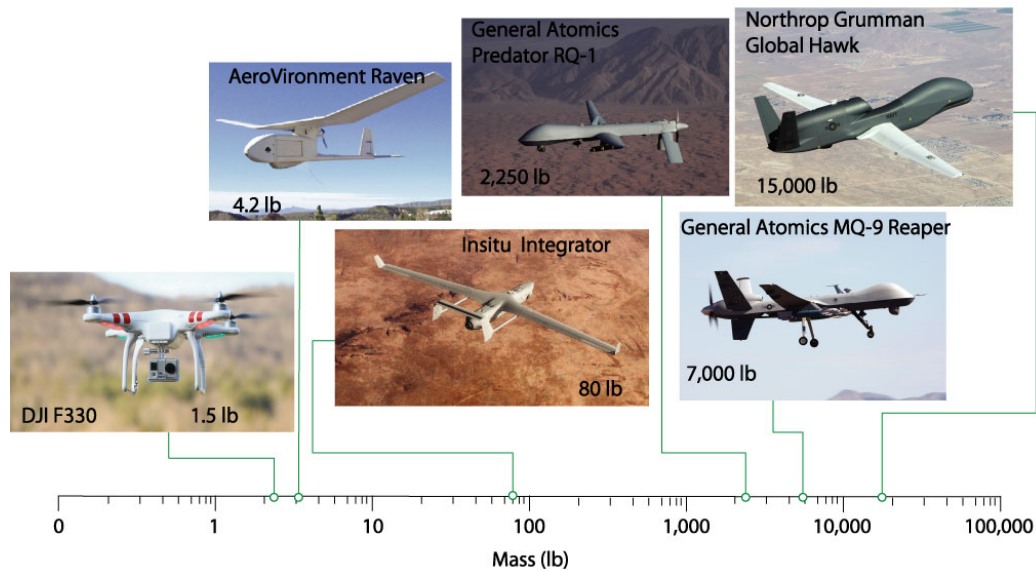


Figure 2.2: Various size UAVs and their applications such as military target engagement, surveillance, mapping, and entertainment with respective masses ranging from 15,000 lb to 1.5 lb.

2.3.3 Summary of UAS

With advances in low-cost, reliable, and safe UAS, there exists a strong desire to commercialize this technology for uses that include monitoring agricultural crop health and harvesting, consumer shipping and delivery services, public transportation, inspection of infrastructure like bridges, pipelines, and roadways, search and rescue, oceanic and atmospheric observation, law enforcement, environmental sensing, and yet unrealized applications. Many commercial industries can benefit highly from UAS for tasks that require high-precision, long duration, or repetitive tasks, freeing up valuable human resources for more advanced contributions in the workplace.

2.4 Quadcopter Aerial Robots

As interest in micro scale UAS increases, quadcopter UAVs have consumed much of the research, and quadcopter capabilities have been compared favorably against other

UAS [19]. Unlike traditional helicopters where a rear rudder is necessary to counteract the main propeller moment, the quadcopter is designed to exhibit no moment at the center of mass during hover, simplifying the system complexity. Additionally, quadcopters are driven by four independent actuators — increasing the force of a single actuator independently enables aggressive and agile maneuvers. For example, three general configurations of motor speeds can influence the roll, pitch, and yaw of the quadcopter as shown in Fig. 2.4. In the simplest of cases, a designer can assume that each rotational degree of freedom is uncoupled, drastically simplifying the dynamics and control required for various autonomous commands. A typical commercially-available quadcopter is shown in Fig. 2.3.



Figure 2.3: Commercially available quadcopter from 3D Robotics with onboard GPS, sonar sensor, barometer, and external video camera.

2.4.1 Quadcopter Challenges

While the quadcopter platform is very attractive due to its relatively low complexity and high capability compared to traditional helicopters, the system does present some challenges. For example, quadcopters are an example of an underactuated system. That is, the system does not contain actuators for every degree of freedom (6-DOF, 4 control inputs). In addition to underactuation, the quadcopter is a dynamically

unstable system and without active control would not be able to fly. Controller design for quadcopter systems is inherently nonlinear and is usually only valid for a small regime of orientation. Even though quadcopter platforms present unique challenges, the combination of vertical take-off and landing, hover capability, and system agility makes quadcopter aerial robots an excellent choice for aerial locomotion.

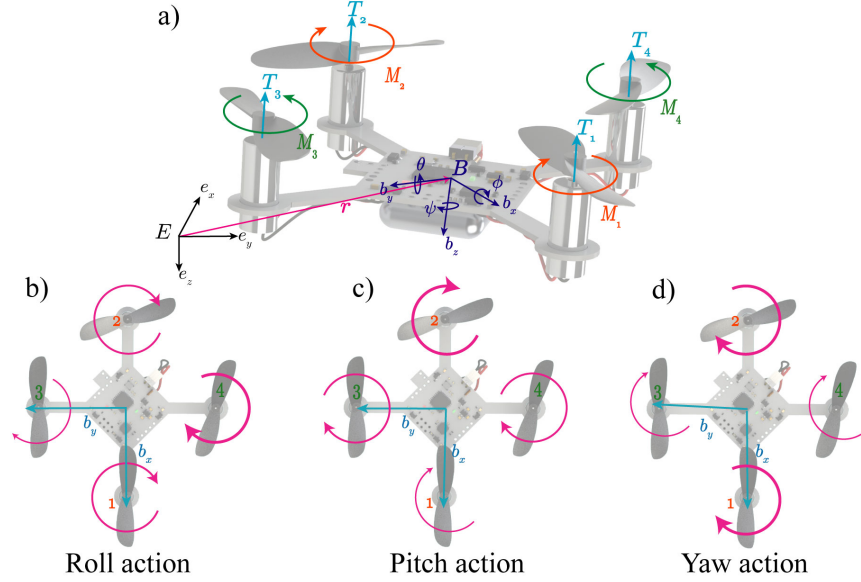


Figure 2.4: Quadcopter system and actuation principles given by a) thrust and moment from angular velocity of actuators. b) Roll action is achieved by increasing the angular velocity of motor 4 while decreasing the angular velocity of motor 3. c) Pitch action is achieved by increasing the angular velocity of motor 2 while decreasing the angular velocity of motor 1. d) Yaw action is achieved by increasing the angular velocity of motor pair 1 and 2 while decreasing that of motor pair 3 and 4.

2.5 Review of Aerial-Terrestrial Hybrid Robotics

Research and development of hybrid aerial terrestrial robotics is limited. Most aerial terrestrial robots are composed of separate aerial and terrestrial actuators that are attached together to form a hybrid system. For example, the micro air-land vehicle (MALV) II shown in Fig. 2.5(a) is a fixed wing propeller driven aerial vehicle

with attached DC motor wheel-leg drive system for terrestrial locomotion [20, 21]. A hexapedal winged robot equipped with flapping wings, shown in Fig. 2.5(b), has been developed to increase the overall running speed of a terrestrial robot [22], with future possibility of a fully functional flying and crawling robot. An ultra-light jumping and gliding robot capable of jumping 27 times its own height, shown in Fig. 2.5(c), has been developed to mimic a desert locust [23].

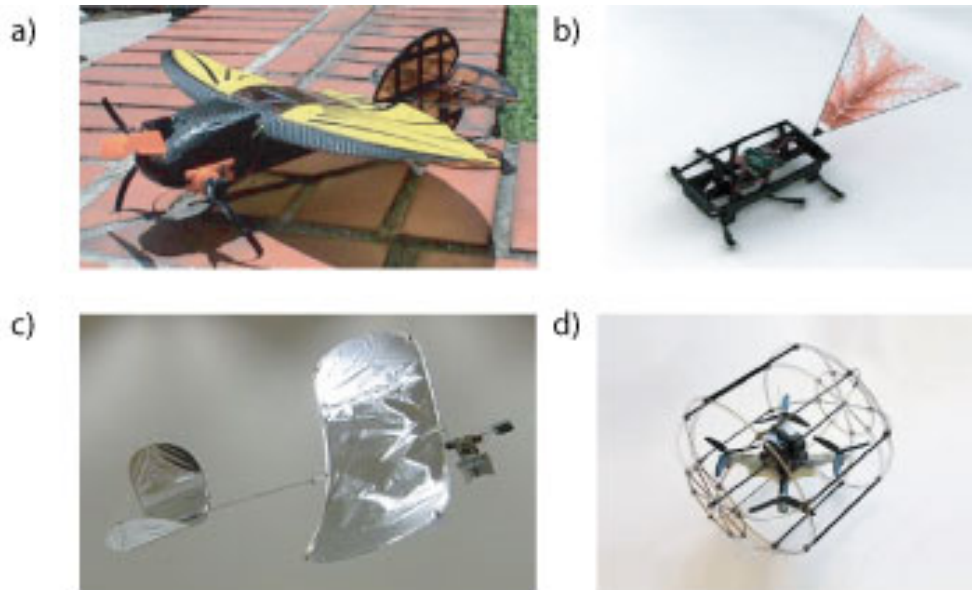


Figure 2.5: Existing hybrid aerial-terrestrial designs. a) Micro air-land vehicle, b) hexapedal flapping winged robot, c) ultra-light jumping and gliding robot d) HYTAQ robot.

Recently, a hybrid quadcopter system was developed and translates downward thrust from the propellers into forward walking motion of legs, but the speed of the design was limited by the slow-acting shape memory alloy actuators [24]. The same group developed a novel cylindrical quadcopter-based aerial-terrestrial robot [25], shown in Fig. 2.5(d), that is capable of both efficient ground and aerial locomotion. The novel cylindrical exoskeleton keeps the system stable during ground locomotion, but the robot is unable to access tight spaces such as pipes and air ducts due to its geometry, turning method, and size.

2.6 Summary

This chapter introduced unmanned autonomous systems and their applications to modern-day society, reviewed the distinction between various UAS and their applications, introduced quadcopter aerial robots, addressed quadcopter challenges, and reviewed previous aerial-terrestrial robotic systems. The following chapter focuses on the design of a novel spherical aerial-terrestrial robot.

Chapter 3

ATR System Design and Fabrication

For urban applications, the ATR is designed to be as small and lightweight as possible. An advantage of a small, ultra-lightweight platform is the ability to access tight spaces such as pipes and air ducts for applications such as inspection, surveillance, mapping, and search and rescue. The ATR's spherical exoskeleton geometry enables the robot to perform turns of varying radii while simultaneously rolling over the ground, expanding the range of possible trajectories over previous designs that are only able to roll straight and pivot-turn [25]. While the novel spherical design enhances mobility, it is also accompanied by numerous challenges, namely stability.

The ATR is designed so that the robot is capable of righting itself from any orientation. The robot can achieve passive stability by positioning itself on the flat surfaces located on the sides of the exoskeleton. The spherical exoskeleton is designed to offer a variety of turning radii for advanced trajectories. An additional challenge in the design of the ATR is the limited payload of the inner flying platform. The ATR's spherical exoskeleton is designed to be as light as possible while protecting the flying platform from impact and providing sufficient rigidity for efficient ground

locomotion.

The basic design of the ATR is shown Figure 3.1(a). The system consists of a hover-capable flying robotic platform surrounded by an outer lightweight spherical exoskeleton and rotating support axle. The ATR's exoskeleton, when in terrestrial mode [Fig. 3.1(b)], rotates about the platform's central axis on its outer exoskeleton. The flat surface on either end of the axle creates a perching surface, as illustrated in Fig. 3.1(c). By orienting the ATR so that it sits on either side, the platform is able to rest in a stable position.

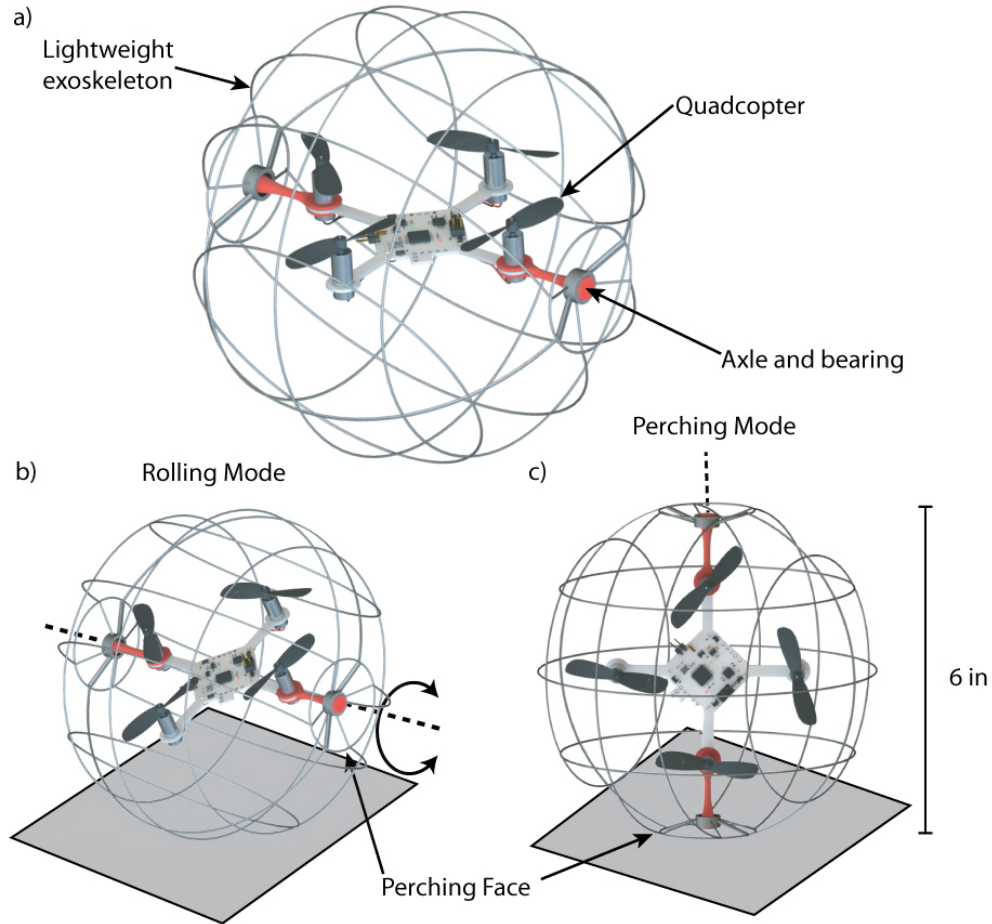


Figure 3.1: Spherical aerial terrestrial robot: (a)key components, (b)rolling mode, and (c)perching mode.

3.1 Quadcopter Platform Design

The quadcopter platform, shown in Fig. 3.2, used to create the ATR has a footprint of approximately 135 mm by 135 mm, measured along the platform arms from one rotor tip to the opposite rotor tip. The quadcopter and battery pack (1S1P 350 mAh lithium-polymer battery) weigh 28.7 g combined, and at full throttle, the quadcopter can lift an additional 16.9 g. The complete ATR concept, which consists of the quadcopter, axle, bearings, and lightweight steel exoskeleton, has a combined weight of 35.24 g allowing for a payload capacity of 10.36 g. Additional payload capacity can be achieved by design optimization, minimizing the weight of components, and increasing the number or size of motors and rotors.

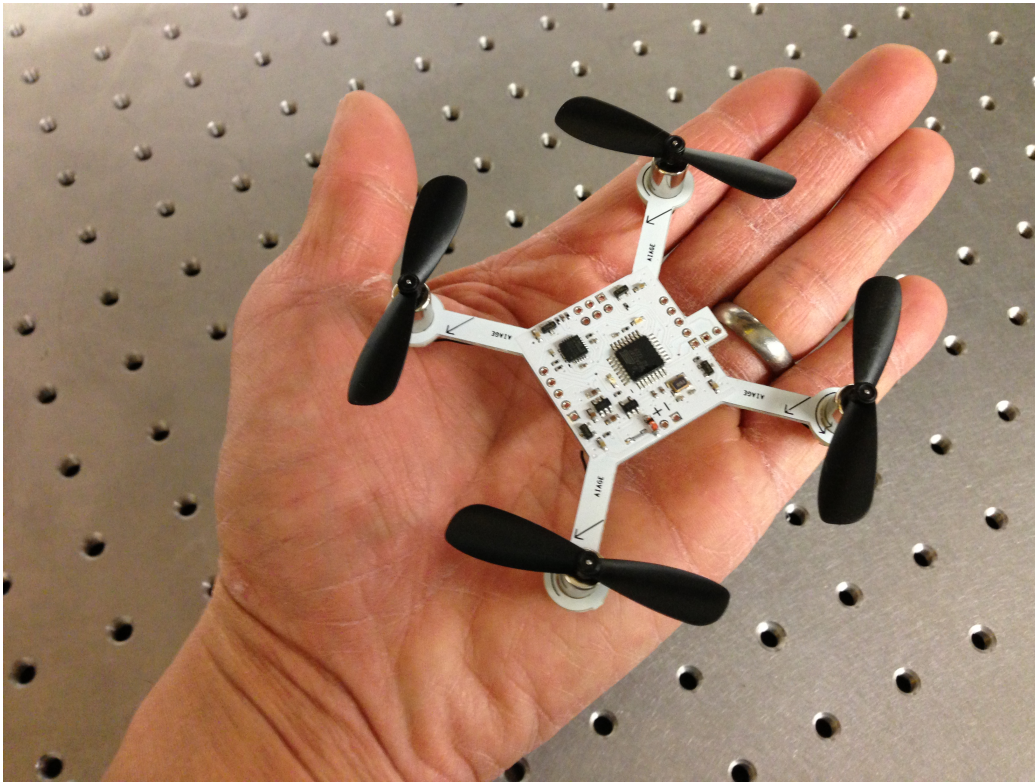


Figure 3.2: Micro quadcopter used to create the ATR.

The body of the quadcopter is fabricated from a single printed circuit board and carbon fiber arms (4.5 g) which significantly reduces the overall mass, with surface-

mount components that include a microcontroller (ATmega328P), three-axis gyro and accelerometer (MPU-6050), and MOSFET's for driving the motors. The platform is equipped with a 2.4 GHz XBee radio receiver for communication with a ground control station, and four small (7 mm x 17 mm) coreless brushed DC motors.

Compared to larger quadcopter systems where brushless motors are commonly used, the brushed DC motors used in the design do not require sophisticated electronic speed controllers, significantly reducing weight. However, the trade off in this case is longevity, as brushed motors tend to wear more quickly during use compared to brushless motors. The quadcopter uses an InvenSense MPU-6050 six-axis inertial measurement unit (IMU) that fuses raw accelerometer and gyroscope data to report the attitude angles, ϕ , θ , ψ , at 100 Hz for attitude stabilization in both modes of locomotion. The micro-quadcopter specifications are given in Table 3.1.

Table 3.1: Micro-quadcopter specifications

Component	Specification
Microcontroller	ATmega328P, 32 KB flash memory
IMU	MPU-6050 3 axis Gyro and Accelerometer, 100Hz
Communication	2.4 GHz Xbee 1mW trace antenna, 50Hz Microsoft Xbox Controller (User input)
Actuator	Coreless brushed DC motor (2.71 g) 407 mN Thrust, 16.6 W
Battery	350 mAh Li-Po (9.35 g)
Propeller	45 mm

The programming interface is a Serial Peripheral Interface (SPI) and enables the user to upload custom control software. Two way wireless communication between the platform and ground control station (GCS) is through a custom developed graphic user interface and Microsoft Xbox controller. Various buttons on the controller are used for pre-programmed autonomous flight, and the analog joysticks can be used for manual user input. A wireless link between the quadcopter and the GCS makes

on-the-fly controller tuning and remote monitoring of attitude, control effort, and battery voltage possible.

3.2 Exoskeleton Design

The ATR is able to conserve considerable power in rolling mode compared to conventional multirotor flying platforms. In initial work [25], the added mass of a protective cage was minimal in comparison to the overall payload capacity and size of the robot, but the added mass of the ATR's spherical exoskeleton is similar in magnitude to the mass and payload capacity of the quadcopter, and must be considered for efficiency purposes and impact on system dynamics. The spherical exoskeleton was chosen to fit tightly around the quadcopter platform so that the robot can easily be hand launched and is capable of maneuvering through tight indoor spaces. While a diameter of 135 mm fulfills the design requirements, a larger sphere diameter of 152 mm (6 in) was chosen to protect the inner quadcopter and allow for compliance of the sphere when bouncing or falling from heights. Each ring diameter is constrained by the overall sphere diameter, D , and can be designed for a desired turn radius, r_t from the design angle, γ , as,

$$r_t(\gamma_1, \gamma_2) = \frac{\sin \gamma_2 (\cos \gamma_2 - \cos \gamma_1)}{\sin \gamma_1 - \sin \gamma_2}. \quad (3.1)$$

The angle, γ , is measured from the \hat{b}_y axis to the intersection of the ring diameter d_n with the sphere, as shown in Fig. 3.3(b) and completely defines the turn radii when accompanied by D . The designer is given freedom to choose γ_1 , γ_2 , and γ_3 to achieve various turn radii depending on which pair of rings the ATR rolls. For example, a larger diameter ring can be used for long, sweeping maneuvers and smooth transition to straight rolling, or a smaller diameter ring can be used to maneuver in tighter

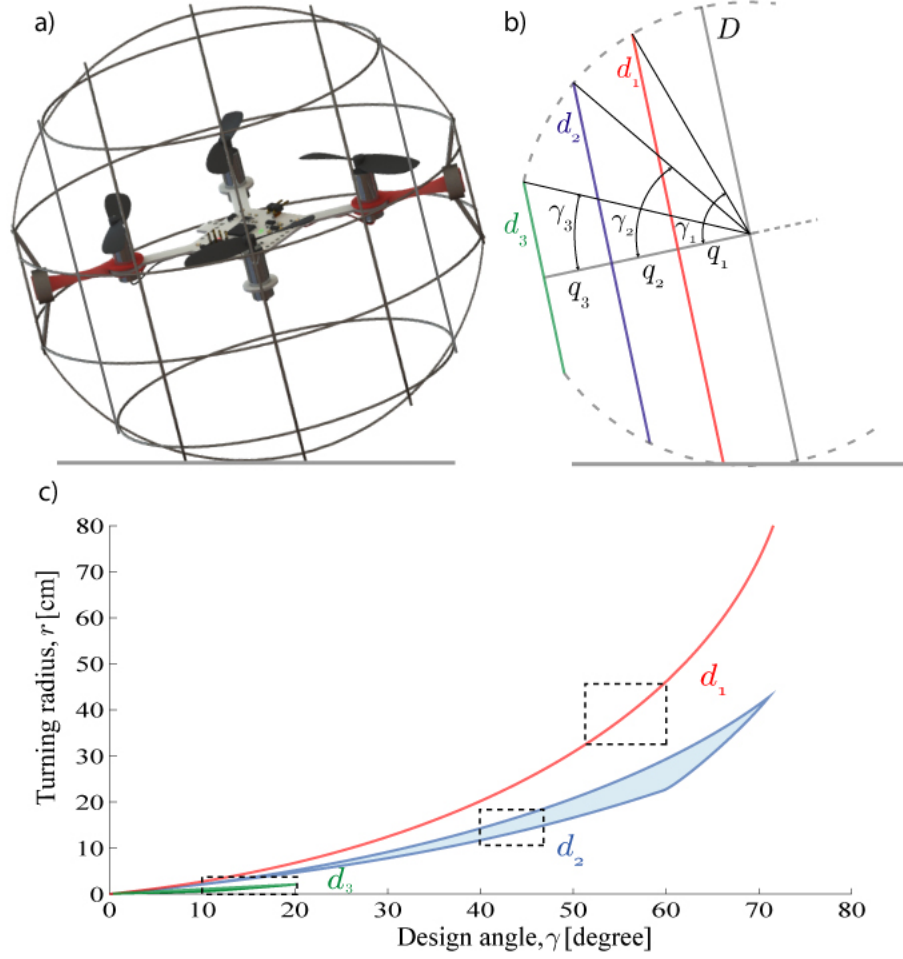


Figure 3.3: Spherical exoskeleton design: (a) mode of operation, (b) design geometry, and (c) turn radius corresponding to d_1 , d_2 , d_3 for varying γ .

constrained spaces. In Fig. 3.3(c), a range of possible d_1 is chosen to give a turn radius $32 \text{ cm} < r_{t1} < 48 \text{ cm}$ and from the resulting range of d_1 values, Eq. (3.1) and Fig. 3.3(c) the designer can determine proper ranges for d_2 , and d_3 to satisfy design requirements. For this design, $r_{t1} = 32.1 \text{ cm}$, $r_{t2} = 17 \text{ cm}$, and $r_{t3} = 6 \text{ cm}$ are chosen to give a wide range of possible turning radii. It is important to note that the diameter of each ring also influences the overall mass of the exoskeleton quadratically by,

$$m_n = \frac{\rho_w \pi^2 d_n^2 d_w}{4}, \quad (3.2)$$

where ρ_w and d_w are the ring material density and diameter, d_n is the ring diameter, and m_n is the ring mass. The designer must take care to design an exoskeleton that meets both the terrestrial maneuverability requirements while still satisfying the requirements for aerial flight. Due to extreme payload limitations of approximately 17 g, the exoskeleton is designed to be as light as possible, while still allowing the ATR to fall and tumble from heights. Additionally, the steel exoskeleton is designed with a degree of compliance that allows elastic deformation of the cage during impact, preventing damage to the robot.

