UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

PRELIMINARY DESIGN OF ROBUST ADAPTIVE CONTROL LAWS FOR AN AERIAL MANIPULATOR

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

John-Paul Burke

Norman, Oklahoma

2019

PRELIMINARY DESIGN OF ROBUST ADAPTIVE CONTROL LAWS
FOR AN AERIAL MANIPULATOR

A THESIS APPROVED FOR THE
SCHOOL OF AEROSPACE AND MECHANICAL ENGINEERING

BY

Dr. Andrea L'Afflitto, Chair

Dr. David Miller

Dr. Wei Sun

Acknowledgements

I would like to first thank Dr. Andrea L'Afflitto of the Gallogly College of Engineering at the University of Oklahoma. I cannot express enough gratitude for all the tools and opportunities he has provided in order to allow me to make my own mark on the field of control of aerial systems. He always was there to listen or steer me in the proper direction of research and writing, while allowing the work to be my own.

I also would like to thank Dr. David Miller and Dr. Wei Sun of the Gallogly College of Engineering at the University of Oklahoma. Their input and comments on this thesis were exceptionally valuable in this process, and I am greatly indebted to them.

To all of my collaborators in the Advanced Control Systems Laboratory at the University of Oklahoma, I extend my deepest thanks for all the time, effort, and insight we shared in our research, classwork, and writing. Their listening, advice, and friendship have been among the highlights of my graduate school experience.

To my fiancé Mackenzie, I want to thank her for her unwavering support throughout this process. Her listening ear and endless provision of coffee provided many welcome moments of joy .

I would like to thank my family and friends for always being there, supporting and encouraging me throughout my undergraduate and graduate career, pushing me to succeed at every step along the way.

This thesis was partly supported by the National Science Foundation through Grant no. 1700640, the US Army Research Lab through Grant no. 40304747, and DARPA under Grant no. D18AP00069. I wish to express my gratitude to these sponsors for their continuous support.

Contents

List of Figures

Abstract

One platform commonly used in modern control systems research is the quadcopter unmanned aerial vehicle (UAV). This platform finds many uses in both civilian and military aviation, such as aerial surveillance and imaging, search and rescue operations, and remote sensing. To perform these tasks, it is increasingly common to rely on autonomous UAVs to allow the vehicle to perform desired tasks without an operator. One weakness of the quadcopter UAV is its underactuation, since this vehicle has six degrees of freedom and only four control inputs. To overcome this complexity, it is proposed to actuate the vehicle propellers, creating a tiltrotor vehicle, in this thesis of the H-configuration. To this end, the equations of motion of the vehicle will be established, and an original robust model reference adaptive control law will be formulated to control the vehicle in the presence of disturbances.

Another current goal in UAV research is in providing a method for the vehicle to manipulate its environment. In this thesis, a two-link robotic manipulator mounted on a cylindrical hinge will be used. This manipulator will have its own trajectory generation and control formulation for its end-effector, after which it will be mounted to the H-configuration tiltrotor. This combined aerial manipulator will be numerically simulated with the manipulator and tiltrotor control laws running simultaneously, demonstrating the feasibility of the combined system.

Keywords: Aerial manipulation, tiltrotor, model reference adaptive control, robust control

1. Introduction

In the study of control systems, one platform that has come into high demand in the past two decades is the quadcopter Unmanned Aerial Vehicle (UAV), due to the platform's high maneuverability, capability to hover, and ability to vertically take off and land. Additionally, the wide range of potential flight profiles allows quadcopters to be used for many purposes, such as aerial surveillance in both military and civilian applications, package delivery, infrastructure inspection, and remote operation and manipulation. One of the benefits of the quadcopter is that the kinematics and dynamics of the platform are well established in the literature [1–3]. However, one difficulty of the classical quadcopter is that it is only actuated in four of the six degrees of freedom, meaning that the vehicle cannot exert thrust forces in the horizontal plane.