3.3 Fabrication

Since the ATR has an extremely limited payload, the spherical exoskeleton was designed to be as light as possible. Various materials were considered to fulfill the design requirements, and are shown in Fig. 3.5, such as: 3D Printed Polylactic Acid (PLA), carbon fiber composite, titanium wire, and various diameters of spring steel wire. The difficulties in working with tough to join materials like titanium and complex molds for composite fabrication prevented quick design iteration, and ultimately, spring steel wire was chosen for the design due to its relatively easy workability and resulting lightweight exoskeleton (6.54 g as built). It is noted that these alternate materials can yield a lighter weight exoskeleton, but the steel wire design satisfies payload restrictions.

The exoskeleton of the ATR is fabricated from 0.020" diameter steel spring wire, commonly referred to as music wire. The form shown in Fig. 3.4(a) used for the concentric and orthogonal wire rings was constructed by inscribing grooves into a 6 inch polypropylene half-sphere on a horizontal lathe. The grooves serve as a fixture for the steel rings during construction and ensure that the finished ATR is perfectly spherical, resulting in smooth terrestrial locomotion. After constructing the mold,

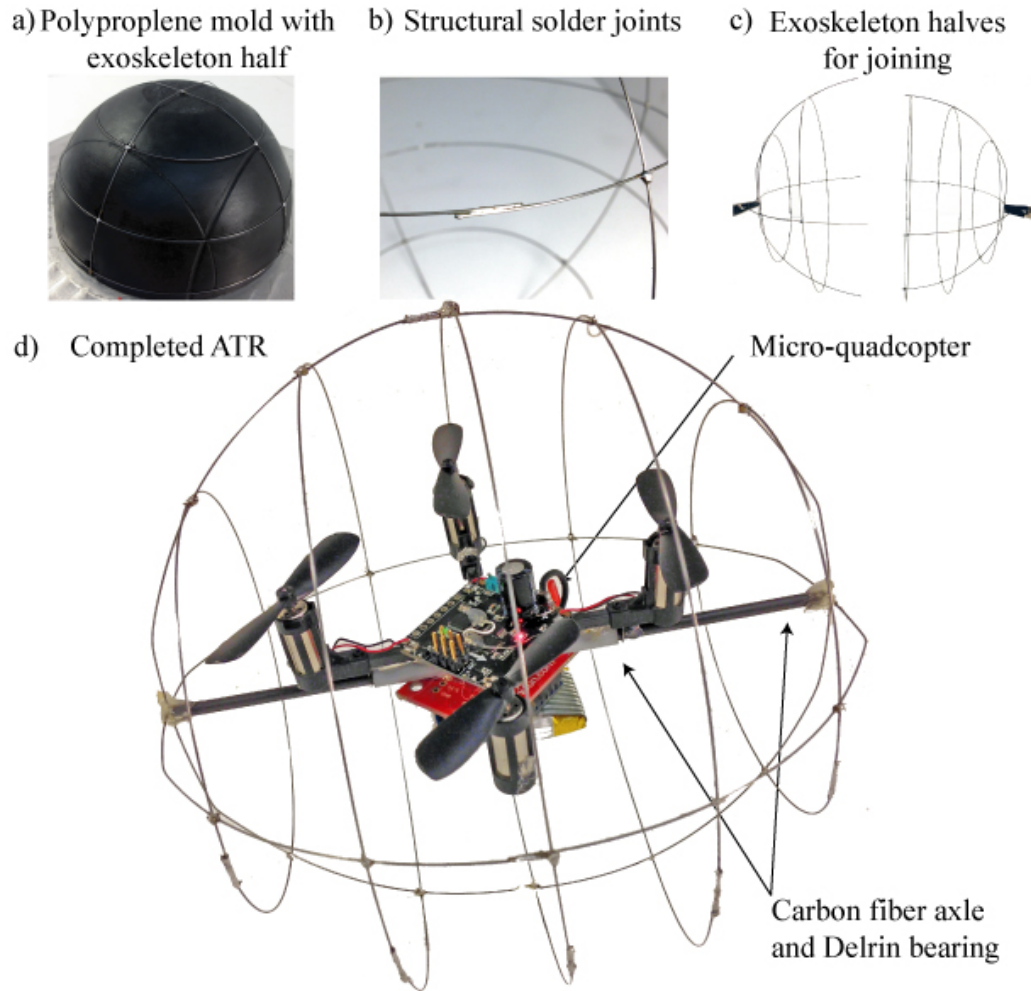


Figure 3.4: Fabrication of the spherical exoskeleton: (a) completed half sphere on polypropylene mold, (b) structural solder joining the structure, (c) combining of completed half spheres, and (d) completed ATR with carbon fiber axle, low-friction Delrin journal and micro-quadcopter.

wire bands were sized on the mold, sanded to remove surface contamination and oxidization, cleaned with a solvent, and then fluxed and soldered to create a loop as shown in Fig. 3.4(b). The soldered joint was then tightly wrapped with strands of thin copper wire, fluxed, and soldered once again to improve the joint integrity to ensure the ATR does not break on impact. The completed sphere halves shown in Fig. 3.4(c) were joined with solder at the central ring of the right half, completing the exoskeleton.

The quadcopter mass center was designed to coincide with the rotation axle of the ATR for balanced rotation about the \hat{b}_y axis and a neutral inactive position. In previous work, a similar design includes a mass center offset that guarantees stability at rest [25]. Since the inertia of the quadcopter used for the ATR is small compared to the motor pitching moment, the quadcopter can be easily righted at the beginning of operation from any orientation. The axle is designed to be as light as possible while firmly grasping and preventing any rotation of the quadcopter fuselage. Previous designs used a 3D printed PLA plastic axle, but low-rigidity and misalignment increased drag in the bearing. The new bearing design uses a rigid $\frac{1}{8}$ inch hollow carbon fiber shaft fixed to the exoskeleton. The quadcopter is then affixed to a Delrin[®] acetal journal shown in Fig. 3.4(d). The axle extends the diameter of the exoskeleton and improves the rigidity of the ATR while maintaining proper alignment for the micro-quadcopter to rotate in the exoskeleton.

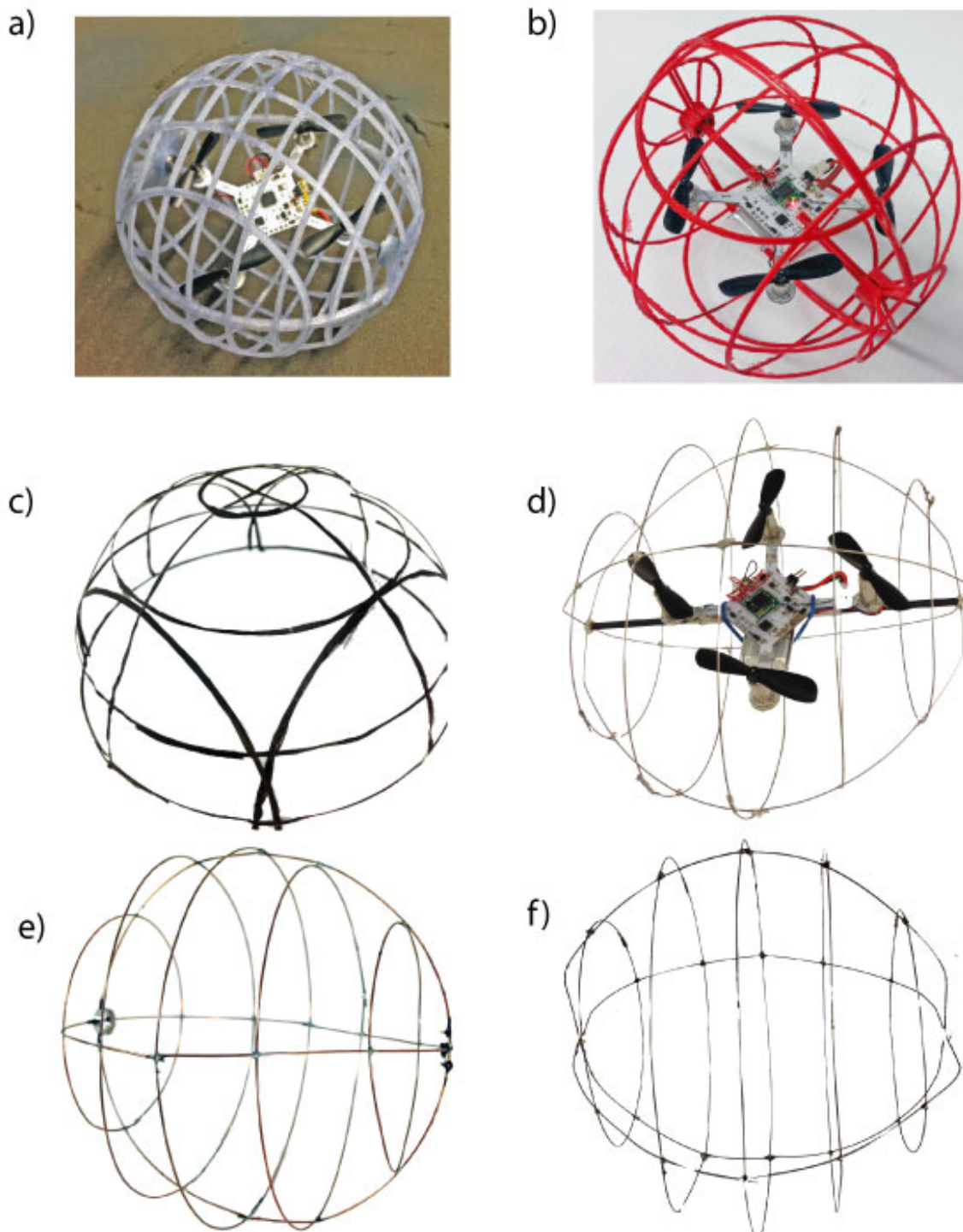


Figure 3.5: Exoskeleton design iterations: (a)3D printed PLA, (b)lightweight 3D printed PLA (c)carbon fiber composite , (d)titanium spring wire (e)steel welding wire 0.30 in diameter, and (f)0.20 in spring steel wire (music wire).

3.4 Control Hardware and Software

3.4.1 Quadcopter Electronics

The micro-quadcopter used in this design is actuated by small (7 mm x 17 mm) brushed DC motors. The motors are driven by an 8 kHz pulse-width modulated 3.3 V signal that switches a metal-oxide field-effect transistor (FET) to control the motor drive-voltage. The quadcopter is also fitted with two light emitting diodes (LED) that allow the user to program indicators such as whether the platform is enabled or disabled. The electronics schematic is shown in Fig. 3.6.

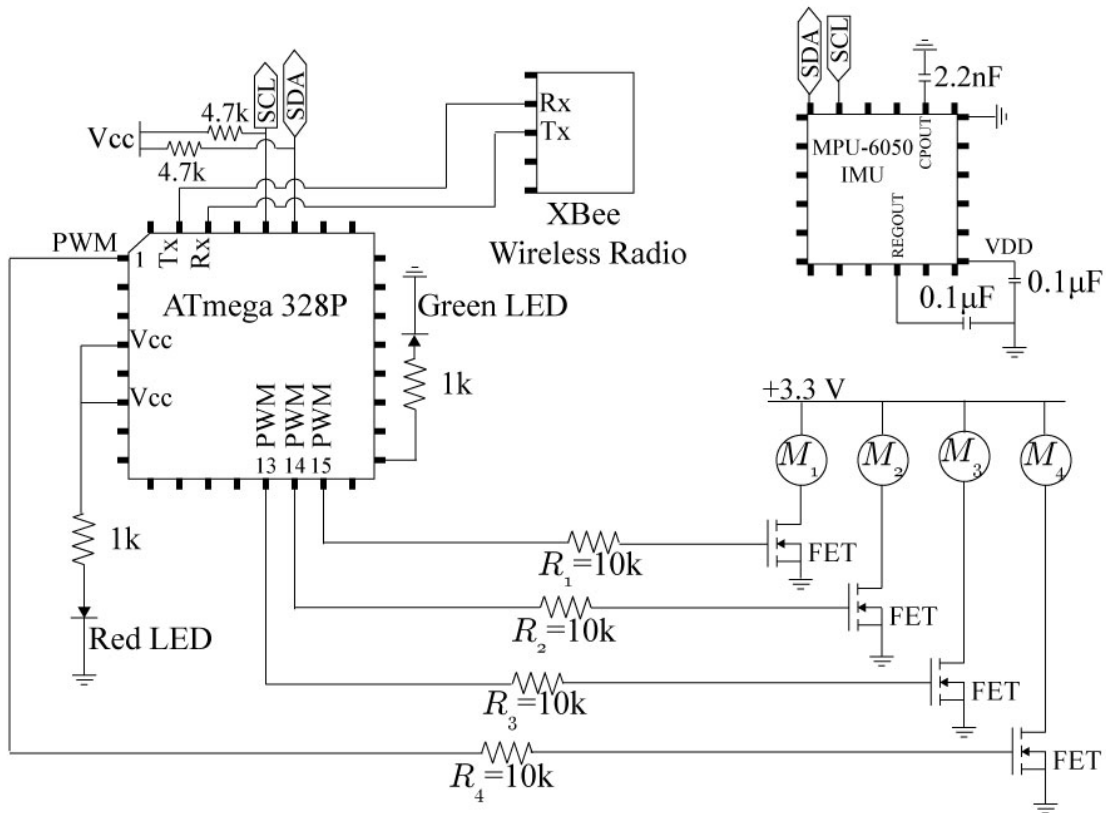


Figure 3.6: Circuit diagram for the micro-quadcopter platform with Wireless XBee radio

3.4.2 Wireless Communication

The ATR is capable of wireless serial communication at 50 HZ with a fitted 2.4 GHz XBee radio. This system was chosen due to its high reliability, low power, robustness, and ease of interfacing. Individual registers are set and the firmware was slightly modified in order to obtain communication rate operation at 57600 baud, interfacing over a serial line and minimal delay.

A custom graphic user interface (GUI), shown in Fig. 3.7, was developed to send autonomous tasks, or joystick commands to the ATR. The user is also able to adjust controller gains, switch between ground and aerial control modes, clear accumulated controller errors, and monitor the system attitude. The GUI was developed in MATLAB, which allows trajectories and motor commands to be easily sent to the ATR from simulations in Simulink.

3.4.3 Inertial Measurement Unit

The development of fast, accurate, and affordable solid state accelerometers, initially spurred by consumer electronics like smart phones, has been instrumental in the development of inertial measurement units (IMU) for quadcopters. For example, the IMU used in this design can deliver orientation and acceleration data at 100 Hz for stabilization and attitude measurement. The IMU has two primary sensors for attitude control: a 3-axis accelerometer that detects accelerations and a 3-axis gyroscope for determining angular rates. This combination allows for real-time measurement of the angular rates, as well as the accelerations that the quadcopter experiences. InvenSenses Digital Motion Processor (DMP) uses these sensors to provide fast and accurate attitude data such as roll, pitch and yaw. The MPU-6050 communicates with the ATmega328p using the I2C bus running at 400 KHz allowing for fast retrieval of data from the DMP.

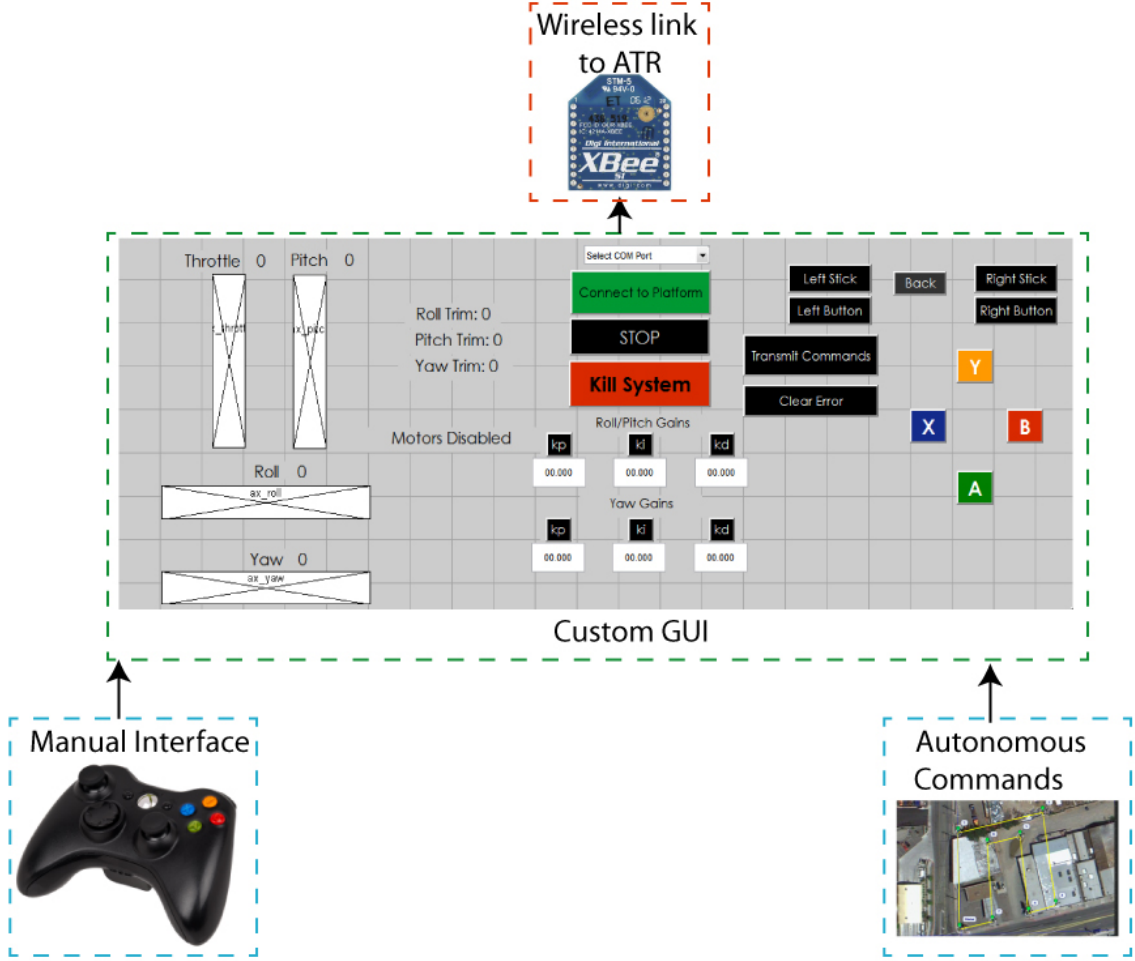


Figure 3.7: Custom designed graphic user interface with options for joystick input and autonomous commands. The ground control station communicates with the ATR at 50 Hz via wireless 2.4 GHz XBee radio.

3.4.4 Digital PID Control

The focus of controller synthesis for the ATR is stabilization of each orientation angle ϕ , θ , and ψ . Once the attitude of the robot is controlled, higher level position controllers can be implemented for rate control, object avoidance, and trajectory tracking with the use of additional sensors such as motion capture systems [26], optic flow [27], sonar [28], LIDAR [29], and barometric sensors.

The control architecture used in this work is shown in Fig. 3.8 and consists of an inertial measurement unit sensor and cascaded discrete proportional-integral-

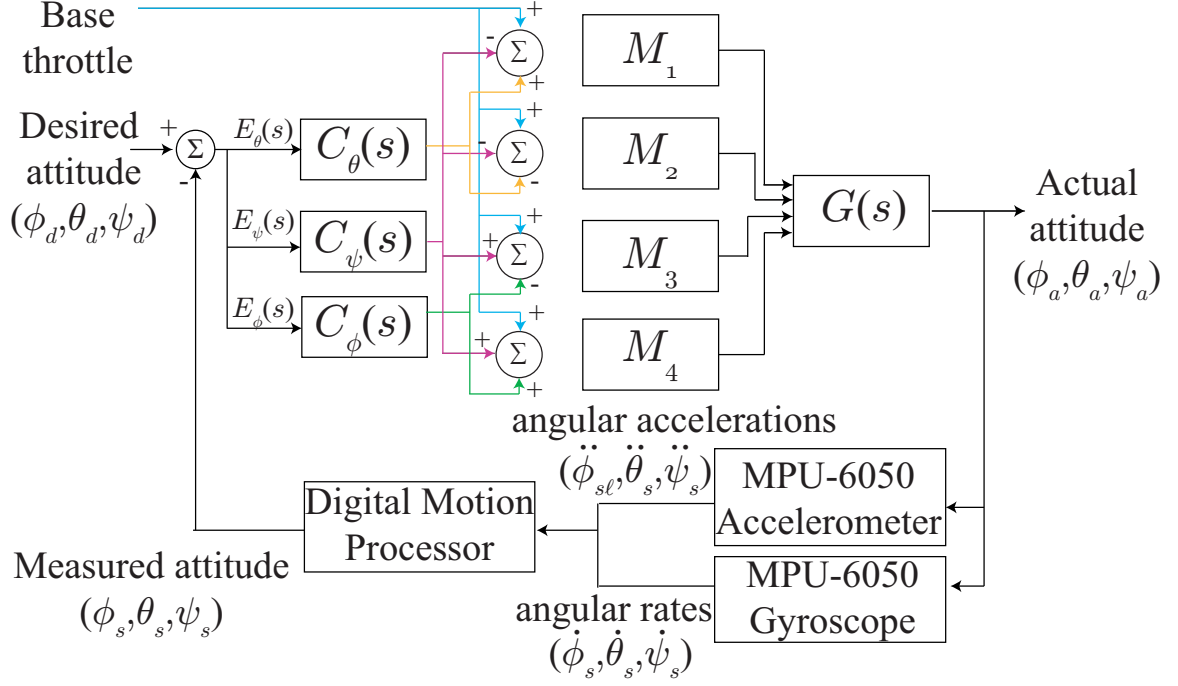


Figure 3.8: Attitude controller design for stabilization of the robot orientation, ϕ , θ , ψ .

derivative controller operating at 100 Hz. A base throttle command is used to control the ATR's z height in open loop and distributed to each motor. Each controller effort for the respective orientation angles is either added or subtracted from the respective motor. For example, if a positive correction in pitch angle is desired, the angular velocity of M_1 is increased while that of M_2 is decreased based on the controller effort.

3.5 Summary

this chapter presented the ATR design, including the lightweight spherical exoskeleton, micro quadcopter, control hardware and software. Additionally, system fabrication details are discussed. The spherical exoskeleton is designed to enable the robot perform various radii turns while rolling and is lightweight enough so that the robot

can still fly. The control hardware and software allows for wireless communication, control, and parameter tuning. A digital PID controller is used for attitude stabilization and is implemented at 100 Hz.

Chapter 4

System Modeling

Developing the governing equations of motion is an essential aspect for design and control of the ATR. The following section presents a model that captures the aerial and terrestrial locomotion of the ATR.

4.1 Reference Frames

Reference frames are introduced to represent and measure properties such as position, orientation, and velocity of a rigid body as shown in Fig. 4.1, and are defined by a basis, or linearly independent subset of a vector field, \mathbb{F} , that spans the dimension of the frame. The motion of a particle moving in space is described by giving the location of the particle at each instant of time, relative to an inertial Cartesian coordinate frame. Specifically, one can choose a set of three orthogonal axes and specify the particles location using the triple $(\hat{e}_x, \hat{e}_y, \hat{e}_z) \in \mathbb{R}^3$, where each coordinate gives the projection of the particles location onto the corresponding axis. The trajectory of the particle is represented by the parameterized curve $p(t) = (x(t), y(t), z(t)) \in \mathbb{R}^3$. An inertial reference frame $E \in \mathbb{R}^3$ can be defined as a particular reference frame that has a constant velocity and does not accelerate. A Newtonian reference frame

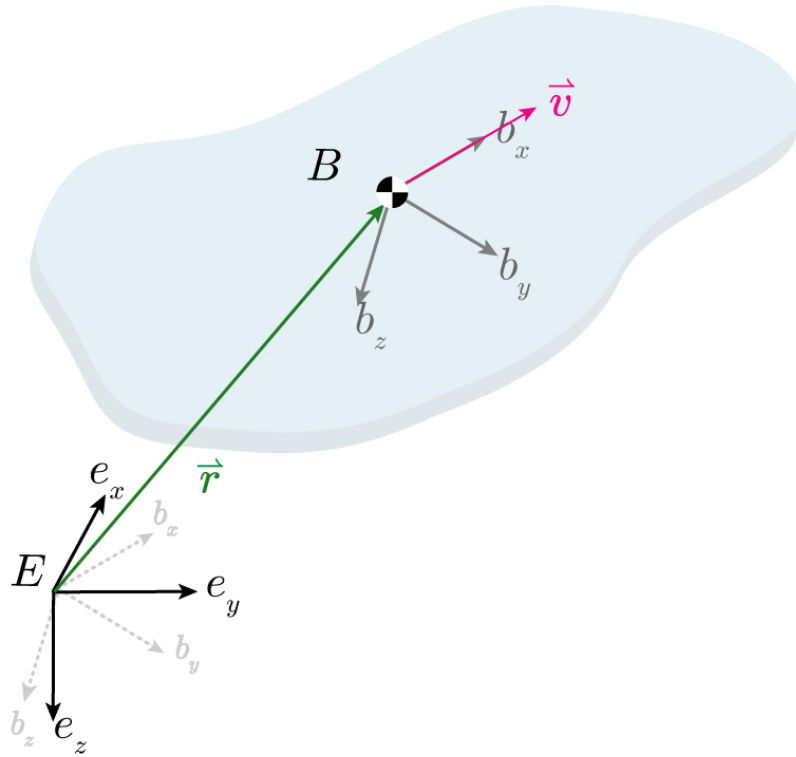


Figure 4.1: Rigid body in space with body-fixed accelerating reference frame B . An additional reference frame, E , is defined and is inertial. The velocity of the body can be expressed in both reference frames B , and E , such that a transformation between B and E exists for all orientations.

is analogous to the inertial reference frame which satisfies Newton's First Law. That is, the motion of a particle not subject to any external forces is in a straight line at a constant velocity. The inertial reference frame, E , used for this work is assumed to be fixed in earth at a home location. The \hat{e}_x axis is directed north, \hat{e}_y is directed east, and \hat{e}_z is directed to the center of earth. For the scope of this work, Earth's curvature is negligible. An additional non-inertial, accelerating reference frame, \hat{B} fixed to the rigid body can be formed by \hat{b}_x , \hat{b}_y , \hat{b}_z , directed forward, right, and downward perpendicular to the body.

4.2 Coordinate Transformation

A transformation exists to express a vector in any reference frame and can be represented in various methods including Euler rotation angles, quaternion transformation, and angle-axis representation. By introducing an additional reference frame for rigid-body dynamics, some vector quantities may be easier to represent in another frame. For example, in Fig. 4.1 the velocity of the rigid body can be represented in B and E as,

$$\vec{v}_{\{B\}} = a\hat{b}_x, \quad (4.1)$$

$$\vec{v}_{\{B\}} = c\hat{e}_x + d\hat{e}_y + f\hat{e}_z, \quad (4.2)$$

where the subscript $\{B\}$ indicates the reference frame of interest. Representing $\vec{v}_{\{B\}}$ in B is much simpler than in E because $\vec{v}_{\{B\}}$ consists of only one component rather than three. This representation allows for more direct and less computationally demanding vector operations and the result can easily be transformed to E if necessary.

4.2.1 Euler Rotation Matrix

Considering the right hand coordinate system, the transformation between B and E in Fig. 4.1, can be represented by three single rotations:

$R(x, \phi)$ Rotation about the \hat{b}_x axis

$$R(x, \phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \quad (4.3)$$

$R(y, \theta)$ Rotation about the \hat{b}_y axis

$$R(y, \theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad (4.4)$$

$R(z, \psi)$ Rotation about the \hat{b}_z axis

$$R(z, \psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.5)$$

The complete rotation matrix to represent a rigid body parameterized by angles ϕ , θ , ψ is given by the product of Eqs. (4.3), (4.4), (4.5), given by,

$$R(\phi, \theta, \psi) = \begin{bmatrix} c(\phi)c(\theta) & s(\theta)c(\psi)s(\phi) - s(\psi)c(\phi) & s(\phi)s(\psi) + c(\phi)s(\theta)c(\psi) \\ s(\psi)c(\theta) & c(\psi)c(\phi) + s(\psi)s(\theta)s(\phi) & c(\phi)s(\psi)s(\theta) - s(\phi)c(\psi) \\ -s(\theta) & s(\phi)c(\theta) & c(\phi)c(\theta) \end{bmatrix}, \quad (4.6)$$

where the notation $s(\theta) = \sin(\theta)$ While the transformation in Eq. (4.6) provides a relatively simple parametrization, singularities in the rotation matrix occur at $R = I$, the identity rotation given by $R(\gamma, 0, -\gamma)$ for example. This singularity refers to a lack of existence of a global solution to determine the rotation angles from the physical rotation, so configurations of this nature are inherently unsolvable with the Euler rotation matrix method.

4.2.2 Quaternion Transformation

Unlike Euler angles, quaternions give a global parametrization of a rotation in $SO(3)$. The quaternion is a hyper-complex number consisting of a three element vector defining the axis perpendicular to rotation and a scalar quantity defining the magnitude of the rotation. Formally, the quaternion is given by,

$$Q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}, \quad (4.7)$$

or alternatively in compact form as $Q = (q_0, \vec{q})$ where q_0 is the scalar component and $\vec{q} = (q_1, q_2, q_3)$ is the vector component with $q_0 \in \mathbb{R}$ and $\vec{q} \in \mathbb{R}^3$. Multiplication in the quaternion space is both distributive and associative, but not commutative and satisfies the relations,

$$a\mathbf{i} = \mathbf{i}a, \quad a\mathbf{j} = \mathbf{j}a, \quad a\mathbf{k} = \mathbf{k}a, \quad a \in \mathbb{R},$$

$$\mathbf{i} \cdot \mathbf{i} = \mathbf{j} \cdot \mathbf{j} = \mathbf{k} \cdot \mathbf{k} = -1, \quad (4.8)$$

$$\mathbf{i} \cdot \mathbf{j} = -\mathbf{j} \cdot \mathbf{i} = \mathbf{k}, \quad \mathbf{j} \cdot \mathbf{k} = -\mathbf{k} \cdot \mathbf{j} = \mathbf{i}, \quad \mathbf{k} \cdot \mathbf{i} = -\mathbf{i} \cdot \mathbf{k} = \mathbf{j}.$$

The conjugate of a quaternion is given by $Q^* = (q_0, -\vec{q})$ and the magnitude of a quaternion satisfies,

$$\|Q\|_2 = Q \cdot Q^* = q_0^2 + q_1^2 + q_2^2 + q_3^2. \quad (4.9)$$

The quaternion product between two quaternions Q and P is given by the inner and cross products between the vectors and has the form,

$$Q \otimes P = (q_0 p_0 - \vec{q} \cdot \vec{p}, q_0 \vec{p} + p_0 \vec{q} + \vec{q} \times \vec{p}). \quad (4.10)$$

If a quaternion is a unit quaternion given by $\frac{|q|}{q}$ it can be used as a rotation operator; however, the transformation is not built up as one quaternion product, but two - the normal and its conjugate, given by,

$$\vec{v}' = Q \otimes \begin{bmatrix} 0 \\ \vec{v} \end{bmatrix} \otimes Q^* \quad (4.11)$$

Additionally, if P represents one rotation and Q represents an additional rotation, the quaternion product of P and Q, $P \otimes Q$, represents the combined rotation, making complex rotations simply quaternion products.

The rotation in Eq. (4.11) can be configured to represent rotations about the principle axes by replacing \vec{v} with the respective axes and is given by,

$$R(x, \phi) = Q \otimes \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \otimes Q^* = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 \\ 2(q_1 q_2 + q_0 q_3) \\ 2(q_1 q_3 - q_0 q_2) \end{bmatrix} \quad (4.12)$$

$$R(y, \theta) = Q \otimes \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \otimes Q^* = \begin{bmatrix} 2(q_1 q_2 - q_0 q_3) \\ q_0^2 - q_1^2 + q_2^2 - q_3^2 \\ 2(q_2 q_3 + q_0 q_1) \end{bmatrix} \quad (4.13)$$

$$R(z, \psi) = Q \otimes \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \otimes Q^* = \begin{bmatrix} 2(q_1 q_3 + q_0 q_2) \\ 2(q_2 q_3 - q_0 q_1) \\ q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \quad (4.14)$$

Since the group structure for quaternions directly corresponds to that of rotations,

quaternions provide an efficient representation for rotations which do not suffer from singularities.

4.3 Vector Derivatives in Two Reference Frames

Referring to Fig. 4.1, if E and B are any two reference frames, the first time-derivative of any vector \vec{v} in B and E are related to each other by,

$$\begin{aligned}\frac{d\vec{v}}{dt}_{\{E\}} &= \frac{dv_i}{dt}\hat{b}_i + v_i \frac{d\hat{b}_i}{dt}_{\{E\}}, \\ &= \frac{d\vec{v}}{dt}_{\{B\}} + v_i \vec{\omega}_{\{B\}} \times \hat{b}_i, \\ &= \frac{d\vec{v}}{dt}_{\{B\}} + \vec{\omega}_{\{B\}} \times \vec{v},\end{aligned}\tag{4.15}$$

with $\hat{b}_i = \hat{b}_x, \hat{b}_y, \hat{b}_z$ as the set of mutually perpendicular unit vectors that define the basis for reference frame B . Equation (4.15) enables one to find the time derivative of \vec{v} in E without having to resolve the vector into components parallel to the axes defining the reference frame E .

4.4 Newton's Second Law in Body Coordinates

In the case of Newton's second Law, the time rate of change of a particle's linear momentum is equal to the net sum of all the forces acting on the particle, i.e.,

$$\vec{f} = \frac{d}{dt}(m\vec{v}).\tag{4.16}$$

Assuming that the particle's mass is time-invariant, and \vec{v} is expressed in the body coordinates $\hat{b}_x, \hat{b}_y, \hat{b}_z$ then,

$$\begin{aligned}\vec{f} &= m \frac{d\vec{v}}{dt}, \\ \vec{f}_B &= m \frac{d\vec{v}}{dt}_{\{B\}} + m \vec{\omega}_{\{B\}} \times \vec{v}, \\ &= m \dot{\vec{v}}_{\{B\}} + \vec{\omega}_{\{B\}} \times m \vec{v}.\end{aligned}\tag{4.17}$$

Equation (4.17) is Newton's Second Law in body coordinates. When \vec{v} is represented in the non-accelerating frame E , Eq. (4.17) reduces to a common form,

$$\begin{aligned}\vec{f}_{\{E\}} &= m \dot{\vec{v}}_{\{E\}} + \vec{\omega}_{\{B\}} \times m \vec{v}, \\ &= m \dot{\vec{v}}_{\{E\}} + 0 \times m \vec{v}, \\ &= m \dot{\vec{v}}_{\{E\}}.\end{aligned}\tag{4.18}$$

4.5 Euler's Second Law in Body Coordinates

Similar to Newton's Second Law, presented in Eq. (4.19), the sum of applied torques on a rigid body is equal to the time rate of change of the body's central angular momentum, \vec{H} , is,

$$\vec{\tau} = \frac{d}{dt} \vec{H},\tag{4.19}$$

where,

$$\vec{H} = I_x \omega_x \hat{b}_x + I_y \omega_y \hat{b}_y + I_z \omega_z \hat{b}_z.\tag{4.20}$$

Assuming that the body's inertia is time-invariant, and $\vec{\omega}$ is expressed in the body coordinates $\hat{b}_x, \hat{b}_y, \hat{b}_z$,

$$\vec{\tau}_{\{B\}} = \frac{d\vec{H}}{dt}_{\{B\}} + \vec{\omega}_{\{B\}} \times \vec{H},\tag{4.21}$$

$$\frac{d\vec{H}}{dt}_{\{B\}} = I_x \dot{\omega}_x \hat{b}_x + I_y \dot{\omega}_y \hat{b}_y + I_z \dot{\omega}_z \hat{b}_z, \quad (4.22)$$

$$\begin{aligned} \vec{\omega}_{\{B\}} \times \vec{H} = & \omega_y \omega_z (I_z - I_y) \hat{b}_x + \omega_x \omega_z (I_x - I_z) \hat{b}_y \\ & + \omega_x \omega_y (I_y - I_x) \hat{b}_z. \end{aligned} \quad (4.23)$$

Equation (4.22) is Euler's Second Law represented in body coordinates. When \vec{H} is represented in a fixed inertial frame, Eq. (4.22) reduces to a common form,

$$\vec{\tau}_{\{E\}} = \frac{d\vec{H}}{dt}_{\{E\}} + 0 = I \dot{\vec{\omega}}_{\{B\}}. \quad (4.24)$$

4.6 Newton-Euler Equations

Equations (4.17) and (4.22) fully describe the translational and rotational dynamics of a rigid body in space. A compact matrix representation of the Newton-Euler equations is given by,

$$\begin{bmatrix} \vec{f}_{\{B\}} \\ \vec{\tau}_{\{B\}} \end{bmatrix} = \begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \dot{\vec{v}}_{cm\{B\}} \\ \dot{\vec{\omega}}_{\{B\}} \end{bmatrix} + \begin{bmatrix} \vec{\omega}_{\{B\}} \times m \vec{v}_{cm\{B\}} \\ \vec{\omega}_{\{B\}} \times I \vec{\omega}_{\{B\}} \end{bmatrix}, \quad (4.25)$$

where $\mathbf{M} \in \mathbb{R}^3$, $\mathbf{I} \in \mathbb{R}^3$ are the mass and inertia matrix. The body's linear and angular accelerations $\dot{\vec{v}}_{cm\{B\}}$, $\dot{\vec{\omega}}_{\{B\}}$, external forces, $\vec{f}_{\{B\}}$, and torques, $\vec{\tau}_{\{B\}}$, are defined in the body frame.

4.7 Aerial Locomotion

The quadcopter can be approximated as a rigid body with six degrees of freedom in the inertial frame. The position and attitude of the aircraft are given by $(\hat{e}_x, \hat{e}_y, \hat{e}_z)$ and $(\hat{\phi}, \hat{\theta}, \hat{\psi})$, respectively. The general Newton-Euler Equation (4.25) can be applied

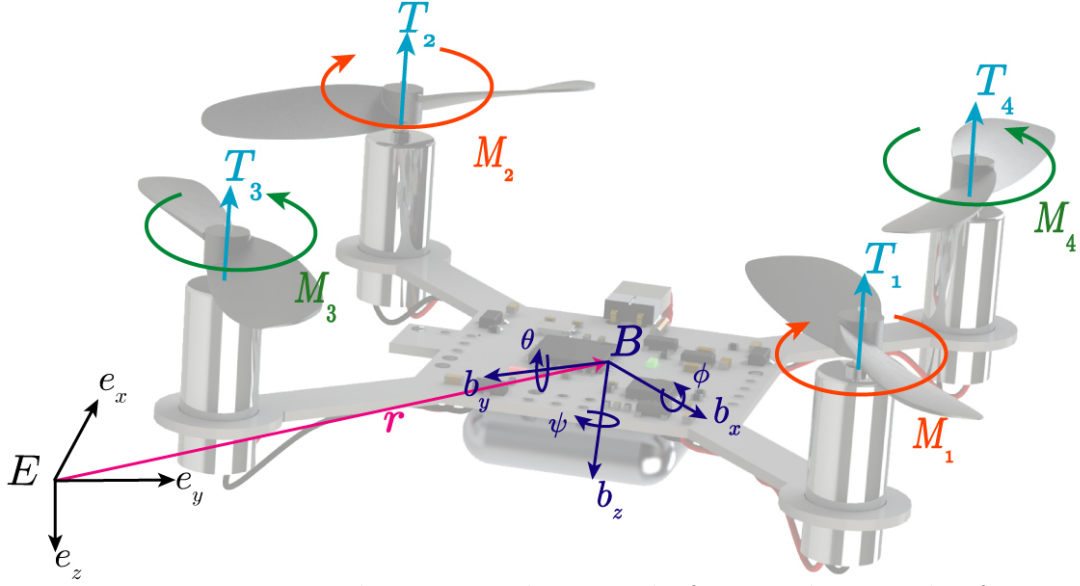


Figure 4.2: Experimental micro-quadcopter platform with inertial reference frame E , defined by \hat{e}_x , Northward, \hat{e}_y , Eastward, and \hat{e}_z , directed to the center of the earth. The quadcopter body frame is defined by \hat{b}_x , forward, \hat{b}_y pointed right, and \hat{b}_z downward.

to the ATR flying platform with minor modifications. First, it is assumed that the quadcopter platform is symmetric about the \hat{b}_x and \hat{b}_y axis, resulting in a symmetric and diagonal inertia matrix — no products of inertia exist, only three principle moments of inertia. The exoskeleton is assumed to be uniform in inertia about each principle axis, and only affects the inertia of the ATR system in the I_{yy} and I_{zz} direction as the bearing friction is much smaller in magnitude compared to the actuator action in aerial mode and the actuators do not have a direct impact on the rotation of the exoskeleton in aerial mode. The body forces and torques from the actuators are given by,

$$\vec{f}_{\{b\}} = [0, 0, -\sum_{i=1}^4 T_i], \quad (4.26)$$

$$\vec{\tau}_{\{b\}} = [l(T_4 - T_3), l(T_1 - T_2), (M_3 + M_4 - M_1 - M_2)], \quad (4.27)$$

where l is the quadcopter arm length. The motor thrust T_i and moment M_i are dependent on the motor torque and moment constants k_T , k_M and are a quadratic

function of the rotor angular velocity Ω_i , i.e., [30]

$$\vec{T}_i = -k_T \Omega_i^2 \hat{b}_z, \quad \vec{M}_i = -k_M \Omega_i^2 \hat{b}_z. \quad (4.28)$$

As shown in Fig. (4.2), the quadcopter can be represented in both the inertial reference frame, E , and the quadcopter rigid body frame B . Note that the linear and angular motions are coupled since the linear velocity in body coordinates depends on the current orientation. Each rotor speed can be calculated for a desired orientation from Eq. (4.26) and (4.27) and is given by,

$$\vec{\Omega} = \begin{bmatrix} \Omega_1 \\ \Omega_2 \\ \Omega_3 \\ \Omega_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & -1 & 0 & 1 \\ 1 & 0 & -1 & -1 \\ 1 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} \sqrt{\frac{-mg}{4k_T}} \\ \sqrt{\frac{\tau_{by}}{2lk_T}} \\ \sqrt{\frac{\tau_{bx}}{2lk_T}} \\ \sqrt{\frac{\tau_{bz}}{4k_M}} \end{bmatrix}. \quad (4.29)$$

Combining Eq. (4.25), (4.26), and (4.27) gives a representation of the dynamic equations of motion for the ATR in the body frame, B ,

$$\ddot{z} = \sum_{k=1}^4 \frac{T_i}{m} - g \hat{e}_z \cdot \hat{b}_z, \quad (4.30)$$

$$\ddot{\phi} = \frac{1}{I_{xx}} [k_T l (\Omega_4^2 - \Omega_3^2) + \dot{\theta} \dot{\psi} (I_{yy} - I_{zz}) + \sum_{k=1}^4 J_{r_k} \Omega_k \dot{\theta} + I_{s_{zz}} \dot{\psi} \dot{\alpha}],$$

$$\ddot{\theta} = \frac{1}{I_{yy}} [k_T l (\Omega_1^2 - \Omega_2^2) + \dot{\phi} \dot{\psi} (I_{zz} - I_{xx}) - \sum_{k=1}^4 J_{r_k} \Omega_k \dot{\phi} + I_{s_{yy}} \ddot{\alpha}],$$

$$\ddot{\psi} = \frac{1}{I_{zz}} [k_M (\Omega_3^2 + \Omega_4^2 - \Omega_1^2 - \Omega_2^2) + \dot{\phi} \dot{\theta} (I_{xx} - I_{yy}) + \sum_{k=1}^4 J_{r_k} \dot{\Omega}_k + I_{s_{xx}} \dot{\phi} \dot{\alpha}].$$

where $\ddot{\alpha}$ is the angular acceleration of the spherical exoskeleton, J_r is the rotor inertia, and $\Omega_{1,2,3,4}$ are the individual motor angular velocities.

It is important to note that since the inertia of the exoskeleton and the inertia of the quadcopter platform are similar in magnitude, the rotation, α , and inertia, \mathbf{I}_s , of the exoskeleton can greatly affect aerial locomotive dynamics. The addition of the exoskeleton does not increase the system inertia, \mathbf{I} , uniformly. The I_{yy} and I_{zz} terms nearly double, while I_{xx} remains only that of the quadcopter, resulting non-axisymmetric dynamics that are more difficult to control than traditional quadcopter systems. The lightweight exoskeleton with inertia \mathbf{I}_s , provides an additional torque on the ATR during flight if the exoskeleton is rotating after take-off at an angular velocity of, $\dot{\alpha}\hat{b}_y$. Finally, it is noted that the effect of torques opposing the exoskeleton angular momentum is reduced with increasing $\dot{\alpha}$, which can contribute positively to the ATR's ability to maintain a desired heading in aerial mode.

4.8 Terrestrial Locomotion

Rolling forward or backward can be achieved by orienting the micro-quadcopter in a manner that provides a component of thrust along the horizontal direction (parallel to the ground). A novelty of the ATR is its capability to turn while rolling in a method similar to a railcar, where the ATR can position itself on its exoskeleton rings of varying diameter in contact with the ground, causing the platform to turn in a circular manner. Additionally, the ATR is capable of turning in place by creating a moment imbalance between motors, analogous to yaw action in aerial mode. Perching, as shown in Fig. 3.1(c), allows the ATR to rest in a passively stable state and is accomplished by rotating the platform about its \hat{b}_x axis by $\pm 90^\circ$ onto one of the flat ends of the sphere. These three configurations describe the gross behavior of the ATR over the ground.

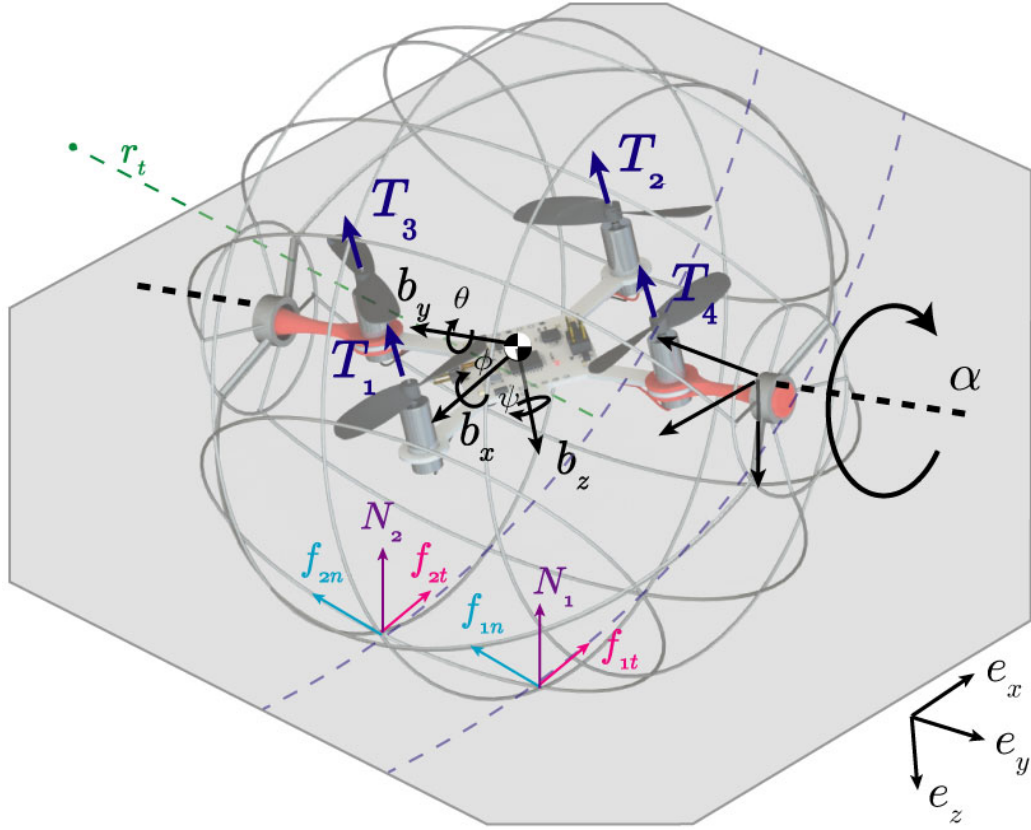


Figure 4.3: The ATR system in terrestrial locomotion mode, powered by the brushed motors propelling and directing the rolling exoskeleton. The ATR is capable of smooth turning by positioning itself onto a set of its rings and rolling in a circular pattern.

Rolling

The dynamic equations presented in Eq. (4.30) apply specifically to aerial locomotion and can generally be applied to the terrestrial locomotion with some considerations. As shown in Fig. 4.3, the ATR must maintain a set-point pitch angle by increasing or decreasing the angular velocity of motors 1 and 2 to achieve a desired pitch angle, θ , resulting in a horizontal component of thrust for forward or reverse rolling motion. A vertical configuration with $\theta = \pm 90^\circ$ provides the most terrestrial thrust for rolling, rotation about the \hat{b}_x axis becomes more difficult to control as it relies solely on the moment produced by the motors, rather than the thrust, resulting in a possible loss of control. This configuration also decreases the robot's ability to roll about the \hat{b}_x

axis, limiting its ability to roll and turn simultaneously. For these reasons and the fact that small pitch angles such as $\theta = 5 - 10^\circ$ provide sufficient thrust for quick rolling locomotion at low overall thrust, an attitude constraint is formed that ensures the robot remains in contact with the ground, and is given by,

$$\sum_{k=1}^4 k_T \Omega_i^2 \cos \theta \cos \phi - mg > \ddot{z}. \quad (4.31)$$

The ground motion of the ATR is limited to rolling, sliding, or a combination of the two. Consider straight line motion of the ATR, transitioning from aerial locomotion to terrestrial locomotion with initial velocity, \vec{v}_0 , thrust, \vec{T}_x , and initial angular velocity, $\dot{\vec{\alpha}}_0 = 0$. Upon ground impact, the acceleration of the mass center and angular acceleration of the sphere are,

$$\dot{\vec{v}}_{cm} = \frac{T_x}{m} \hat{e}_x - \mu_k g \hat{e}_z, \quad (4.32)$$

$$\ddot{\alpha} = \frac{\mu_k m g r_i}{I_{yy}}. \quad (4.33)$$

It is important to note that friction from the ground acts immediately, starting to slow the ATR down, bringing the exoskeleton into rotation. Pure rotation can only occur when, $\vec{v}_{cm} = r_i \dot{\vec{\alpha}}$, where r_i is the radius of the ring in contact with the ground and $\dot{\vec{\alpha}}$ is the angular velocity of the spherical exoskeleton. This relationship directly relates the rotation of the spherical exoskeleton and the linear velocity of the robot, forming a non-holonomic configuration constraint of pure rolling without slipping. Although a majority of the ATR's behavior follows the non-holonomic constraint, sliding and twisting motion must be considered during normal operation. From Eq. (4.32), (4.33), and the non-holonomic constraint, the impact condition of rolling or sliding along the ground can be determined from the initial impact velocity, \vec{v}_0 , thrust, \vec{T} , friction coefficient, μ_k , and contact radius, r_i .

The journal bearing used in the design exhibits a torque that opposes the rotation of the exoskeleton. This torque, τ_b , is primarily dependent on the contact surfaces, quadcopter platform mass, exoskeleton angular velocity, $\dot{\alpha}$, bearing temperature, and lubrication condition. A general expression for the opposing torque due to bearing friction is given by,

$$\vec{\tau}_b = mg\mu_b \frac{D}{2} \hat{b}_y, \quad (4.34)$$

where μ_b is the coefficient of rolling friction for the bearing and D is the bearing diameter. In practice, μ_k can vary up to an order of magnitude over the life of the bearing, so care must be taken in the choice of materials for the bearing design.

Turning

The novelty of the ATR's maneuverability in terrestrial locomotion mode lies in the shape of the spherical exoskeleton. In addition to yawing the platform and spinning about the \hat{b}_z axis, the robot is geometrically designed to turn via varying diameter concentric rings, similar to how a solid axle train navigates turns on its track. An actuation imbalance between motors 3 and 4, and corresponding robot roll angle, ϕ , places the robot onto different sized sets of rings, and when the exoskeleton rolls, the robot turns in a circular manner due to the smaller and larger path of each contact ring. The platform turning radius, r_t , from the inner contact ring is given by,

$$r_t = \frac{r_2 q}{r_1 - r_2}, \quad (4.35)$$

where r_2 , r_1 are the smaller and larger diameters of contact rings and q is the distance between rings as shown in Fig. 3.3. The ATR is equipped with three ring sets on either side of the robot, making complex turns of varying radius possible. The ATR can

position itself on the proper set of rings by rotating about its \hat{b}_x axis an amount,

$$\phi = \tan^{-1} \left(\frac{r_1 - r_2}{q} \right). \quad (4.36)$$

For simplicity in modeling the terrestrial rolling and turning, and additional reference frame is formed in the exoskeleton body frame. This frame is similar to the quadcopter body frame, but is not influenced by the rolling or pitching of the quadcopter and the \hat{b}_{s_x} and \hat{b}_{s_y} axes are coplanar with the \hat{e}_x and \hat{e}_y axes. The frame origin is at the mass center of the system (center of the sphere) and coincides with the quadcopter body frame. The forces and torques $T_{x,y,z}$, $\tau_{x,y,z}$ on the exoskeleton induced by the quadcopter actuators can be obtained from a simple transformation in a variety of methods outlined in 4.2. Assuming both rings are in contact with the ground during turning, the forces and torques on the exoskeleton in the exoskeleton body frame are given as,

$$\ddot{x} = \frac{1}{m}[T_x - f_{1t} - f_{2t}], \quad (4.37)$$

$$\ddot{y} = \frac{1}{m}[T_y - f_{1n} - f_{2n}],$$

$$\ddot{z} = \frac{1}{m}[-T_z + mg - N_1 - N_2],$$

$$\ddot{\phi} = \frac{1}{I_{xx}}[\tau_x + r_1 f_{1n} + r_2 f_{2n}], \ddot{\alpha} = \frac{-1}{I_{syy}}[r_1 f_{1t} + r_2 f_{2t} + \tau_b],$$

$$I_{zz}\ddot{\psi} = \frac{1}{I_{zz}}[\tau_z + \frac{q}{2}(f_{2t} - f_{1t})],$$

where f_{1t} , f_{2t} , f_{1n} , f_{2n} are the ring tangent and normal frictional forces as shown in Fig. 4.3, and N_1, N_2 are the normal forces in the \hat{e}_z direction. Terms in Eq. (4.30) containing angular rates $\dot{\theta}$ and $\dot{\phi}$ are neglected in terrestrial analysis as the angular

rates are minimal and do not appreciably impact terrestrial dynamics.

From Eq. (4.37) a relationship between the applied thrust, T_x , and the exoskeleton angular acceleration, $\ddot{\alpha}$, is given by,

$$\ddot{\alpha} = \frac{\tau_z + T_x q(r_2 - 0.5) - q\tau_b}{qI_{yy} + I_{zz} \frac{r_1 + r_2}{2r_t + \frac{q}{2}} + mq(r_2 - 1) \left(\frac{r_1 + r_2}{2} \right)}, \quad (4.38)$$

where r_1 and r_2 are the larger and smaller diameter rings of contact, q is the distance between rings, r_t is the turning radius, and τ_b is the bearing friction. The position and orientation of the exoskeleton are fully described by the rotation of the exoskeleton, α , and are parameterized in cartesian coordinates by,

$$x = x_0 + r_t \sin \left(\frac{\alpha(r_1 + r_2)}{2r_t + \frac{q}{2}} \right), \quad (4.39)$$

$$y = y_0 - r_t + r_t \cos \left(\frac{\alpha(r_1 + r_2)}{2r_t + \frac{q}{2}} \right),$$

$$\psi = \frac{\alpha(r_1 + r_2)}{2r_t + \frac{q}{2}}.$$

Equations (4.37), (4.38), (4.39), describe the motion of the ATR when turning in terrestrial mode.

Assuming the ATR can instantaneously change its roll angle, ϕ , complex paths can be achieved by varying the ATR's turn radius from each ring during rotation of the exoskeleton. An additional method for turning is accomplished by actuation of motor pairs 1 and 2, or 3 and 4, creating a net moment about the \hat{b}_z axis that causes the robot to pivot about its point of contact on the ground. This action, combined with the geometric constraint in Eqs. (4.36), (4.35), (4.37), and (4.39) allow the designer to optimize the robot design for complex ground maneuvers that require minimal effort.

Perching

The ATR is able to achieve passive stability from any arbitrary orientation by rotating about its center of mass to land on a perching surface located on either side of the robot as shown in Fig. 4.4. The onboard inertial measurement unit provides attitude measurement, and like stability control, a setpoint of $\phi = \pm 90^\circ$ is achievable by creating a net moment from an actuation imbalance of motor pair 3 and 4. Additionally, actuating a single motor can position the ATR in perching mode provided the moment provided by the motor, k_M , is not too large to affect rotation about another axis. The ATR can also be equipped with a grasping mechanism in the hollow axle that enables the robot to grip onto vertical surfaces like netting, textiles, or even small tree branches in future designs. This mechanism could possibly be actuated with a shape memory alloy (SMA). Shape memory alloys are a special type of metal usually comprised of a Nickel-Titanium alloy, that when heated deforms and when cooled, returns to its original shape. For example, a simple shortening wire actuator could be used to actuate a compliant grasping mechanism for perching, and when the robot transitions to a locomotive mode, un-activated. Due to the ATR's small mass, future perching mechanisms need not be extremely sizable.

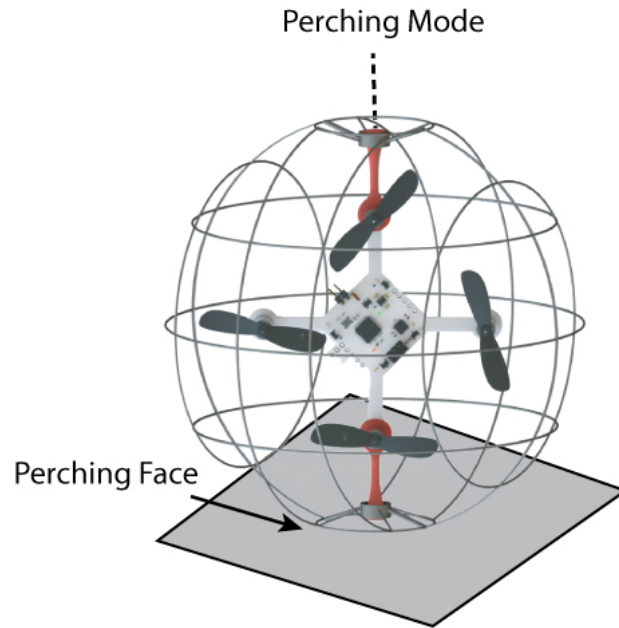


Figure 4.4: The ATR system in perching mode. The system can achieve passive stability by rolling onto the flat surfaces located at the ends of the robot.

4.9 Summary

This chapter presented the governing equations of motion of the ATR which are utilized in the simulation and design of the robot. Preliminaries such as coordinate transformations and the general Newton-Euler equations were discussed, followed by an in-depth model of the system dynamics in both modes of locomotion.

Chapter 5

Prototype Characterization

This chapter explains how the prototype ATR performance was measured to validate the expected performance from design and modeling. In Section 5.1, the motor constants k_T , and k_m are determined and power consumption is measured over the range of motor capabilities. In Section 5.2, the physical system parameters such as mass and inertia are determined. In Section 5.3, the synthesized attitude controller is tested and system performance is quantified.

5.1 Motor Characterization

A custom test stand, shown in Fig. 5.1 was fabricated to characterize the thrust, power consumed, current draw, and battery voltage as a function of motor angular velocity. The test stand is constructed from 6061 Aluminum in a manner to transfer thrust from the motors to the gram scale without propeller interference.

The "L" shaped design is constructed with equal length arms to eliminate geometric multiplication of force, and is fixed to the test bench with a low friction bearing. Additionally, measurements are made in a quasi-static manner to reduce any frictional effects from the bearing. A power resistor is fitted to the platform to measure

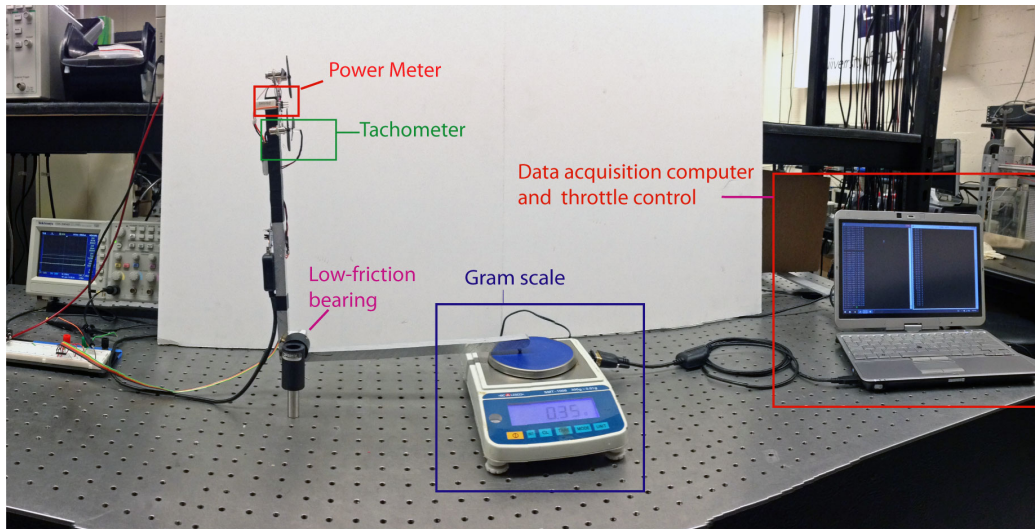


Figure 5.1: Motor thrust characterization test stand with tachometer, power meter, gram scale for thrust measurement, and serial communication link for data acquisition and throttle commands to the platform.

current draw, and battery voltage is measured and transferred to the data-acquisition computer via the microcontroller analog-to-digital port for power consumption calculations. The test stand also measures propeller angular velocity with an infrared emitter and detector circuit, shown in Fig. 5.1. When the detector is blocked from the emitter by the propeller, a change in detector voltage is noted and propeller frequency can be measured with an oscilloscope. The tachometer accurately and consistently measures propeller angular velocity through the range of motor angular velocity in this test (26,000 rpm). The thrust is characterized in Fig. 5.2(a) over the full range of throttle inputs and can be used to determine the motor thrust constant, k_T . The motor moment constant, k_M is characterized on a 6 axis force and torque transducer (ATI Industrial Automation Nano17, not pictured).

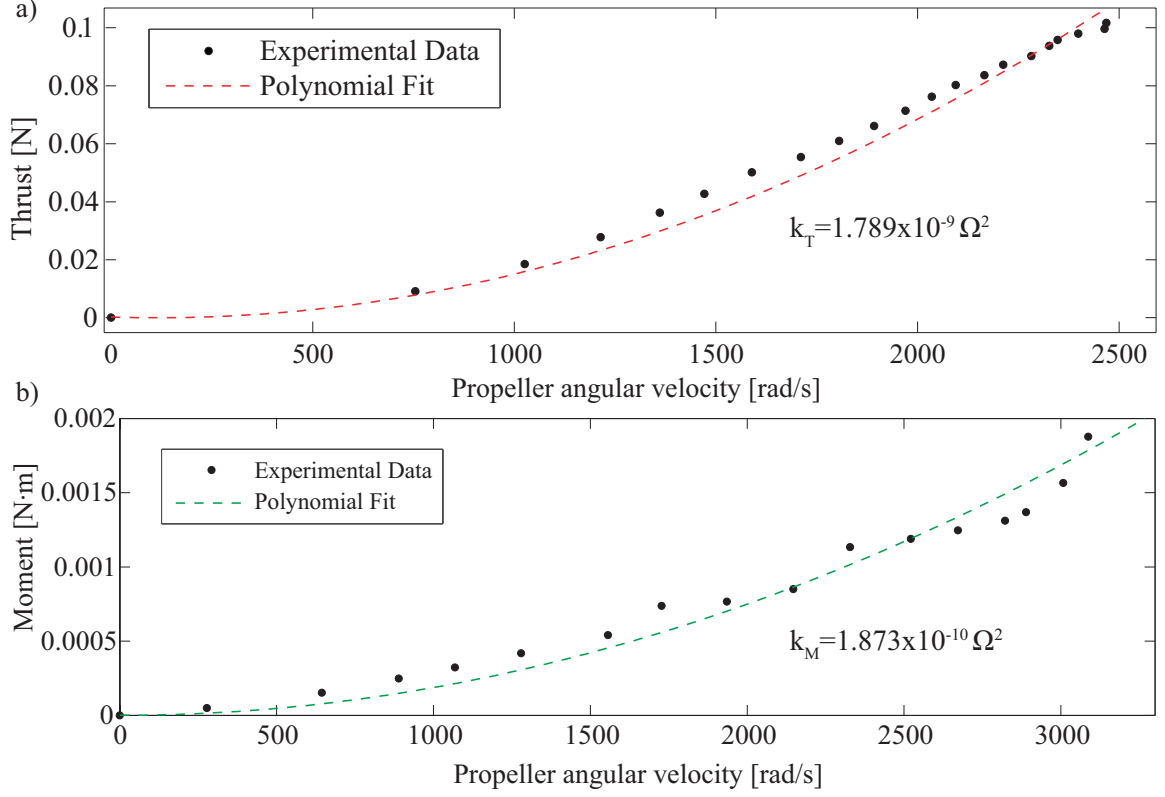


Figure 5.2: a) Motor thrust and b) moment constant characterization with second order polynomial fit.

5.2 Physical System Characterization

The system inertias, masses, and physical parameters such as arm length and center of mass are estimated using a reliable SolidWorks model and high-precision scale and are reported in Table 5.1. It is important to note that while the quadcopter system inertias in the \hat{b}_x and \hat{b}_y are nearly identical, the addition of the spherical exoskeleton results in system inertias that are not equal, complicating the controller design as the motion is highly coupled.

Table 5.1: ATR system parameters

System Parameter	Value
Micro-quadcopter mass	28.7 g
Spherical exoskeleton mass	6.54 g
Micro-quadcopter inertia	[11944, 11998, 22480] g·mm ²
Spherical exoskeleton inertia	[17760, 16476, 17760] g·mm ²
Rotor inertia	21.35 g·mm ²
Micro-quadcopter arm length, l	40 mm
Motor thrust constant, k_T	$1.761 \times 10^{-8} \frac{\text{N} \cdot \text{s}}{\text{rad}}$
Motor moment constant, k_M	$1.873 \times 10^{-10} \frac{\text{N} \cdot \text{m} \cdot \text{s}}{\text{rad}}$
Exoskeleton ring radii $[r_{1,2,3,4}]$	[75.0, 68.6, 44.8, 16.5]
Exoskeleton ring spacing, q	30 mm
Bearing shaft diameter, D	0.125 in
Coefficient of bearing friction, μ_b	0.15

5.3 Controller Performance

A simulation of the system dynamics in closed loop is used to estimate initial gains K_p , K_i , and K_d for each axis. To experimentally tune the gains, the quadcopter is fitted to a constraining test stand that isolates each axis from any other movement, and an iterative process is used on both the test stand and in-air flight for acceptable controller performance. A step response is generated for the derived model and compared to experimental results as shown in Fig. 5.3(a). The experimental platform is fixed on a test stand with one degree of freedom about the pitch axis for testing. While this method effectively isolates one axis for testing, the results are not an exact representation of the flight dynamics as translation and orientation are highly coupled. It is noted that the model does not address aerodynamic effects such as propeller wash, drag, and rotor dynamics, which are known to have an impact

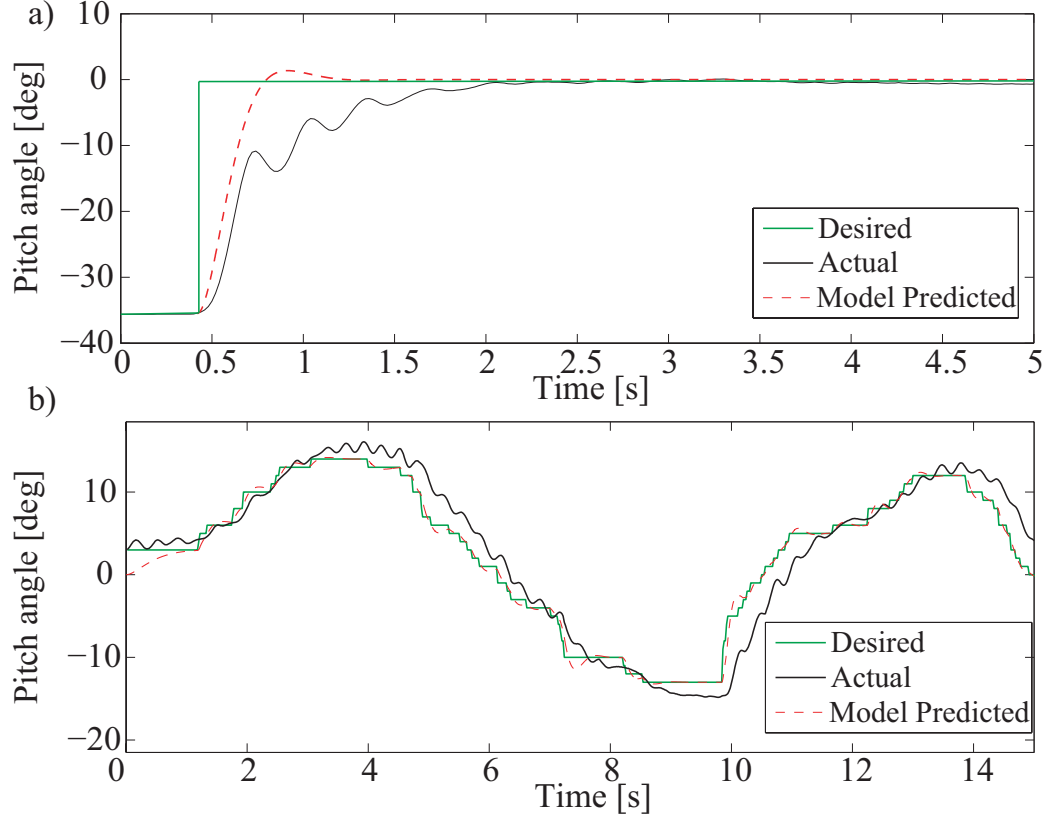


Figure 5.3: Measured system response due to: a) Step response and b) input tracking. The solid green line represents the desired pitch angle, dashed line represents the simulated model response, and the solid black line represents the experimental response on 1-DOF test stand.

on overall system performance [31]. The platform's ability to accurately track an input is shown in Fig. 5.3(b), with results similar to the step response. The RMS error for input tracking over a 15 second test is 1.47 percent. The controller integral output contains a maximum saturation to prevent integral controller windup. The error signal to the derivative portion of the controller is passed through a first order low-pass filter with cutoff frequency 20Hz to help remove sharp spikes in the signal from motor vibration and noise. The final controller gains are $K_p = 0.26$, $K_i = 0.12$, and $K_d = 0.04$.

Chapter 6

Simulation

Upon developing the governing equations of motion and performing the system characterization, the system is simulated in MATLAB Simulink to validate the model with experimental data and generate open-loop trajectories suitable for each mode of transportation. The inputs to the model are the physical system parameters given in Table 5.1, initial position, orientation, velocity, and angular velocity, controller parameters, and desired orientation and altitude.

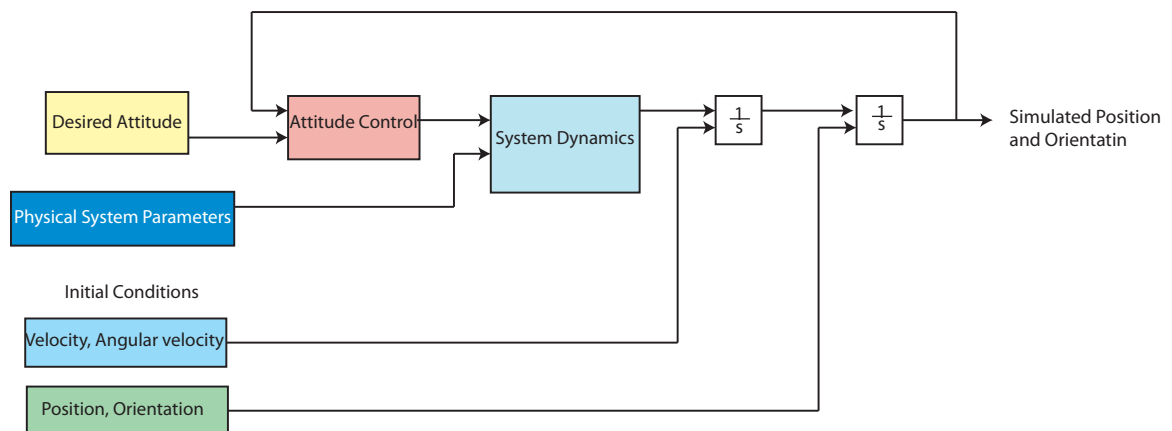


Figure 6.1: Simulation block diagram.

The simulation is solved with the Bogacki-Shampine Formula (MATLAB ODE3) at a fixed-step time interval of 0.01 seconds, which is identical to the sampling time

of the microcontroller control software. A general schematic for simulation is given in Fig. 6.1, and the detailed Simulink block diagram is given in Appendix C. Each block in Fig. 6.1 is described by one or several MATLAB files and can be easily incorporated into other simulators. The simulation starts with the initial state taken from the initial condition block. The desired and actual position are input to the control block, which generates the necessary motor outputs for attitude correction. The System dynamics block receives the motor inputs and outputs the system accelerations \ddot{x} , \ddot{y} , \ddot{z} , $\ddot{\phi}$, $\ddot{\theta}$, $\ddot{\psi}$. The system accelerations are then twice integrated with their respective initial conditions to yield the system output. Separate simulations are developed for both aerial and terrestrial locomotion, but final conditions from either model can easily be used as initial conditions to the other model for simulations that include both rolling and flying. The system positions, orientations, velocities, and accelerations are recorded for use in animations of the simulation for easy visualization of the system dynamics.

For example, in Fig. 6.2 the ATR is simulated to maintain a setpoint pitch angle and roll across the ground for a distance and then commanded to take off into aerial mode.

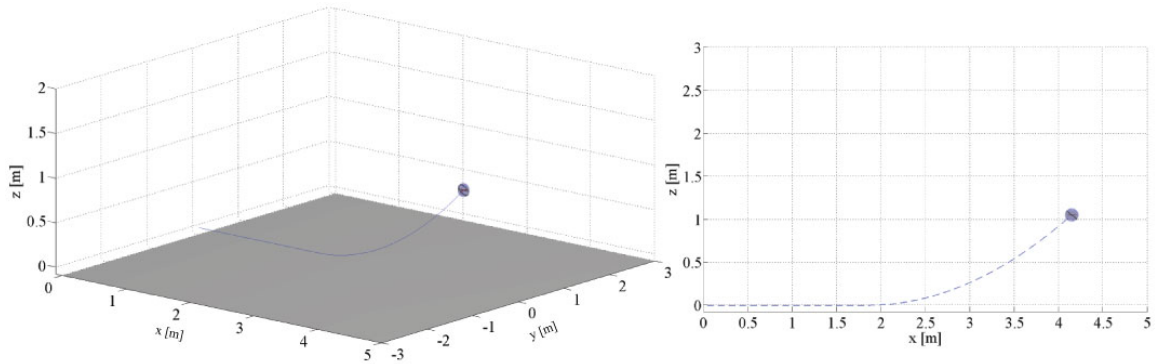


Figure 6.2: Simulation of the ATR rolling on the ground and transitioning to aerial mode.

Similarly, the ATR can be simulated to take-off vertically, fly forward, and land, as

shown in Fig. 6.3. For the purpose of demonstration, the ATR provides a correctional thrust to help eliminate its initial velocity before landing and rolling.

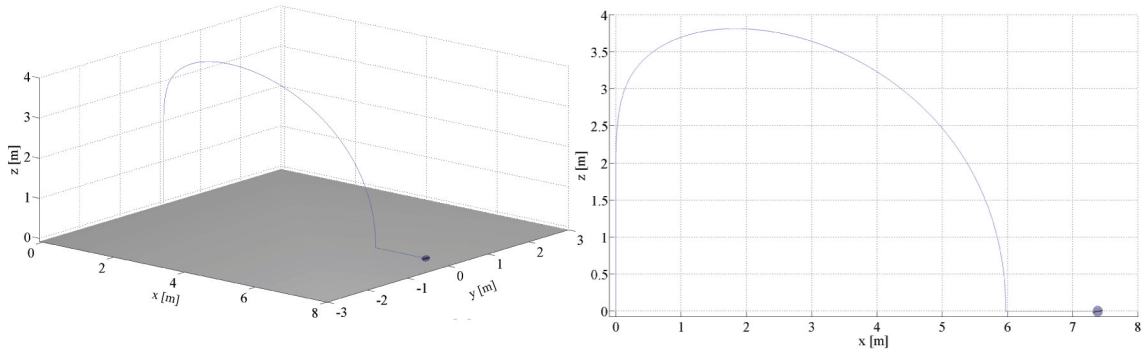


Figure 6.3: Simulation of the ATR performing a vertical take-off, flying forward, correcting its angle to minimize its velocity, and rolling on the ground after landing.

Additionally, a path for complex terrestrial locomotion or aerial flight into a small opening, shown in Fig. 6.4 can be simulated. A practical application of this type of motion is a situation where the ATR must interrogate a target on the ground and then fly to another target located in a hard to reach space, like an air duct.

6.1 Summary

The simulation results, namely required motor angular velocities and system orientation, can be recorded and sent to the experimental ATR for open loop trajectory generation.

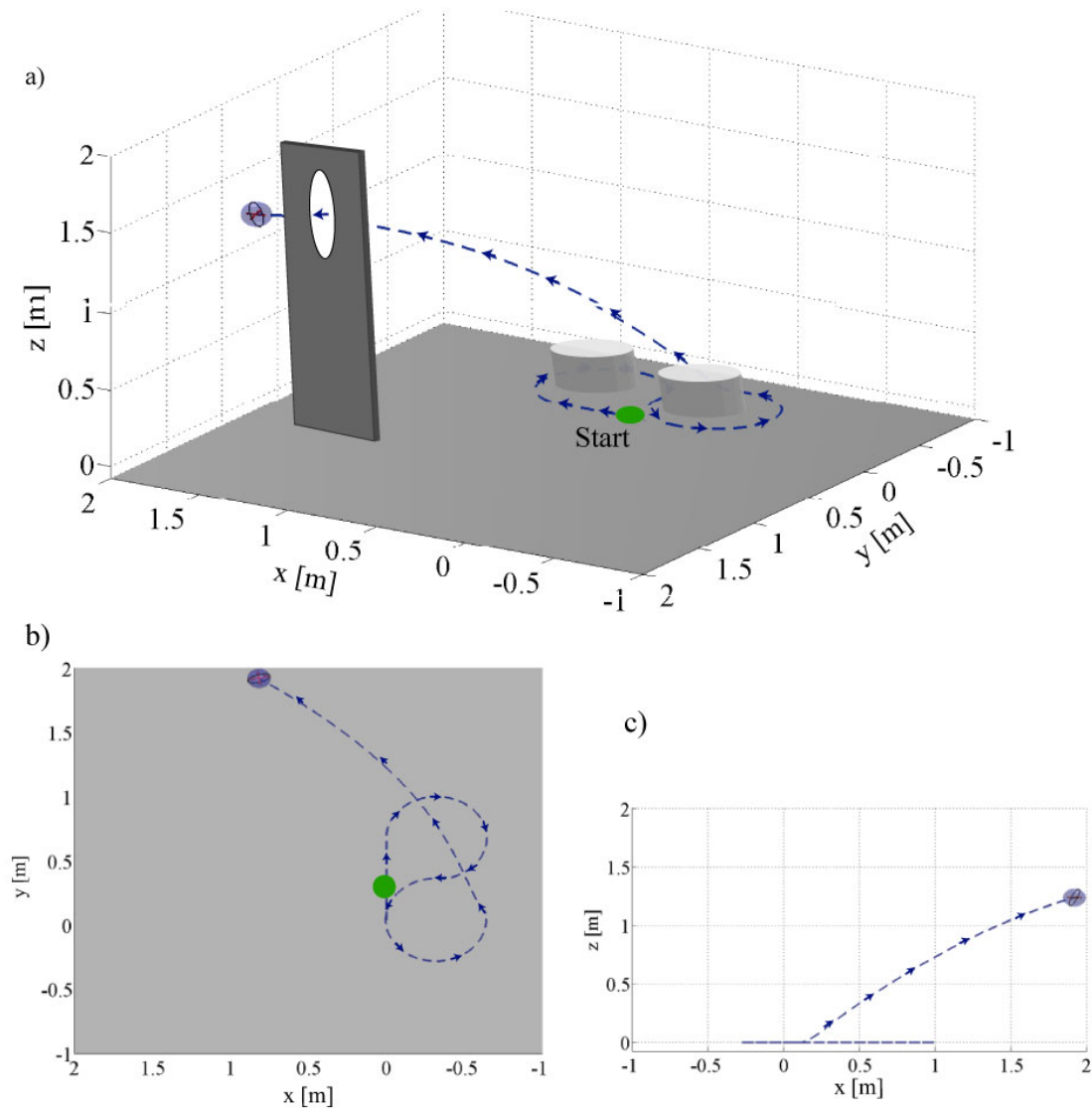


Figure 6.4: Simulation of the ATR rolling on the ground to maneuver through obstacles and then transitioning to flying mode through a small opening.

Chapter 7

Experimental Results and Discussion

In this chapter, the experimental results of the ATR prototype in both aerial and terrestrial mode are presented. Using the simulation results from Chapter 6, open loop trajectories are tested for both modes of locomotion. The ATR is shown to require significantly less power in terrestrial mode than in flying mode. When compared to a traditional quadcopter platform, the ATR is only slightly less efficient in aerial mode.

7.1 Autonomous Hover

The fabricated ATR was evaluated in an indoor environment with foam ground and netted walls to protect the ATR from impact. The ATR was given an initial motor input pulse for take-off, followed by a constant motor input required for hover. The ATR prototype is capable of hover at 72 percent throttle and 14.3 W power demand. It is important to note that this power consumption figure is not a direct measurement, it is calculated from the platform characterization results in Chapter 5 as the addition of a power monitor to the experimental system would increase the system mass and

decrease the robot efficiency. The ATR drifts approximately 8 ft over the duration of the 15 second test with no user lateral input. As shown in Fig. 7.1 the ATR can also be thrown into the air with an initial condition and achieve stability to hover. The ATR exhibits a significant amount of drift while hovering, and is likely due to small controller corrections in attitude to combat aerodynamic forces such as propeller wash, and air currents in the test bed from sources such as fans, open doors, etc. A motion capture system or other similar external position sensing and control method could help reduce these effects.

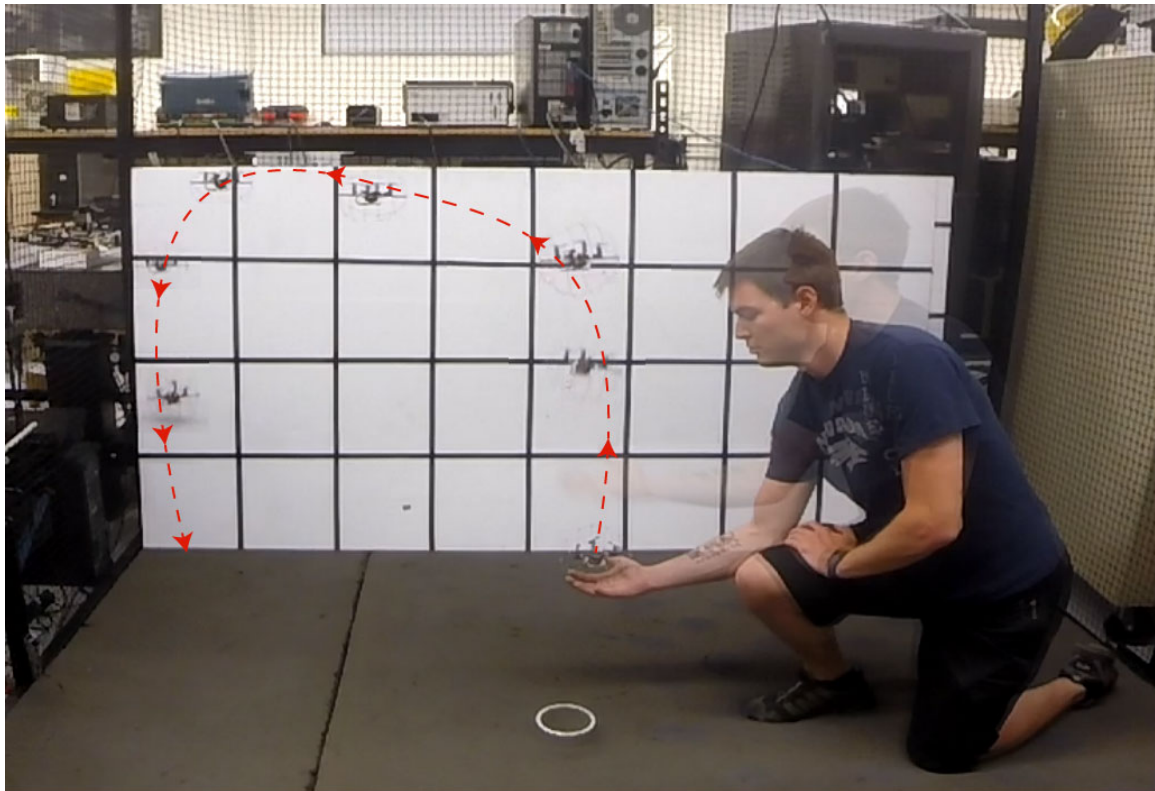


Figure 7.1: The ATR can be thrown into the air and self-stabilize to hover. The duration of this test is approximately 14 seconds.

7.2 Open Loop Aerial Trajectory Tracking

Next, the ATR's ability to follow a simulation generated trajectory was tested. The desired trajectory is as follows:

- 1) Autonomous take-off. Throttle to 93 percent, maintain heading and stabilize.
- 2) Hover in place for 0.5 seconds.
- 3) Pitch forward for 0.35 seconds, $\theta_{desired} = -20^\circ$. Move forward.
- 4) Pitch backward for 0.35 seconds, $\theta_{desired} = 20^\circ$. Reduce velocity.
- 5) Roll right for 0.35 seconds, $\phi_{desired} = 20^\circ$. Move right.
- 6) Roll left for 0.35 seconds, $\phi_{desired} = -20^\circ$. Reduce velocity.
- 7) Pitch backward for 0.35 seconds, $\theta_{desired} = 20^\circ$. Move backward.
- 8) Pitch forward for 0.35 seconds, $\theta_{desired} = -20^\circ$. Reduce velocity.
- 9) Roll left for 0.35 seconds, $\phi_{desired} = -20^\circ$. Move left.
- 10) Roll right for 0.35 seconds, $\phi_{desired} = 20^\circ$. Reduce velocity.
- 11) Hover in place

The resulting trajectory is a square pattern of approximately 1.7 m. Fig. 7.2 shows the resulting model and experimental trajectory scaled for the image size. The ATR is able to track the desired trajectory reasonably well, but without position sensing, open loop control does not provide good trajectory tracking. Most notably, the ATR does not traverse to each corner of the desired path. Additionally, the ATR does not return to the desired position at the end of the test, leaving the square pattern open. To quantify this error, a measure is developed to estimate the open loop tracking performance where the error in final position, δ_f , is compared to the total path distance, δ_p and is given by the path error,

$$e_r = \frac{\delta_f}{\delta_p} \quad (7.1)$$

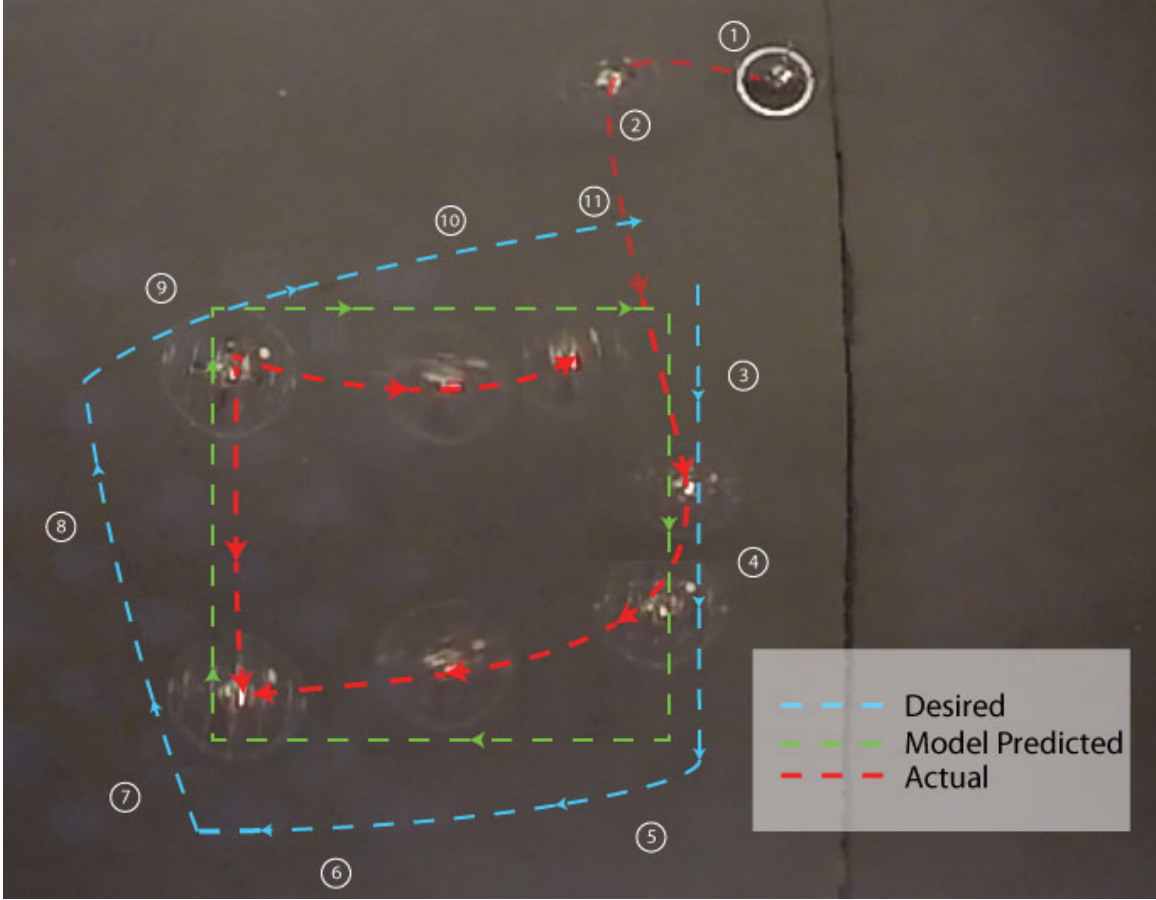


Figure 7.2: Open loop trajectory tracking for a desired square pattern.

Since the collected data is in the form of a single aerial video and the pixel to robot height relationship is unknown, the resulting error calculation is limited. The robot path is traced in video editing software, and a pixel distance relationship is calculated from known features in the image. The robot travels a path of approximately 6.45 m and falls short of the desired path end point by 0.2 m. The resulting path error, $e_r = .031$, indicates that the final position error is small, but Fig. 7.2 indicates that the path traveled by the ATR is not exactly the desired path. For example, the ATR misses the corner between steps 4 and 5 by approximately 0.38 m.

Both the formulated model and the experimental system drift when turning, likely

due to the fact that they both have a component of velocity in the direction previously traversed. It is estimated that the trajectory tracking would be drastically improved with a motion capture system or similar sensing method. It is also noted that the ATR drifts considerably on take-off. This can be due to a number of phenomena not captured by the model. For example, the model assumes zero initial conditions on take-off, which may not be true. Also, the formulated model does not capture any aerodynamic effects that may be present at low altitude or during acceleration of the platform from the ground to hover.

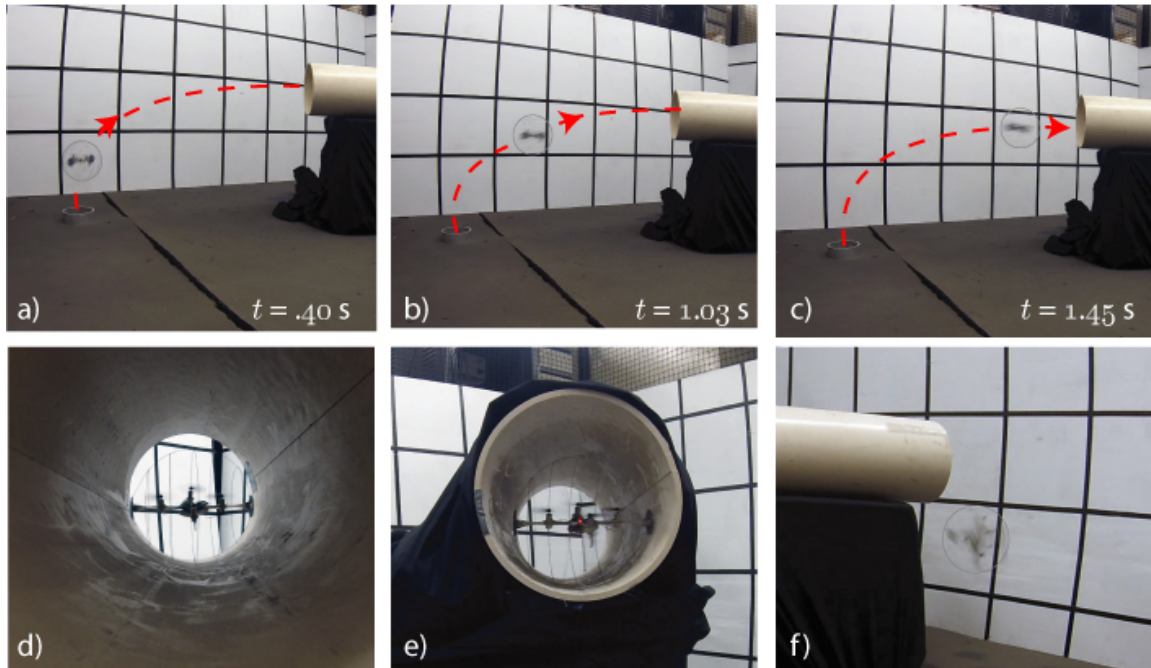


Figure 7.3: Flying behavior of the ATR into a tube, then rolling through the tube and out.

An additional experiment was performed to demonstrate potential application of the ATR. During this test, the ATR was remotely controlled by a joystick and guided into a 7 in diameter tube, shown in Fig. 7.3. The ATR then rolls through the tube and exits. This experiment was performed piecewise as the manual operator is not capable of the advanced control needed for transition between modes and attitude stabilization.

7.2.1 Terrestrial Locomotion

The ATR was then tested to explore the robot's capabilities in tracking terrestrial trajectories in open loop. First, a trajectory is generated for a roll-to-fly maneuver where the robot rolls in a straight line, stops, and ideally takes off vertically. The desired trajectory is as follows:

- 1) Activate motors to a base throttle and stabilize while on the ground
- 2) Pitch forward for 0.8 seconds, $\theta_{desired} = -9^\circ$. Move forward.
- 3) Pitch backward for 0.3 seconds, $\theta_{desired} = 30^\circ$. Reduce velocity.
- 4) Increase motors to full throttle for 0.5 seconds, $\theta_{desired} = 0^\circ$. Take-off.
- 5) Decrease motor command to hover in place for 2 seconds.

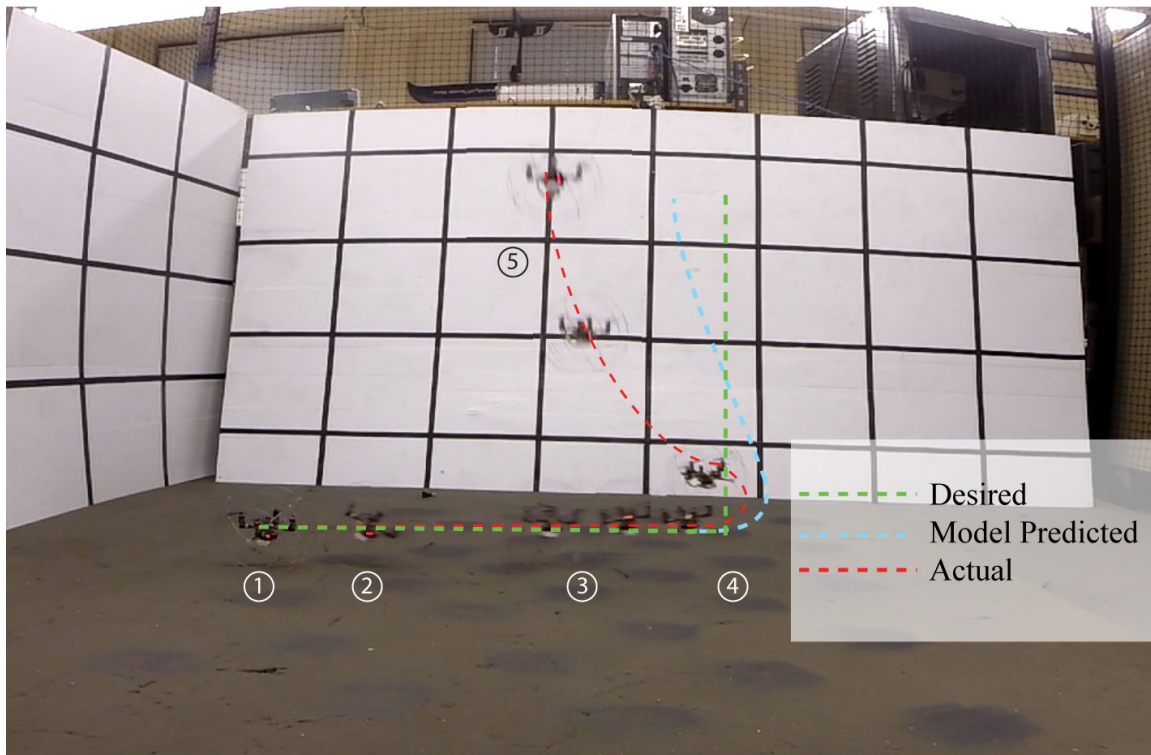


Figure 7.4: Open loop experimental results for a desired roll, stop, fly trajectory.

The experimental trajectory agrees well with model result and correlates to a high degree with the desired trajectory as shown in Fig. 7.4. Application of Eq.

(7.1) indicates an ending position error, $\delta_f = 0.28$ m, and corresponding path error $e_r = 0.16$. The ending position error is also the maximum path error and is good representation of the open loop tracking error. The ATR successfully removes much of its initial ground velocity prior to take off, but an existing velocity and orientation which are not ideally zero result in a negative \hat{e}_x displacement before the robot can equilibrate in hover. The model result exhibits the same phenomena, although to a lesser degree. It is also noticed during the experiment that any deviation from level ground surface provides a disturbance to the ATR's attitude that may alter the desired trajectory. Development of a more robust attitude controller can help reduce this problem, as well as improved exoskeleton design reducing transmission of disturbances to the system.

Next, the turning capabilities of the ATR are tested in a single turn maneuver. Since the designed controller is unable to compensate for large disturbances and the exoskeleton compliance affects turning dynamics, more advanced trajectories are difficult to test. The ATR is programmed to activate motors to a low throttle command, righting the platform, and when stable, the ATR is programmed to a setpoint pitch angle, $\theta = 9^\circ$, for terrestrial thrust, and roll angle, $\phi = 8^\circ$, to position the ATR onto the first set of rings. The desired turning radius is $r_t = 32.1$ cm.

The ATR immediately achieves the set-point roll and pitch angles and begins to turn in a circular manner, but is soon thrown off course and does not track the trajectory accurately. While the tracking performance is poor, the ATR is still capable of performing a turn while rolling. the performance is most notably affected by poor attitude control about the roll and pitch axes when the ball is rolling. Disturbances in the form of ground imperfections and the fact that the exoskeleton is not perfectly round while rolling saturate the control effort, and the ATR is unable to maintain attitude. Additionally, compliance in the exoskeleton is not captured by the for-

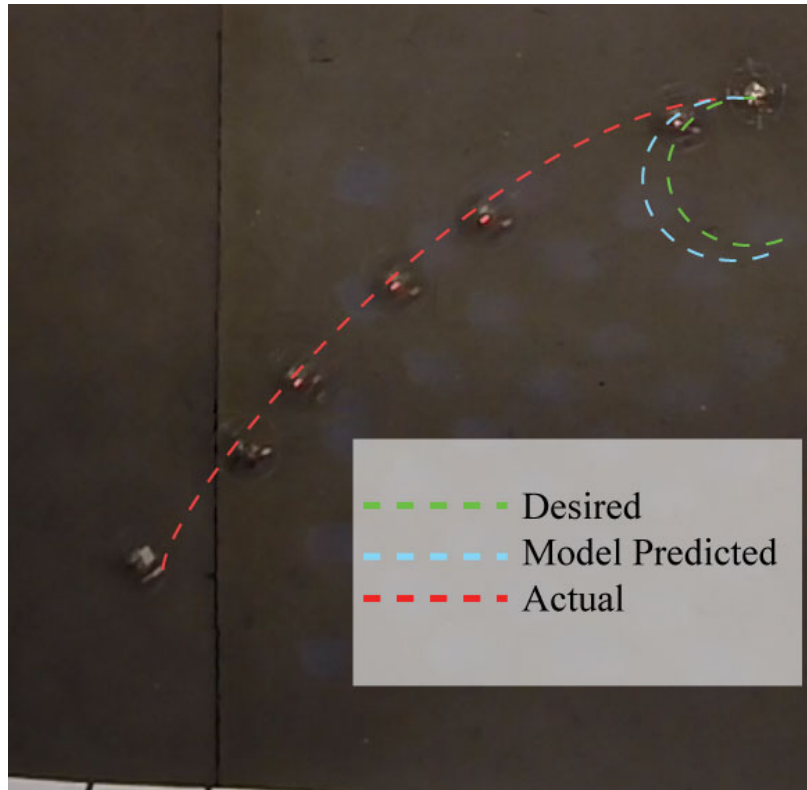


Figure 7.5: Open loop experimental results for a desired roll and turn maneuver

mulated model, which may contribute to non-circular patterns as the rings deform during rolling. It is also difficult to determine whether the path normal forces that keep the robot turning in a circular manner are exceeded. If so, the behavior shown is expected as the robot slips along the surface while rolling.

7.3 Energy Analysis

To experimentally validate the increased efficiency of the ATR, the throttle command required to roll and fly in a straight line was measured for a set distance and the time was recorded. From the platform characterization in Section 5, and translational velocity recorded during experimentation, the time and distance to exhaust a 350 mAh battery was calculated for each respective mode. In rolling mode, the ATR can travel

at 2.46 m/s with a current demand of 1.73 A and has a range of 1.79 km for 12.14 min. Comparatively, in flying mode, the ATR travels at 1.62 m/s with a current demand of 4.35 A and has a range of 469 m for 4.82 min, showing that the ATR's rolling mode is 3.66 times more efficient than flying mode. For the same aerial velocity, an unmodified quadcopter, weighing 28.2 g, can travel 652 m for 6.71 min, which is 39 percent more efficient than the ATR in flying mode, but the ATR's rolling mode is 2.63 times more efficient than the unmodified flying-only platform.

7.4 Summary

This chapter presented the results of experiments using the developed ATR and synthesized control techniques. The robot is capable of tracking both aerial and terrestrial open loop trajectories. Additionally, the robot is able to save considerable energy in terrestrial locomotion mode, resulting in increased range and operating time.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This thesis presented the design, analysis, control, and implementation of a novel, ultra-lightweight spherical aerial-terrestrial robot (ATR). The design of the ATR is comprised of a micro-quadcopter encased in a 6 inch spherical exoskeleton. The ATR has the ability to fly through the air or roll on the ground, for various applications such as inspection, surveillance, mapping, and search and rescue. The developed ATR is currently the smallest quadcopter-based hybrid locomotive robot and is especially suited for urban applications in tight spaces that require a high degree of maneuverability.

The dynamic system model is presented for both modes of locomotion, and then utilized to generate control parameters and inputs to demonstrate hybrid locomotion. Experimental results show that the synthesized flight controller is capable of stabilizing the attitude of the ATR in both modes of locomotion, and input tracking is 1.47 percent RMS error in aerial mode. The ATR is experimentally tested and proves capable in autonomous modes such as throw-to-hover, open loop aerial trajectory tracking, roll-to-fly maneuvers, and demonstrates the capability to perform a turn

while rolling. The ATR is also tested in a practical environment and is able to fly into a narrow opening, roll through a tube, and exit to fly out of the tube. Open loop tracking performance is quantified by the path error and maximum error from desired path. In aerial mode, the ATR tracks a square trajectory with $e_r = 0.031$ over the 6.45 m path and has a maximum error of 0.38 m. In terrestrial mode, the ATR tracks a rolling-to fly trajectory with $e_r = 0.16$ and a maximum error of 0.28 m. Turning performance is poor and the ATR exhibits severe under-steer when turning, which could be due to exoskeleton compliance, disturbances in the form of ground imperfections, or poor attitude control during ground maneuvers.

The general deviation from desired paths in both terrestrial and aerial trajectories is likely due to effects not captured by the formulated model such as: aerodynamic effects, motor thrust dynamics and nonlinearities, system compliance, and external disturbances. Results from open-loop trajectory tracking can be improved with the addition of an external localization system used for position feedback control. Then, instead of controlling only the attitude of the ATR, precise position control can be implemented.

It was estimated that the ATR can roll along the ground for over 12 minutes and cover the distance of 1.7 km, or it can fly for 4.82 minutes and travel 469 m, on a single 350 mAh battery. Compared to a traditional flying-only robot, the ATR in rolling mode is 2.63 times more efficient, and in flying mode is only 39 percent less efficient. The ATR can transition seamlessly between operation modes and is capable of navigating through constrained spaces.

8.2 Future Work

The ATR is a highly capable aerial-terrestrial robot and future development is encouraged by the author. The control of the ATR is both an interesting and complex

control problem, and further development in this area can enhance the capabilities of the ATR. Specifically, position control can increase the accuracy of the ATR's trajectory tracking.

Improvements in the fabrication of the exoskeleton can enhance the efficiency of the ATR in aerial mode. For example, a rigid composite exoskeleton could both lighten the ATR and improve terrestrial performance. Additionally, the quadcopter platform can be designed as part of the exoskeleton axle, further reducing the system mass and lowering the quadcopter center of mass that can enhance the system's passive stability.

Bibliography

- [1] P. Rudol and P. Doherty, "Human body detection and geolocalization for uav search and rescue missions using color and thermal imagery," in *Aerospace Conference, 2008 IEEE*, 2008, pp. 1–8.
- [2] M. A. Goodrich, B. S. Morse, D. Gerhardt, J. L. Cooper, M. Quigley, J. A. Adams, and C. Humphrey, "Supporting wilderness search and rescue using a camera-equipped mini uav," *Journal of Field Robotics*, vol. 25, no. 1-2, pp. 89–110, 2008.
- [3] C. Luo, A. Espinosa, D. Pranantha, and A. De Gloria, "Multi-robot search and rescue team," in *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, 2011, pp. 296–301.
- [4] M. Bloesch, S. Weiss, D. Scaramuzza, and R. Siegwart, "Vision based mav navigation in unknown and unstructured environments," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010, Conference Proceedings, pp. 21–28.
- [5] I. Dryanovski, W. Morris, and J. Xiao, "An open-source pose estimation system for micro-air vehicles," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, pp. 4449–4454.
- [6] W. R. Davis, B. B. Kosicki, D. M. Boroson, and D. Kostishack, "Micro air vehicles for optical surveillance," *Lincoln Laboratory Journal*, vol. 9, no. 2, pp. 197–214, 1996.
- [7] J. Berni, P. Zarco-Tejada, L. Suarez, and E. Fereres, "Thermal and narrowband multispectral remote sensing for vegetation monitoring from an unmanned aerial vehicle," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 47, no. 3, pp. 722–738, 2009.
- [8] S. Nebiker, A. Annen, M. Scherrer, and D. Oesch, "A light-weight multispectral sensor for micro uav opportunities for very high resolution airborne remote sensing," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 37, pp. 1193–2000, 2008.

- [9] T. Suzuki, D. Miyoshi, J. Meguro, Y. Amano, T. Hashizume, K. Sato, and J. Takiguchi, "Real-time hazard map generation using small unmanned aerial vehicle," in *SICE Annual Conference, 2008*, 2008, pp. 443–446.
- [10] V. Kumar and N. Michael, "Opportunities and challenges with autonomous micro aerial vehicles," *Int. J. of Robotics Research*, vol. 31, no. 11, pp. 1279 – 1291, 2012.
- [11] O. Defense. Terra-max ugv. [Online]. Available: <http://oshkoshdefense.com/technology-1/unmanned-ground-vehicle/>
- [12] C. Thorpe, M. Herbert, T. Kanade, and S. Shafter, "Toward autonomous driving: the cmu navlab. ii. architecture and systems," *IEEE expert*, vol. 6, no. 4, pp. 44–52, 1991.
- [13] E. D. Dickmanns and A. Zapp, "Autonomous high speed road vehicle guidance by computer vision," in *International Federation of Automatic Control. World Congress (10th). Automatic control: world congress.*, vol. 1, 1988.
- [14] J. Markoff, "Google cars drive themselves, in traffic," *The New York Times*, vol. 10, p. A1, 2010.
- [15] I. General Atomics. Predator uas. [Online]. Available: <http://www.ga-asi.com/products/aircraft/predator.php>
- [16] I. Insitu. Scaneagle-le capabilities. [Online]. Available: <http://www.insitu.com/systems/scaneagle/capabilities>
- [17] I. AeroVironment, "Uas: Rq-b raven." [Online]. Available: <http://www.avinc.com/uas/smalluas/raven/>
- [18] A. Bachrach, S. Prentice, R. He, and N. Roy, "Rangerobust autonomous navigation in gps-denied environments," *Journal of Field Robotics*, vol. 28, no. 5, pp. 644–666, 2011. [Online]. Available: <http://dx.doi.org/10.1002/rob.20400>
- [19] J. Colorado, A. Barrientos, A. Martinez, B. Lafaverge, and J. Valente, "Mini-quadrotor attitude control based on hybrid backstepping & frenet-serret theory," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1617–1622.
- [20] R. J. Bachmann, F. J. Boria, R. Vaidyanathan, P. G. Ifju, and R. D. Quinn, "A biologically inspired micro-vehicle capable of aerial and terrestrial locomotion," *Mechanism and Machine Theory*, vol. 44, no. 3, pp. 513–526, 2009.
- [21] R. J. Bachmann, R. Vaidyanathan, and R. D. Quinn, "Drive train design enabling locomotion transition of a small hybrid air-land vehicle," in *Intelligent Robots and Systems, IROS 2009. IEEE/RSJ International Conference on*, 2009, Conference Proceedings, pp. 5647–5652.

- [22] K. Peterson, P. Birkmeyer, R. Dudley, and R. Fearing, “A wing-assisted running robot and implications for avian flight evolution,” *Bioinspiration & Biomimetics*, vol. 6, no. 4, p. 046008, 2011.
- [23] M. Kovac, J.-C. Zufferey, and D. Floreano, “Towards a self-deploying and gliding robot,” *Flying insects and robots*, p. 271, 2009.
- [24] A. Kalantari and M. Spenko, “Design and manufacturing of a walking quadrotor aerial vehicle,” in *ASME International Design Engineering Technical Conference*, 2012, pp. 1067–1072.
- [25] —, “Design and experimental validation of hytaq, a hybrid terrestrial and aerial quadrotor,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 2013, Conference Proceedings, pp. 4445–4450.
- [26] D. Mellinger, N. Michael, and V. Kumar, “Trajectory generation and control for precise aggressive maneuvers with quadrotors,” *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012.
- [27] S. Zingg, D. Scaramuzza, S. Weiss, and R. Siegwart, “Mav navigation through indoor corridors using optical flow,” in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 3361–3368.
- [28] S. Bouabdallah and R. Siegwart, “Full control of a quadrotor,” in *Intelligent robots and systems, 2007. IROS 2007. IEEE/RSJ international conference on*. IEEE, 2007, pp. 153–158.
- [29] S. Grzonka, G. Grisetti, and W. Burgard, “A fully autonomous indoor quadrotor,” *Robotics, IEEE Transactions on*, vol. 28, no. 1, pp. 90–100, 2012.
- [30] D. Mellinger, M. Shomin, and V. Kumar, “Control of quadrotors for robust perching and landing,” in *Proc. Int. Powered Lift Conf*, 2010, pp. 119–126.
- [31] S. Bouabdallah, P. Murrieri, and R. Siegwart, “Design and control of an indoor micro quadrotor,” in *Robotics and Automation, 2004. Proceedings. ICRA’04. 2004 IEEE International Conference on*, vol. 5. IEEE, 2004, Conference Proceedings, pp. 4393–4398.

Appendix A

Motor Characterization

motor command	Hz O-scope	RPM	Thrust [g]	Lift [mN]	Current [A]	Voltage [V]	POWA [W]	thrust [kg]	omega [rad/s]	thrust [N]
0	0	0	0	0	0	0	0	0	0	0
5	240.1	7203	3.704	36.33624	0.55323741	3.88	2.146561151	0.003704	754.2963961	0.036336
10	326.5	9795	7.538	73.94778	0.979101124	3.820786517	3.740936372	0.007538	1025.730001	0.073948
15	386.4	11592	11.344	111.28464	1.397446809	3.76893617	5.266887823	0.011344	1213.911401	0.111285
20	433.1	12993	14.778	144.97218	1.7375	3.732222222	6.484736111	0.014778	1360.623778	0.144972
25	468.3	14049	17.434	171.02754	2.0592	3.6916	7.60174272	0.017434	1471.20784	0.171028
30	505.7	15171	20.446	200.57526	2.348852459	3.66	8.5968	0.020446	1588.703405	0.200575
35	544.5	16335	22.602	221.72562	2.593563218	3.638275862	9.436098454	0.022602	1710.5972	0.221726
40	574.7	17241	24.877	244.0401	2.803414634	3.621219512	10.15177977	0.024877	1805.473298	0.244043
45	602.4	18072	26.96	264.4776	3.029078947	3.603421053	10.91504685	0.02696	1892.495415	0.264478
50	627	18810	29.12	285.6672	3.278113208	3.581792453	11.74152115	0.02912	1969.778594	0.285667
55	647.9	19437	31.098	305.07138	3.473442623	3.570491803	12.40189841	0.031098	2035.43788	0.305071
60	666.7	20001	32.72	320.9832	3.644234234	3.56036036	12.97478711	0.03272	2094.499822	0.320983
65	689.4	20682	34.136	334.87416	3.808285714	3.558285714	13.55096865	0.034136	2165.813975	0.334874
70	704.2	21126	35.59	349.1379	3.942142857	3.560178571	14.03473253	0.03559	2212.309547	0.349138
75	726.4	21792	36.808	361.08648	4.097582418	3.558351648	14.58063915	0.036808	2282.052904	0.361086
80	740.7	22221	38.255	375.28155	4.238552632	3.560855263	15.09287245	0.038255	2326.977679	0.375282
85	747.1	22413	39.078	383.35518	4.357716535	3.56015748	15.51415712	0.039078	2347.083871	0.383355
90	763.5	22905	39.978	392.18418	4.448787879	3.560757576	15.84105514	0.039978	2398.605991	0.392184
95	784.3	23529	40.633	398.613	4.553150685	3.566027397	16.23666009	0.040633	2463.951118	0.39861
100	785.5	23529	41.495	407.06595	4.65037037	3.571851852	16.61043402	0.041495	2467.721029	0.407066

Appendix B

MATLAB .m Files

B.1 GUI

breaklines

```

1 function varargout = control_GUI(varargin)
% CONTROL_GUI MATLAB code for control_GUI.fig

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn',   @control_GUI_OpeningFcn, ...
                  'gui_OutputFcn',    @control_GUI_OutputFcn, ...
                  'gui_LayoutFcn',    [], ...
11                  'gui_Callback',    []);

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
21

% --- Executes just before control_GUI is made visible.
function control_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
% varargin   command line arguments to control_GUI (see VARARGIN)

% Choose default command line output for control_GUI
31 global joy rtrim ptrim bl2 br2% establish global variables

```

```

-----
handles.output = hObject;
%create joystick object and assign axes
-----%
bl2=0;
br2=0;
joy=vrjoystick(1);
throttle=-axis(joy,2);
yaw=-axis(joy,3);
roll=axis(joy,4);
pitch=axis(joy,5);
41 %set all trims = 0
rtrim=0;
ptrim=0;
%create timer for sending and receiving data
-----%
handles.timer = timer(...
    'ExecutionMode', 'fixedRate', ...           % Run timer repeatedly
    'Period', .02, ...                         % Initial period is 0.02
    sec.
    'TimerFcn', {@update_data,hObject}); % Specify callback function
%create timer for updating plot data of the LIDAR
-----%
51 handles.timer2 = timer(...
    'ExecutionMode', 'fixedRate', ...           % Run timer repeatedly
    'Period', .02, ...                         % Initial period is 0.02
    sec.
    'TimerFcn', {@update_outputs,hObject}); % Specify callback function
%plot initial input values
-----%
handles.bar_throttle = bar(handles.ax_throttle,throttle,'g','BarWidth'
,1);
handles.bar_roll = barh(handles.ax_roll,roll,'g','BarWidth',1);
handles.bar_pitch = bar(handles.ax_pitch,pitch,'g','BarWidth',1);
handles.bar_yaw = barh(handles.ax_yaw,yaw,'g','BarWidth',1);
% Set Axes limits and colors
-----%
61 set(handles.ax_throttle,'YLim',[0 100]);
set(handles.ax_throttle,'XColor',[0.8 0.8 0.8]);
set(handles.ax_throttle,'YColor',[0.8 0.8 0.8]);
set(handles.ax_roll,'XLim',[00 100]);
set(handles.ax_roll,'XColor',[0.8 0.8 0.8]);
set(handles.ax_roll,'YColor',[0.8 0.8 0.8]);
set(handles.ax_pitch,'YLim',[00 100]);
set(handles.ax_pitch,'XColor',[0.8 0.8 0.8]);
set(handles.ax_pitch,'YColor',[0.8 0.8 0.8]);
set(handles.ax_yaw,'XLim',[00 100]);
set(handles.ax_yaw,'XColor',[0.8 0.8 0.8]);
71 set(handles.ax_yaw,'YColor',[0.8 0.8 0.8]);
set(handles.comport,'Value',1);
%Initialize Buttons to not depressed
-----%

```

```

set(handles.XBut1_A,'Value',0);
set(handles.XBut2_B,'Value',0);
set(handles.XBut3_X,'Value',0);
set(handles.XBut4_Y,'Value',0);
set(handles.connect,'Enable','off')
set(handles.stop,'Enable','on')
%initialize Trim
-----%
81 set(handles.ptrimt,'String','Pitch Trim:0');
   set(handles.rtrimt,'String','Roll Trim:0');
   set(handles.ytrimt,'String','Yaw Trim:0');
%-----END CJ INIT CODE
-----%
% Update handles structure
guidata(hObject,handles);%ptcloud
% --- Outputs from this function are returned to the command line.
function varargout = control_GUI_OutputFcn(hObject, eventdata, handles)
% varargout    cell array for returning output args (see VARARGOUT);
% hObject      handle to figure
91 % eventdata   reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in connect.
function connect_Callback(hObject, eventdata, handles)
% hObject      handle to connect (see GCBO)
101 % eventdata   reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
% Only start timer if it is not running
global ctr ctr1
%Start The Data Send/Receive Timer at 50Hz
if strcmp(get(handles.timer,'Running'),'off')
    start(handles.timer);
end
%Start The Plot Lidar Timer at 4Hz or whatever is defined in startup
script
if strcmp(get(handles.timer2,'Running'),'off')
111     start(handles.timer2);
end
ctr=0;
ctr1=0;

% --- Executes on button press in stop.
function stop_Callback(hObject, eventdata, handles)
% hObject      handle to stop (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
121 global obj1
    if strcmp(get(handles.timer,'Running'),'on')

```

```

        stop(handles.timer);
    end
    if strcmp(get(handles.timer2, 'Running'), 'on')
        stop(handles.timer);
    end
    set(handles.stop, 'Enable', 'off')
    %-----
    %Close COM Port
131 % Disconnect from instrument object, obj1.
    fclose(obj1);
    delete(obj1);
    delete(instrfindall)

    function update_outputs(hObject,eventdata,hfigure)
        handles = guidata(hfigure);

    function update_data(hObject,eventdata,hfigure)
141 global joy obj1 ctr ctr1 rtrim ptrim tunegain bl2 br2 krp-s kirp-s
        kdrp-s kpy-s kiy-s kdy-s%recall updatates from global vars
        handles = guidata(hfigure);
        %-----%
        %Get current value of input plots
        %-----%
        throttle = get(handles.bar_throttle, 'YData');
        roll=get(handles.bar_roll, 'YData');
        pitch=-1*get(handles.bar_pitch, 'YData');
        yaw = get(handles.bar_yaw, 'YData');
        %-----%
151 %Get current value of buttons
        %-----%
        BA=button(joy,1);
        BB=button(joy,2);
        BX=button(joy,3);
        BY=button(joy,4);
        BL=button(joy,5);
        BR=button(joy,6);
        BK=button(joy,7);
        BS=button(joy,8);
161 BLS=button(joy,9);
        BRS=button(joy,10);
        %-----%
        %Trigger button callbacks if buttons are depressed
        %-----%
        ctr1=ctr1+1;
        if ctr1==15
            ctr1=0;
            ba=BA;
            bb=BB;
171 bx=BX;
            by=BY;
            bl=BL;

```

```

br=BR;
bs=BS;
bk=BK;
if ba==1
    ptrim=ptrim-1;
    pstring=strcat('Pitch Trim:', num2str(ptrim));
    set(handles.ptrimt, 'String', pstring);
181 else
    ptrim;
end
if bb==1
    rtrim=rtrim+1;
    rstring=strcat('Roll Trim:', ' ', num2str(rtrim));
    set(handles.rtrimt, 'String', rstring);
else
    rtrim;
end
191 if bx==1
    rtrim=rtrim-1;
    rstring=strcat('Roll Trim:', num2str(rtrim));
    set(handles.rtrimt, 'String', rstring);
else
    rtrim;
end
if by==1
    ptrim=ptrim+1;
    pstring=strcat('Pitch Trim:', num2str(ptrim));
201 set(handles.ptrimt, 'String', pstring);
else
    ptrim;
end
if bl==1
    bl2='D';
    fprintf(obj1, 'T50R50P75Y50')
else
    bl;
end
211 if bs==1
    fprintf(obj1, 'T00R50P50Y99')
    fprintf(obj1, 'T00R50P50Y50')
else
    bs;
end
if bk==1
    fprintf(obj1, 'T00R50P50Y00')
else
    bk;
221 end

if br==1
    br2='E';
    fprintf(obj1, 'T00R50P50Y50')

```



```

        else
            br;
        end

        else
231 end
    %-----%
    %%set new value of control input for sending to platform%%
    %% NEW Expo Setup for axes
    expo=2; %exponential variable for axis input. 2 is good
    %-----%
    %throttle axis - apply expo, scale, limit to 50<throttle<99 neutral=50
    %-----%
    throttle_raw=(-axis(joy,2));
    throttle_int=((throttle_raw^3)^(expo-1)+throttle_raw)/expo;
241 throttle_s=100*(throttle_raw);
    if throttle_s>99
        throttle_s=99;
    else
        throttle_s=throttle_s;
    end
    if throttle_s<0
        throttle_s=0;
    else
        throttle_s=throttle_s;
251 end
    %-----%
    %yaw axis - apply expo, scale, limit to 50<yaw<99, neutral=75
    %-----%
    yaw_raw=-axis(joy,3);
    yaw_int=((yaw_raw^3)^(expo-1)+yaw_raw)/expo;
    yaw_s=50*yaw_int+50;
    %yaw_s=yaw_int;
    if yaw_s >99
        yaw_s=99;
261 else
        yaw_s=yaw_s;
    end
    %-----%
    %roll axis - apply expo, scale, limit to 50<roll<99, neutral=75
    %-----%
    roll_raw=axis(joy,4);
    roll_int=((roll_raw^3)^(expo-1)+roll_raw)/expo;
    roll_s=50*(roll_int)+50+rtrim;
    %roll_s=roll_int;
271 if roll_s >99
        roll_s=99;
    else
        roll_s=roll_s;
    end
    %-----%
    %pitch axis - apply expo, scale, limit to 50<pitch<99, neutral=75

```

```

%-----%
pitch_raw=axis(joy,5);
pitch_int=((pitch_raw^3)^(expo-1)+pitch_raw)/expo;
281 pitch_s=50*(pitch_int)+50+ptrim;
%pitch_s=pitch_int;
if pitch_s >99
    pitch_s=99;
else
    pitch_s=pitch_s;
end
%-----%
%plot new value of input
%-----%
291 set(handles.bar_throttle,'YData',throttle_s);
set(handles.bar_roll,'YData',roll_s);
set(handles.bar_pitch,'YData',pitch_s);
set(handles.bar_yaw,'YData',yaw_s);
%-----%
%Format joystick commands for serial send
%-----%
%t_s=num2str(round(throttle_s));

t_s=sprintf('%02d',round(throttle_s));
301 set(handles.throttlestat,'String',t_s)
r_s=sprintf('%02.0f',roll_s);
set(handles.rollstat,'String',r_s)
p_s=sprintf('%02.0f',pitch_s);
set(handles.pitchstat,'String',p_s)
y_s=sprintf('%02.0f',yaw_s);
set(handles.yawstat,'String',y_s)
%-----%
%Tune gains, yo
%-----%
311 prp=get(handles.kprp,'String');
prp=str2num(prp)*1000;
irp=get(handles.kirp,'String');
irp=str2num(irp)*100000;
drp=get(handles.kdrp,'String');
drp=str2num(drp)*1000;
py=get(handles.kpy,'String');
py=str2num(py)*1000;
iy=get(handles.kiy,'String');
iy=str2num(iy)*1000;
321 dy=get(handles.kdy,'String');
dy=str2num(dy)*1000;
%-----%
%String and format gains, yo
%-----%
kprp_s=sprintf('%05d',prp);
kprp_s=strcat('A',kprp_s);
kirp_s=sprintf('%05d',irp);
kirp_s=strcat('B',kirp_s);

```

```

kdrp_s=sprintf( '%05d',drp);
331 kdrp_s=strcat( 'C',kdrp_s);
kpy_s=sprintf( '%05d',py);
kpy_s=strcat( 'D',kpy_s);
kiy_s=sprintf( '%05d',iy);
kiy_s=strcat( 'E',kiy_s);
kdy_s=sprintf( '%05d',dy);
kdy_s=strcat( 'F',kdy_s);
%gains=strcat( 'A',kprp_s, 'B',kirp_s, 'C',kdrp_s, 'D',kpy_s, 'E',kiy_s, 'F',
    kdy_s);

%-----%
341 %Send Serial command at 50Hz and append with "L" and packet indicator
% %-----%;
transmit=get(handles.transmitdata, 'Value');
if transmit ==1
    %if not tuning just send T,R,P,Y
    commout=strcat( 'T',t_s, 'R',r_s, 'P',p_s, 'Y',y_s)
    fprintf(obj1,commout)

else
    transmit=transmit;
351 end

% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% Before exiting, if the timer is running, stop it.
if strcmp(get(handles.timer, 'Running'), 'on')
    stop(handles.timer);
end
% Destroy timer
delete(handles.timer)
361 % END USER CODE

% Hint: delete(hObject) closes the figure
delete(hObject);
% --- Executes during object creation, after setting all properties.
function ax_throttle_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ax_throttle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called

371 % Hint: place code in OpeningFcn to populate ax_throttle
% --- Executes on selection change in comport.
function comport_Callback(hObject, eventdata, handles)
% hObject    handle to comport (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global obj1
id=get(handles.comport, 'value');
switch id

```

```

381     case 1
        fclose(obj1);
    case 2
        obj1 = instrfind('Type', 'serial', 'Port', 'COM1', 'Tag', '');
        P='COM1'
    case 3
        obj1 = instrfind('Type', 'serial', 'Port', 'COM2', 'Tag', '');
        P='COM2'
    case 4
        obj1 = instrfind('Type', 'serial', 'Port', 'COM3', 'Tag', '');
        P='COM3'
391    case 5
        obj1 = instrfind('Type', 'serial', 'Port', 'COM4', 'Tag', '');
        P='COM4'
    case 6
        obj1 = instrfind('Type', 'serial', 'Port', 'COM5', 'Tag', '');
        P='COM5'
end
    if isempty(obj1)
        obj1 = serial(P);
    else
401        fclose(obj1);
        obj1 = obj1(1)
    end
    set(obj1, 'Terminator', 'LF');
    set(obj1, 'BaudRate', 115200);
    set(obj1, 'InputBufferSize', 400000);
    set(obj1, 'Timeout', 3);
    fopen(obj1);
    set(handles.connect, 'Enable', 'on')

411 % --- Executes during object creation, after setting all properties.
function comport_CreateFcn(hObject, eventdata, handles)
% hObject    handle to comport (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
            called

% Hint: popupmenu controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'), get(0, '
    defaultUicontrolBackgroundColor'))
421     set(hObject, 'BackgroundColor', 'white');
end

% --- Executes on button press in sendki.
function sendki_Callback(hObject, eventdata, handles)
global obj1 kirp_s
    fprintf(obj1, kirp_s)
% hObject    handle to sendki (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

431 % --- Executes on button press in sendkd.
    function sendkd_Callback(hObject, eventdata, handles)
    % hObject    handle to sendkd (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    global obj1 kdrp-s
        fprintf(obj1,kdrp-s)

441 % --- Executes on button press in sendkpy.
    function sendkpy_Callback(hObject, eventdata, handles)
    % hObject    handle to sendkpy (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    global obj1 kpy-s
        fprintf(obj1,kpy-s)

    % --- Executes on button press in sendkiy.
451 function sendkiy_Callback(hObject, eventdata, handles)
    % hObject    handle to sendkiy (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    global obj1 kiy-s
        fprintf(obj1,kiy-s)

    % --- Executes on button press in sendkdy.
    function sendkdy_Callback(hObject, eventdata, handles)
461 % hObject    handle to sendkdy (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    global obj1 kdy-s
        fprintf(obj1,kdy-s)

    % --- Executes on button press in clearerror.
    function clearerror_Callback(hObject, eventdata, handles)
    % hObject    handle to clearerror (see GCBO)
471 % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    global obj1
        fprintf(obj1,'R')

```

B.2 Simulation Videos

B.2.1 Create Patches

```

% make the initial patches - vertices and faces.m
%quad
[ faces , vertices]=makequad( 'simple_sim3.stl' );
%ring
5 [ faces_r , vertices_r]=makequad( 'ring.stl' );
%Ball - make surface first , resize and convert to patches
[sphx , sphy , sphz]=sphere( );
[ faces_b , vertices_b , c]=surf2patch( .076.*sphx , .076.*sphy , .076.*sphz );
X_ground=[-.1 , -.1 , 8 , 8];
Y_ground=[3 , -3 , -3 , 3];
Z_ground=-.076.*[1 , 1 , 1 , 1];
%% Quad IC specification
r = [-0 , -0 , 0]; % Reference position
A = [0 , 0 , pi/2]; % Reference orientation (x-y-z Euler angle)
15 %% Euler angle -> Orientation matrix
R = Euler2R(A);
%% Vertices - Apply rotation and position for each
%-----Quad-----
vertices=[vertices(:,1)-.046 , vertices(:,2)-.076 , vertices(:,3)-.001];
VertexData = GeoVerMakeObj(r,R, vertices);
%-----Ring-----
vertices_r=[vertices_r(:,1)-.076 , vertices_r(:,2)-.076 , vertices_r(:,3)-0
.001];
VertexData_r = GeoVerMakeObj(r,R, vertices_r);
%-----Ball-----
25 vertices_b=[vertices_b(:,1) , vertices_b(:,2) , vertices_b(:,3)];
VertexData_b = GeoVerMakeObj(r,R, vertices_b);
%-----
% Make new patches
[PatchData_X , PatchData_Y , PatchData_Z] = GeoPatMake( VertexData , faces );
[PatchData_X_r , PatchData_Y_r , PatchData_Z_r] = GeoPatMake( VertexData_r ,
faces_r );
[PatchData_X_b , PatchData_Y_b , PatchData_Z_b] = GeoPatMake( VertexData_b ,
faces_b );
% Draw patches
figure
hold on
35 %-----Ball-----
ball=patch(PatchData_X_b , PatchData_Y_b , PatchData_Z_b , 'b');
set( ball , 'FaceLighting' , 'phong' , 'EdgeLighting' , 'phong' , 'EdgeColor' , 'none
');
set( ball , 'EraseMode' , 'normal' );
alpha(0.1)
%-----Quad-----
quad = patch(PatchData_X , PatchData_Y , PatchData_Z , 'r' );
set( quad , 'FaceLighting' , 'phong' , 'EdgeLighting' , 'phong' , 'EdgeColor' , 'none
');
set( quad , 'EraseMode' , 'normal' );
%-----Ring-----
45 ring = patch(PatchData_X_r , PatchData_Y_r , PatchData_Z_r , 'k' );

```

```

set (ring , 'FaceLighting' , 'phong' , 'EdgeLighting' , 'phong' , 'EdgeColor' , 'none
');
set (ring , 'EraseMode' , 'normal' );
%-----Ground-----
ground=patch(X_ground , Y_ground , Z_ground , 'g' );
set (ground , 'FaceLighting' , 'phong' , 'EdgeLighting' , 'phong' , 'EdgeColor' , '
none' , 'FaceColor' , [0.7 0.7 0.7] );
set (ground , 'EraseMode' , 'normal' );

% Axes settings
xlabel ( 'x' , 'FontSize' , 14 );
55 ylabel ( 'y' , 'FontSize' , 14 );
zlabel ( 'z' , 'FontSize' , 14 );
set (gca , 'FontSize' , 14 );
axis vis3d equal;
view ( [-37.5 , 30] );
camlight;
grid on;
xlim ( [-10 , 10] );
ylim ( [-10 , 10] );
zlim ( [-1 , 1] );

```

B.2.2 Animation

```

clear M
% Motion data
minx=-.1;
maxx=8;
miny=-3;
6 maxy=3;
minz=-.076;
maxz=4;
t = tsim'; % Time data
r = [xsim , ysim , zsim]; % Position data
Q_A = [-phisim , -thetasim , psisim]; % Quad orientation Data
B_A = [-phisim , alphasim , psisim] %ball/ring orientation Data
n_time = length(tsim); % length of video
% Compute propagation of vertices and patches
16 for i_time=1:n_time
    R_Q = Euler2R(Q_A(i_time , :));
    R_B=Euler2R(B_A(i_time , :));
    %-----Quad-----
    VertexData(:, :, i_time) = GeoVerMakeObj(r(i_time , :), R_Q, vertices);
    [X,Y,Z] = GeoPatMake(VertexData(:, :, i_time), faces);
    PatchData_X(:, :, i_time) = X;
    PatchData_Y(:, :, i_time) = Y;
    PatchData_Z(:, :, i_time) = Z;
    %-----Ball-----

```

```

VertexData_b(:, :, i_time) = GeoVerMakeObj(r(i_time, :), R_B, vertices_b)
;
26 [X_b, Y_b, Z_b] = GeoPatMake(VertexData_b(:, :, i_time), faces_b);
PatchData_X_b(:, :, i_time) = X_b;
PatchData_Y_b(:, :, i_time) = Y_b;
PatchData_Z_b(:, :, i_time) = Z_b;
%-----Ring-----
VertexData_r(:, :, i_time) = GeoVerMakeObj(r(i_time, :), R_B, vertices_r)
;
[X_r, Y_r, Z_r] = GeoPatMake(VertexData_r(:, :, i_time), faces_r);
PatchData_X_r(:, :, i_time) = X_r;
PatchData_Y_r(:, :, i_time) = Y_r;
PatchData_Z_r(:, :, i_time) = Z_r;
36 end
% Draw initial figure 4 total axes
%-----
Scene=figure('units','normalized','outerposition',[0.0 0.05 1 .95])
% subplot('Position',[.05 .1 .6 .85]);
xlabel('x [m]','FontSize',22,'FontName','Times New Roman');
ylabel('y [m]','FontSize',22,'FontName','Times New Roman');
zlabel('z [m]','FontSize',22,'FontName','Times New Roman');
set(gca,'FontSize',22,'FontName','Times New Roman');
%axis vis3d equal;
46 view([40,30]);
camlight;
grid on;
xlim([minx,maxx]);
ylim([miny,maxy]);
zlim([minz,maxz]);
%patches for each solid%
p1 = patch(PatchData_X(:, :, 1), PatchData_Y(:, :, 1), PatchData_Z(:, :, 1), 'r')
;
p1_b = patch(PatchData_X_b(:, :, 1), PatchData_Y_b(:, :, 1), PatchData_Z_b
(:, :, 1), 'b');
p1_r = patch(PatchData_X_r(:, :, 1), PatchData_Y_r(:, :, 1), PatchData_Z_r
(:, :, 1), 'k');
56 alpha(p1_b, 0.25);
%-----APPEARANCE-----
set(p1_b, 'FaceLighting', 'phong', 'EdgeLighting', 'phong', 'EdgeColor', 'none
');
set(p1_b, 'EraseMode', 'normal');
set(p1, 'FaceLighting', 'phong', 'EdgeLighting', 'phong', 'EdgeColor', 'Red')
set(p1_r, 'FaceLighting', 'phong', 'EdgeLighting', 'phong', 'EdgeColor', '
Black')
ground=patch(X_ground, Y_ground, Z_ground, 'g');
set(ground, 'FaceLighting', 'phong', 'EdgeLighting', 'phong', 'EdgeColor', '
none', 'FaceColor', [0.7 0.7 0.7]);
set(ground, 'EraseMode', 'normal');
66 % %-----
% subplot('Position',[.76 .718 .2 .245]);
% title('Top View','FontSize',12)

```



```

% xlabel('x [m]', 'FontSize', 12);
% ylabel('y [m]', 'FontSize', 12);
% zlabel('z [m]', 'FontSize', 12);
% set(gca, 'FontSize', 10);
% %axis vis3d equal;
% view([0,90]);
% camlight;
76 % grid on;
% xlim([minx,maxx]);
% ylim([miny,maxy]);
% zlim([minz,maxz]);
% %patches for each solid%
% p2 = patch(PatchData_X(:,:,1), PatchData_Y(:,:,1), PatchData_Z(:,:,1), 'r'
    ');
% p2_b = patch(PatchData_X_b(:,:,1), PatchData_Y_b(:,:,1), PatchData_Z_b
    (:,:,1), 'b');
% p2_r = patch(PatchData_X_r(:,:,1), PatchData_Y_r(:,:,1), PatchData_Z_r
    (:,:,1), 'k');
% alpha(p2_b, 0.25);
% %-----APPEARANCE-----
86 % set(p2_b, 'FaceLighting', 'phong', 'EdgeLighting', 'phong', 'EdgeColor', '
    none');
% set(p2_b, 'EraseMode', 'normal');
% set(p2, 'FaceLighting', 'phong', 'EdgeLighting', 'phong', 'EdgeColor', 'Red'
    ');
% set(p2_r, 'FaceLighting', 'phong', 'EdgeLighting', 'phong', 'EdgeColor', '
    Black');
% %-----
% subplot('Position', [.76 .384 .2 .245]);
% title('Front View', 'FontSize', 12);
% xlabel('x [m]', 'FontSize', 12);
% ylabel('y [m]', 'FontSize', 12);
% zlabel('z [m]', 'FontSize', 12);
96 % set(gca, 'FontSize', 10);
% %axis vis3d equal;
% view([90,0]);
% camlight;
% grid on;
% xlim([minx,maxx]);
% ylim([miny,maxy]);
% zlim([minz,maxz]);
% %-----
% p3 = patch(PatchData_X(:,:,1), PatchData_Y(:,:,1), PatchData_Z(:,:,1), 'r'
    ');
106 % p3_b = patch(PatchData_X_b(:,:,1), PatchData_Y_b(:,:,1), PatchData_Z_b
    (:,:,1), 'b');
% p3_r = patch(PatchData_X_r(:,:,1), PatchData_Y_r(:,:,1), PatchData_Z_r
    (:,:,1), 'k');
% alpha(p3_b, 0.25);
% %-----APPEARANCE-----
% set(p3_b, 'FaceLighting', 'phong', 'EdgeLighting', 'phong', 'EdgeColor', '
    none');

```

```

% set(p3_b,'EraseMode','normal');
% set(p3,'FaceLighting','phong','EdgeLighting','phong','EdgeColor','Red
    ')
% set(p3_r,'FaceLighting','phong','EdgeLighting','phong','EdgeColor','
    Black')
% %-----
% subplot('Position',[.76 .053 .2 .245]);
116 % title('Side View','FontSize',12)
% xlabel('x [m]','FontSize',12);
% ylabel('y [m]','FontSize',12);
% zlabel('z [m]','FontSize',12);
% set(gca,'FontSize',10);
% %axis vis3d equal;
% view([0,0]);
% camlight;
% grid on;
% xlim([minx,maxx]);
126 % ylim([miny,maxy]);
% zlim([minz,maxz]);
% p4 = patch(PatchData_X(:,:,1),PatchData_Y(:,:,1),PatchData_Z(:,:,1),'r
    ');
% p4_b = patch(PatchData_X_b(:,:,1),PatchData_Y_b(:,:,1),PatchData_Z_b
    (:,:,1),'b');
% p4_r = patch(PatchData_X_r(:,:,1),PatchData_Y_r(:,:,1),PatchData_Z_r
    (:,:,1),'k');
% alpha(p4_b,0.25);
% %-----
% %-----APPEARANCE-----
% set(p4_b,'FaceLighting','phong','EdgeLighting','phong','EdgeColor','
    none');
% set(p4_b,'EraseMode','normal');
136 % set(p4,'FaceLighting','phong','EdgeLighting','phong','EdgeColor','Red
    ')
% set(p4_r,'FaceLighting','phong','EdgeLighting','phong','EdgeColor','
    Black')

% Animation Loop
for i_time=1:n_time

    set(p1,'XData',PatchData_X(:,:,i_time));
    set(p1_b,'XData',PatchData_X_b(:,:,i_time));
    set(p1_r,'XData',PatchData_X_r(:,:,i_time));
    set(p1,'YData',PatchData_Y(:,:,i_time));
146 set(p1_b,'YData',PatchData_Y_b(:,:,i_time));
    set(p1_r,'YData',PatchData_Y_r(:,:,i_time));
    set(p1,'ZData',PatchData_Z(:,:,i_time));
    set(p1_b,'ZData',PatchData_Z_b(:,:,i_time));
    set(p1_r,'ZData',PatchData_Z_r(:,:,i_time));
    %-----
%     set(p2,'XData',PatchData_X(:,:,i_time));
%     set(p2_b,'XData',PatchData_X_b(:,:,i_time));
%     set(p2_r,'XData',PatchData_X_r(:,:,i_time));

```

```

156 %      set(p2,'YData',PatchData_Y(:,:,i_time));
%      set(p2_b,'YData',PatchData_Y_b(:,:,i_time));
%      set(p2_r,'YData',PatchData_Y_r(:,:,i_time));
%      set(p2,'ZData',PatchData_Z(:,:,i_time));
%      set(p2_b,'ZData',PatchData_Z_b(:,:,i_time));
%      set(p2_r,'ZData',PatchData_Z_r(:,:,i_time));
%      %-----
%      set(p3,'XData',PatchData_X(:,:,i_time));
%      set(p3_b,'XData',PatchData_X_b(:,:,i_time));
%      set(p3_r,'XData',PatchData_X_r(:,:,i_time));
%      set(p3,'YData',PatchData_Y(:,:,i_time));
166 %      set(p3_b,'YData',PatchData_Y_b(:,:,i_time));
%      set(p3_r,'YData',PatchData_Y_r(:,:,i_time));
%      set(p3,'ZData',PatchData_Z(:,:,i_time));
%      set(p3_b,'ZData',PatchData_Z_b(:,:,i_time));
%      set(p3_r,'ZData',PatchData_Z_r(:,:,i_time));
%      %-----
%      set(p4,'XData',PatchData_X(:,:,i_time));
%      set(p4_b,'XData',PatchData_X_b(:,:,i_time));
%      set(p4_r,'XData',PatchData_X_r(:,:,i_time));
%      set(p4,'YData',PatchData_Y(:,:,i_time));
176 %      set(p4_b,'YData',PatchData_Y_b(:,:,i_time));
%      set(p4_r,'YData',PatchData_Y_r(:,:,i_time));
%      set(p4,'ZData',PatchData_Z(:,:,i_time));
%      set(p4_b,'ZData',PatchData_Z_b(:,:,i_time));
%      set(p4_r,'ZData',PatchData_Z_r(:,:,i_time));
%      %-----

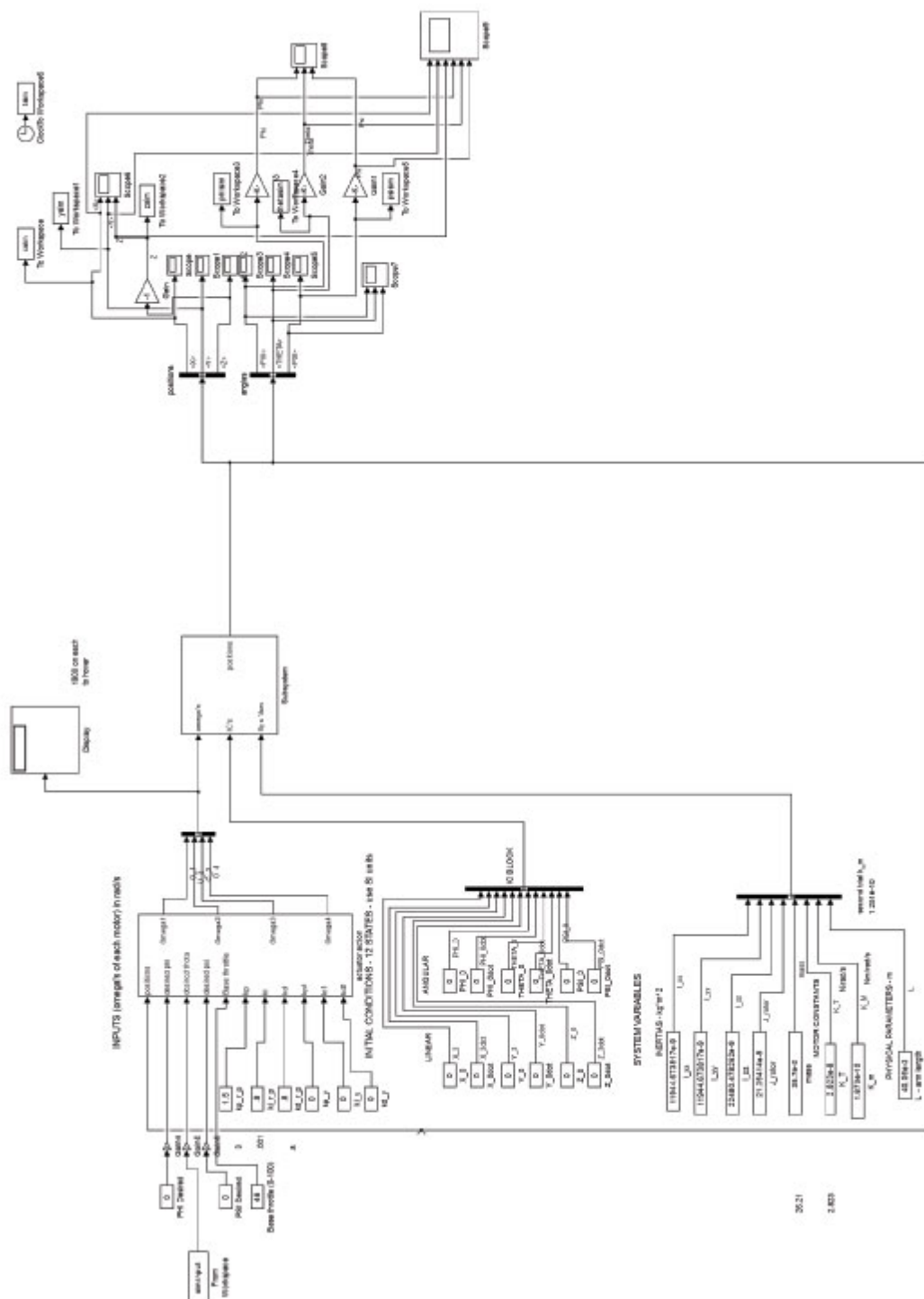
      drawnow;
      M(i_time) = getframe(Scene);
end
186 movie2avi(M,'flyingtoroll_good.avi','FPS',15,'compression','none','
      QUALITY',100,'KEYFRAME',2);

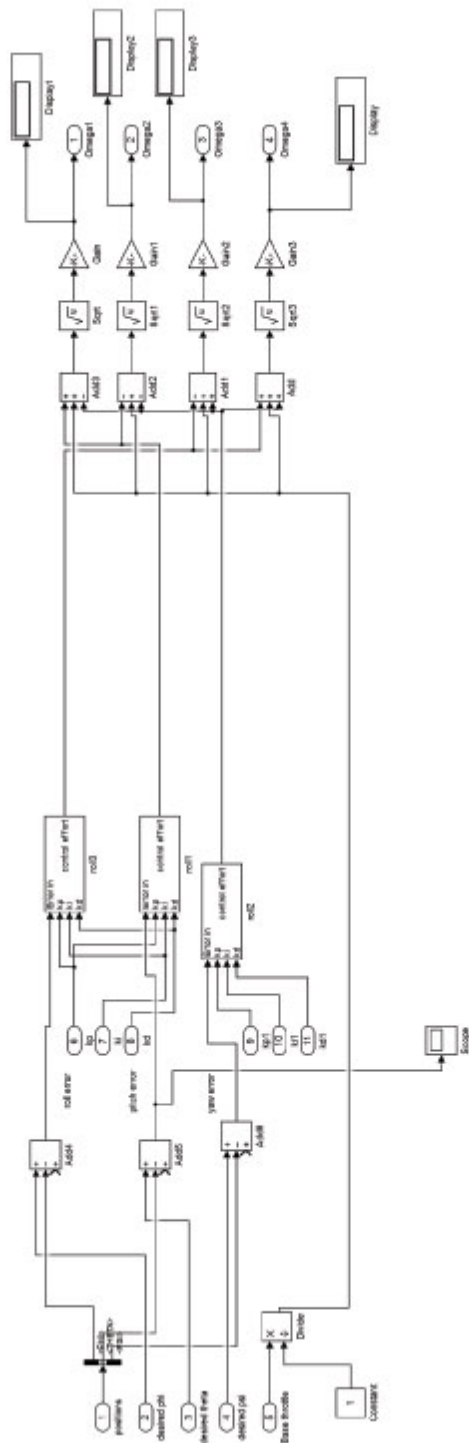
```

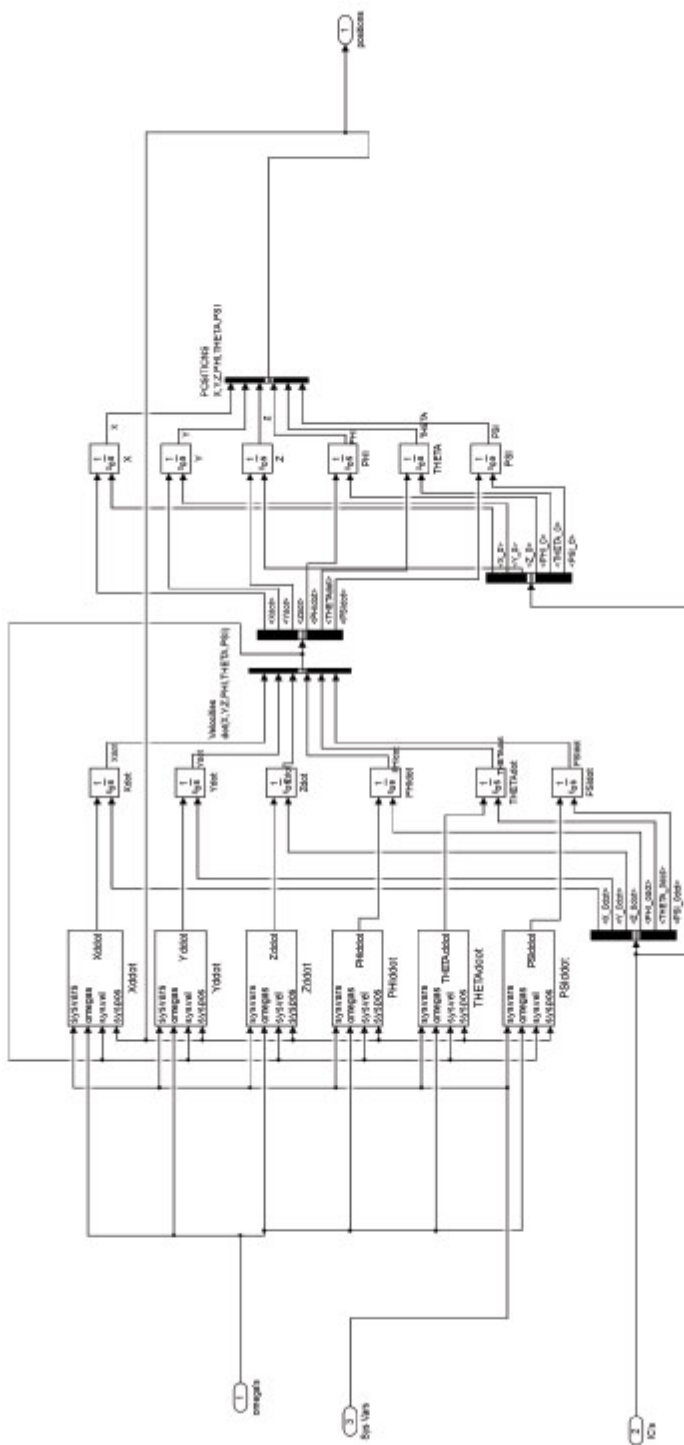
Appendix C

Simulink Block Diagram

C.1 System Block Diagram







Appendix D

ATR Control Code

breaklines

```

// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050
//   class using DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/
// jrowberg/i2cdevlib
4 //
// Changelog:
//   2013-05-08 - added seamless Fastwire support
//               - added note about gyro calibration
//   2012-06-21 - added note about Arduino 1.0.1 + Leonardo
//               compatibility error
//   2012-06-20 - improved FIFO overflow handling and simplified read
//               process
//   2012-06-19 - completely rearranged DMP initialization code and
//               simplification
//   2012-06-13 - pull gyro and accel data from FIFO packet instead
//               of reading directly
//   2012-06-09 - fix broken FIFO read sequence and change interrupt
//               detection to RISING
//   2012-06-05 - add gravity-compensated initial reference frame
//               acceleration output
14 //               - add 3D math helper file to DMP6 example sketch
//               - add Euler output and Yaw/Pitch/Roll output formats
//   2012-06-04 - remove accel offset clearing for better results (
//               thanks Sungon Lee)
//   2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead
//               of 250
//   2012-05-30 - basic DMP initialization working

/* =====
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

24 Permission is hereby granted, free of charge, to any person obtaining a

```



```

        copy
of this software and associated documentation files (the "Software"),
to deal
in the Software without restriction, including without limitation the
rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or
sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included
in
all copies or substantial portions of the Software.

34 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY
,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN
THE SOFTWARE.

```

```

*/

44 // I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.
h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
// #include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE
implementation
// is used in I2Cdev.h
54 #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense
evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high

```

```

64  /* =====
    NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
    depends on the MPU-6050's INT pin being connected to the Arduino's
    external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
    digital I/O pin 2.
    * ===== */

    /* =====
74  NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
    when using Serial.write(buf, len). The Teapot output uses this method.
    The solution requires a modification to the Arduino USBAPI.h file ,
        which
    is fortunately simple, but annoying. This will be fixed in the next IDE
    release. For more info, see these links:

    http://arduino.cc/forum/index.php/topic,109987.0.html
    http://code.google.com/p/arduino/issues/detail?id=958
    * ===== */
84  // uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
    // quaternion components in a [w, x, y, z] format (not best for parsing
    // on a remote host such as Processing or something though)
    // #define OUTPUT_READABLE_QUATERNION

    // uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
    // (in degrees) calculated from the quaternions coming from the FIFO.
    // Note that Euler angles suffer from gimbal lock (for more info, see
    // http://en.wikipedia.org/wiki/Gimbal_lock)
    // #define OUTPUT_READABLE_EULER

    // uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
94  // pitch/roll angles (in degrees) calculated from the quaternions coming
    // from the FIFO. Note this also requires gravity vector calculations.
    // Also note that yaw/pitch/roll angles suffer from gimbal lock (for
    // more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
    #define OUTPUT_READABLE_YAWPITCHROLL

    // uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
    // components with gravity removed. This acceleration reference frame is
    // not compensated for orientation, so +X is always +X according to the
    // sensor, just without the effects of gravity. If you want acceleration
104  // compensated for orientation, use OUTPUT_READABLE_WORLDACCEL instead.
    // #define OUTPUT_READABLE_REALACCEL

    // uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see
        acceleration
    // components with gravity removed and adjusted for the world frame of
    // reference (yaw is relative to initial orientation, since no
        magnetometer
    // is present in this case). Could be quite handy in some cases.
    // #define OUTPUT_READABLE_WORLDACCEL

```

```

114 // uncomment "OUTPUT_TEAPOT" if you want output that matches the
    // format used for the InvenSense teapot demo
    // #define OUTPUT_TEAPOT

    #define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
    bool blinkState = false;

124 // MPU control/status vars
    bool dmpReady = false; // set true if DMP init was successful
    uint8_t mpuIntStatus; // holds actual interrupt status byte from MPU
    uint8_t devStatus; // return status after each device operation (0
        = success, !0 = error)
    uint16_t packetSize; // expected DMP packet size (default is 42 bytes
        )
    uint16_t fifoCount; // count of all bytes currently in FIFO
    uint8_t fifoBuffer[64]; // FIFO storage buffer

    // orientation/motion vars
    Quaternion q; // [w, x, y, z] quaternion container
134 VectorInt16 aa; // [x, y, z] accel sensor
        measurements
    VectorInt16 aaReal; // [x, y, z] gravity-free accel
        sensor measurements
    VectorInt16 aaWorld; // [x, y, z] world-frame accel sensor
        measurements
    VectorFloat gravity; // [x, y, z] gravity vector
    float euler[3]; // [psi, theta, phi] Euler angle container
    float ypr[3]; // [yaw, pitch, roll] yaw/pitch/roll container
        and gravity vector

    // packet structure for InvenSense teapot demo
    uint8_t teapotPacket[14] = {
144 '$', 0x02, 0, 0, 0, 0, 0, 0, 0, 0, 0x00, 0x00, '\r', '\n' };

    // =====
    // == INTERRUPT DETECTION ROUTINE ==
    // =====

    volatile bool mpuInterrupt = false; // indicates whether MPU
        interrupt pin has gone high
    void dmpDataReady() {
154 mpuInterrupt = true;
    }

    // =====

```

```

// == VARIABLE DEFINITIONS ==
// =====

float kp_roll, ki_roll, kd_roll;

    // Controller gains
float kp_pitch, ki_pitch, kd_pitch;
float kp_yaw, ki_yaw, kd_yaw;
164 float *kp_adjust, *ki_adjust, *kd_adjust;

float roll_error_current, pitch_error_current, yaw_error_current;
                                // Current errors
float pitch_error_filt_current;
float roll_error_old, pitch_error_old, yaw_error_old;
                                // Previous errors
float pitch_error_filt_old;
float roll_error_accum, pitch_error_accum, yaw_error_accum;
                                // Accumulated errors

float roll_control, pitch_control, yaw_control;
                                // Controller
    effort
float P_term_roll, I_term_roll, D_term_roll;
174 float P_term_pitch, I_term_pitch, D_term_pitch;
float P_term_yaw, I_term_yaw, D_term_yaw;

float roll_des, pitch_des, yaw_des, yaw_rate_des;
                                // Desired
    attitudes
float roll_act, pitch_act, yaw_act, yaw_rate_act;
                                // Actual
    attitudes
float yaw_act_old;

    // Old attitudes
float roll_act_old, roll_act_filt, roll_filt_old;
                                // Filtered attitudes
float pitch_act_old, pitch_act_filt, pitch_filt_old;

int throttle, throttle_old, throttle_older, roll, pitch, yaw;

    // Commands from GCS
184 char command, buffer1[3], buffer2[6];
                                //
    Serial commands

int motor_speeds[4];

int t0, t1;
                                //
    Times in milliseconds used to calculate dt
int dt;
                                //

```

```

    Time interval since last loop in milliseconds

int i, j, enabled, disp;

float alpha_low, RC_low, freq_low;

    Low pass filter variables
194
// =====
// ==                      FUNCTION DEFINITIONS                      ==
// =====

void initialize_variables(void);
void update_motors(void);
void motors_init(void);
void DMP_init(void);
204 void DMP_retreival(void);
void calculate_attitudes(void);
void PID(void);
void process_commands(void);

// =====
// ==                      INITIAL SETUP                      ==
// =====

void setup() {
214   // Initialize Motors
   motors_init();

   // configure LED for output
   pinMode(LED_PIN, OUTPUT);

   // Initialize variables
   initialize_variables();

   DMP_init();
224 }

// =====
// ==                      MAIN PROGRAM LOOP                      ==
// =====

void loop() {
   DMP_retreival();

234   // Get time interval since last loop
   t1 = millis();
   dt = (t1 - t0);
   t0 = t1;

```

```

    calculate_attitudes();

    PID();

    // Add control effort and throttle command and send to motors
244  if(throttle>=5&&throttle<100&&enabled==1){
        motor_speeds[0] = throttle+pitch_control-yaw_control;
        motor_speeds[1] = throttle-pitch_control-yaw_control;
        motor_speeds[2] = throttle+roll_control+yaw_control;
        motor_speeds[3] = throttle-roll_control+yaw_control;
    }
    else{
        motor_speeds[0] = 0;
        motor_speeds[1] = 0;
        motor_speeds[2] = 0;
254    motor_speeds[3] = 0;
    }

    update_motors();

    process_commands();

    disp++;

    //Send serial data packet
264  if(disp == 2){
        Serial.print(throttle);
        Serial.print(",");
        Serial.print(pitch_des,2);
        Serial.print(",");
        Serial.print(pitch_act,2);
        Serial.print(",");
        Serial.print(P_term_pitch);
        Serial.print(",");
        Serial.print(I_term_pitch);
274    Serial.print(",");
        Serial.print(D_term_pitch);
        Serial.print(",");
        Serial.println(pitch_control);
        disp = 0;
    }

} // END LOOP

284
// =====
// ===== FUNCTIONS =====
// =====

void motors_init(void){
    pinMode(3,OUTPUT); // Motor 2 BL

```

```

pinMode(9,OUTPUT); // Motor 3 FL
pinMode(10,OUTPUT); // Motor 4 BR
pinMode(11,OUTPUT); // Motor 1 FR
294
TCCR1A = (1<<COM1A1) | (1<<COM1B1) | (1<<WGM10); //Enable fast 8-bit
        PWM mode, non inverting on OC1A and OC1B
TCCR1B = (1<<WGM12) | (1<<CS11); //Finish setting up 8-bit PWM and set
        clock prescaler for ~8 kHz freq
TCCR2A = (1<<COM2A1) | (1<<COM2B1) | (1<<WGM21) | (1<<WGM20); //Enable
        fast PWM mode, non inverting on OC2A and OC2B
TCCR2B = (1<<CS21); // Set prescaler for ~8 kHz freq
OCR2A = 0; //Initialize motors to zero duty cycle
OCR2B = 0;
OCR1A = 0;
OCR1B = 0;
}
304
void update_motors(void){
    for(int i=0;i<4;i++){
        if(motor_speeds[i]<=0) motor_speeds[i] = 0;
        if(motor_speeds[i]>=100) motor_speeds[i] = 100;
    }

    OCR1B = (motor_speeds[0]*255)/100; // FR motor
    OCR2A = (motor_speeds[1]*255)/100; // BL Motor
    OCR2B = (motor_speeds[2]*255)/100; // FL Motor
314 OCR1A = (motor_speeds[3]*255)/100; // BR Motor
    }

void initialize_variables(void){
    throttle = 0; // Initialize inputs to default "resting"
    state
    roll = 50;
    pitch = 50;
    yaw = 50;

324 roll_des = 0; // Desired attitude to zero
    pitch_des = 0;
    yaw_des = 0;

    roll_error_accum = 0; // Accumulated errors to zero
    pitch_error_accum = 0;
    yaw_error_accum = 0;

    roll_error_old = 0; // Previous errors to zero
    pitch_error_old = 0;
334 yaw_error_old = 0;

    kp_roll = .55*4.5; // Set controller gains
    ki_roll = .25*4.5*.001;
    kd_roll = 0.12*4*1000;

```

```

kp_pitch = 0.160;
ki_pitch = 0.200*.001;
kd_pitch = 0.055*1000;
kp_yaw = .625;
ki_yaw = 0.375*.001;
344 kd_yaw = 0.25*1000;

t0 = 0; // Previous time to zero
enabled = 1; // Disable motors by default
digitalWrite(LED_PIN, 1);
disp = 0;

// Set up low pass filter
freq_low = 1;
RC_low= 1/(2*3.14159*freq_low);
354 alpha_low = 0.01/(RC_low+0.01);
}

void DMP_init(void){

// join I2C bus (I2Cdev library doesn't do this automatically)
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
Wire.begin();
TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
364 Fastwire::setup(400, true);
#endif

// initialize serial communication
Serial.begin(57600);
while (!Serial); // wait for Leonardo enumeration, others continue
immediately

// NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or
Arduinio
// Pro Mini running at 3.3v, cannot handle this baud rate reliably due
to
374 // the baud timing being too misaligned with processor ticks. You must
use
// 38400 or slower in these cases, or use some kind of external
separate
// crystal solution for the UART timer.

// initialize device
Serial.println(F("Initializing I2C devices..."));
mpu.initialize();

// verify connection
Serial.println(F("Testing device connections..."));
384 Serial.println(mpu.testConnection() ? F("MPU6050 connection successful
") : F("MPU6050 connection failed"));

```



```

// wait for ready
Serial.println(F("Send any character to begin: "));
while (Serial.available() && Serial.read()); // empty buffer
while (!Serial.available()); // wait for data
while (Serial.available() && Serial.read()); // empty buffer again

// load and configure the DMP
Serial.println(F("Initializing DMP..."));
394 devStatus = mpu.dmpInitialize();

mpu.setXGyroOffset(39);
mpu.setYGyroOffset(68);
mpu.setZGyroOffset(230);
mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

// make sure it worked (returns 0 if so)
if (devStatus == 0) {
404 // turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP..."));
    mpu.setDMPEnabled(true);

    // enable Arduino interrupt detection
    Serial.println(F("Enabling interrupt detection (Arduino external
        interrupt 0)..."));
    attachInterrupt(0, dmpDataReady, RISING);
    mpuIntStatus = mpu.getIntStatus();

    // set our DMP Ready flag so the main loop() function knows it's
    // okay to use it
    Serial.println(F("DMP ready! Waiting for first interrupt..."));
414 dmpReady = true;

    // get expected DMP packet size for later comparison
    packetSize = mpu.dmpGetFIFOPacketSize();
}
else {
    // ERROR!
    // 1 = initial memory load failed
    // 2 = DMP configuration updates failed
    // (if it's going to break, usually the code will be 1)
424 Serial.print(F("DMP Initialization failed (code "));
    Serial.print(devStatus);
    Serial.println(F(")"));
}
}

void DMP_retreival(void){
    // if programming failed, don't try to do anything
    if (!dmpReady) return;
434

```

```

// wait for MPU interrupt or extra packet(s) available
while (!mpuInterrupt && fifoCount < packetSize) {
}

// reset interrupt flag and get INT_STATUS byte
mpuInterrupt = false;
mpuIntStatus = mpu.getIntStatus();

// get current FIFO count
444 fifoCount = mpu.getFIFOCount();

// check for overflow (this should never happen unless our code is too
// inefficient)
if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
    // reset so we can continue cleanly
    mpu.resetFIFO();
    Serial.println(F("FIFO overflow!"));

    // otherwise, check for DMP data ready interrupt (this should happen
    // frequently)
}
454 else if (mpuIntStatus & 0x02) {
    // wait for correct available data length, should be a VERY short
    // wait
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    // read a packet from FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);

    // track FIFO count here in case there is > 1 packet available
    // (this lets us immediately read more without waiting for an
    // interrupt)
    fifoCount -= packetSize;
464

    // Get Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
}

}

474 void calculate_attitudes(void){
    // Get current attitude
    roll_act = (ypr[2]*.7071-ypr[1]*.7071)*180/M_PI*1.47;    // Actual
    roll
    pitch_act = (ypr[1]*.7071+ypr[2]*.7071)*180/M_PI*1.47;    // Actual
    pitch
    yaw_act = ypr[0]*180/M_PI;    // Actual yaw
    yaw_rate_act = (yaw_act-yaw_act_old)*1000/dt;    // Yaw
    rate in [ /s] (dt is in ms, so multiply by 1000 to get in seconds

```

```

    )

    roll_act_filt = roll_filt_old+alpha_low*(roll_act - roll_filt_old);
    roll_filt_old = roll_act_filt;

484    pitch_act_filt = pitch_filt_old+alpha_low*(pitch_act - pitch_filt_old);

    // Get desired attitude
    roll_des = (roll-50)*1;           // Max angle when sticks are
        mashed is 50*1= 50 [  ]
    pitch_des = (pitch-50)*1;
    // If I'm receiving an active yaw command, update desired heading
494    if(yaw > 51 || yaw < 49){
        yaw_des += (yaw-50)*.04;
    }
    // Else, just maintain old heading
}

void PID(void){

    // Calculate current errors
    roll_error_current = roll_des - roll_act;           // Current
        Roll error
504    roll_error_accum += roll_error_current*dt;         //
        Accumulated Roll error

    pitch_error_current = pitch_des - pitch_act;           //
        Current Pitch error
    pitch_error_filt_current = pitch_error_filt_old+alpha_low*(
        pitch_error_current - pitch_error_filt_old);    // Current Pitch
        error filtered
    pitch_error_accum += pitch_error_current*dt;         //
        Accumulated Pitch error

    yaw_error_current = yaw_des - yaw_act;           // Current
        Yaw error
    yaw_error_accum += yaw_error_current*dt;         //
        Accumulated Yaw error

514    // Calculate controller efforts
    P_term_roll = kp_roll*roll_error_current;
    I_term_roll = ki_roll*roll_error_accum;
    D_term_roll = kd_roll*(roll_error_current - roll_error_old)/dt;

    P_term_pitch = kp_pitch*pitch_error_current;

```

```

I_term_pitch = ki_pitch*pitch_error_accum;
D_term_pitch = kd_pitch*(pitch_error_current - pitch_error_old)/dt;

524 roll_control = P_term_roll + I_term_roll + D_term_roll;
pitch_control = P_term_pitch + I_term_pitch + D_term_pitch;
yaw_control = kp_yaw*yaw_error_current + ki_yaw*yaw_error_accum +
    kd_yaw*(yaw_error_current - yaw_error_old)/dt;

// Current errors become previous errors
roll_error_old = roll_error_current;
pitch_error_old = pitch_error_current;
pitch_error_filt_old = pitch_error_filt_current;
yaw_error_old = yaw_error_current;
534 roll_act_old = roll_act;
pitch_act_old = pitch_act;
yaw_act_old = yaw_act;

}

void process_commands(void){
// If 1 character available, deal with it
544 if (Serial.available() == 2) {
    command = Serial.read();
    Serial.read();
    /*
    if(command=='R'){
        record = 1;
    }

    else if(command=='T'){
        record = 0;
554 for (i=0;i++;i<j+1){
    Serial.print(pitch_des_recorded[i],2);
    Serial.print(",");
    Serial.print(pitch_act_recorded[i],2);
    Serial.print(",");
    Serial.println(dt_recorded[i],2);
    }
    j = 0;
    }
    */
564 }

// If 12 characters are available, assign TRPY values
else if (Serial.available() == 13) {
    Serial.read(); // read the T
    for (i=0;i<2;i++){
        buffer1[i] = Serial.read();

```

```

    }
    buffer1[i] = ' ';
    throttle = atoi(buffer1);
574  throttle_older = throttle_old;
    throttle_old = throttle;

    Serial.read(); // read the R
    for(i=0;i<2;i++){
        buffer1[i] = Serial.read();
    }
    buffer1[i] = ' ';
    roll = atoi(buffer1);

584  Serial.read(); // read the P
    for(i=0;i<2;i++){
        buffer1[i] = Serial.read();
    }
    buffer1[i] = ' ';
    pitch = atoi(buffer1);

    Serial.read(); // read the Y
    for(i=0;i<2;i++){
594  buffer1[i] = Serial.read();
    }
    buffer1[i] = ' ';
    yaw = atoi(buffer1);

    if(throttle_older < 5 && throttle >= 5){
        enabled = 1;
        yaw_des = yaw_act;
        roll_error_accum = 0;
        pitch_error_accum = 0;
        yaw_error_accum = 0;
604  digitalWrite(LED_PIN, 1);
    }
    /*
    else if(throttle == 0 && yaw == 0){
        enabled = 0;
        digitalWrite(LED_PIN, 0);
    }*/
}

// If 7 characters available, assign gain values
614 /*else if (Serial.available() == 7 && enabled == 0) {
    command = Serial.read();

    if(command == 'A'){
        for(i=0;i<5;i++){
            buffer2[i] = Serial.read();
        }
        buffer2[i] = ' ';
        kp = (float)atol(buffer2)/1000;

```

```

624     digitalWrite(LED_PIN, 0);
        delay(100);
        digitalWrite(LED_PIN, 1);
    }

    else if(command == 'B'){
        for(i=0;i<5;i++){
            buffer2[i] = Serial.read();
        }
        buffer2[i] = ' ';
        ki = (float)atol(buffer2)/100000;
634     digitalWrite(LED_PIN, 0);
        delay(100);
        digitalWrite(LED_PIN, 1);
    }

    else if(command == 'C'){
        for(i=0;i<5;i++){
            buffer2[i] = Serial.read();
        }
        buffer2[i] = ' ';
644     kd = (float)atol(buffer2)/1000;
        digitalWrite(LED_PIN, 0);
        delay(100);
        digitalWrite(LED_PIN, 1);
    }

    else if(command == 'D'){
        for(i=0;i<5;i++){
            buffer2[i] = Serial.read();
        }
654     buffer2[i] = ' ';
        kp_yaw = (float)atol(buffer2)/1000;
        digitalWrite(LED_PIN, 0);
        delay(100);
        digitalWrite(LED_PIN, 1);
    }

    else if(command == 'E'){
        for(i=0;i<5;i++){
            buffer2[i] = Serial.read();
664     }
        buffer2[i] = ' ';
        ki_yaw = (float)atol(buffer2)/1000;
        digitalWrite(LED_PIN, 0);
        delay(100);
        digitalWrite(LED_PIN, 1);
    }

    else if(command == 'F'){
        for(i=0;i<5;i++){
674     buffer2[i] = Serial.read();

```

```
        }  
        buffer2[i] = ' '  
        kd_yaw = (float)atol(buffer2)/1000;  
        digitalWrite(LED_PIN, 1);  
        delay(100);  
        digitalWrite(LED_PIN, 0);  
    }  
}*/  
684 // If none of the options above, something got lost in transmission so  
    clear buffer  
    else {  
        while(Serial.available() && Serial.read()); // empty buffer  
    }  
}
```