This limitation of being unable to exert thrust in the horizontal plane means that quadcopters must pitch or roll to provide lateral thrust, causing a change in attitude of the vehicle, which can be undesirable for the tasks the vehicle is expected to perform. One situation, of particular relevance to the research to be presented, is in the case of a robotic manipulator attached to the quadcopter, because any change in vehicle attitude requires compensation in the manipulator in order to achieve a desired configuration or end-effector position. To solve this problem, the use of a tiltrotor quadcopter, allowing for the ability to exert force in the horizontal plane, is proposed as a solution. In order to create a control algorithm for the tilt-rotor, the

kinematic and dynamic equations of motion must be deduced. Similar to a standard quadcopter, the control algorithm for the tilt-rotor will consider the thrust forces and moment of thrust forces as the control inputs. However, to take advantage of the ability to tilt propellers, a new control input, the horizontal component of the thrust force, will be considered as well.

The tiltrotor quadcopter considered in this thesis will be equipped with a robotic manipulator to enable payload manipulation. In this thesis, the robotic arm will not be considered as integrated with the aerial platform. Instead, the robotic manipulator will be considered as a disturbance, whose dynamics are partly known. The original robust adaptive control laws presented in this thesis will guarantee satisfactory flight performance despite the robotic manipulator's dynamics.

This thesis is structured as follows. In Chapter 2, the literature on standard and tiltrotor UAV's, robotic manipulators, and aerial manipulation system will be examined. Furthermore, the lack of literature on the use of Model Reference Adaptive Control (MRAC) control of tiltrotors and the use of tiltrotors in aerial manipulation will be highlighted as a motivation for this thesis.

In Chapter 3, a MRAC algorithm will be created in order to provide reference trajectory tracking within some user-defined constraints on the trajectory tracking error. Once applied to the tiltrotor quadcopter, this control law allows the vehicle to operate despite unknown inertial properties of both the vehicle and payload, as well as other disturbances, such as wind effects and sensor failure. Initially, the payload modelled will be an inverted double pendulum, with the control algorithm knowing neither the oscillation frequency nor the mass and inertial properties. Numerical

simulation involving original virtual reality models of the tiltrotor quadcopter will demonstrate the robustness of the control algorithm.

In Chapter 4, the manipulator used on the aerial manipulator will be modelled, and the trajectory of the end-effector and control law will be formulated. The kinematics of the manipulator will first be derived, and then an algorithm to generate the end-effector reference trajectory is presented. A Proportional-Integral-Derivative (PID) algorithm to control each of the joint servos will then be formulated. Finally, the manipulator, trajectory generation, and control algorithm will be simulated in the Matlab Simulink environment to demonstrate the efficacy of the chosen control.

In Chapter 5, the manipulator is combined with the aerial platform, creating an aerial manipulator system. The performance of the proposed robust MRAC algorithm will be tested in an original Simulink simulation. A detailed walkthrough on the implementation and use of this simulation will be provided, in order to allows future users to operate the simulation. In the Appendix, the method of moving a Computer Aided Design (CAD) model into the Simulink environment used in this thesis will be presented. This simulator will be made publicly available on the Mathworks website.

2. Literature Review

2.1. Introduction

This thesis is aimed to analyze the dynamics of aerial manipulators and design robust control laws for the vehicles. The aerial manipulators considered in this work comprise of a tilt-rotor quadcopter and a two-link robotic arm. In this chapter, some major highlights on the state of the art on quadcopter unmanned aerial vehicles, tiltrotors, two-link lightweight robotic arms, and aerial manipulators.

2.2. Literature Review on Quadcopters

In current control systems research, one commonly used platform is the quadcopter UAV due to its relatively low operational cost and numerous challenges deriving from the fact that these vehicles are underactuated and unable to transport powerful processing units to execute complex control algorithms. These vehicles are usually employed in activities such as infrastructure inspection, aerial surveillance, and remote sensing. Future applications involving the use of quadcopters include transporting payloads and providing a platform for aerial manipulation.

The simplest way to model the quadcopter is as a rigid body without considering the vehicle aerodynamics [4]. However, this model is inadequate because it fails to compensate for the coupled body and motor dynamics of the quadcopter, and ignores the effects of the rotors themselves [4]. A more complex approach, where the coupled dynamics are considered, is outlined in [2], wherein the quadcopter

model is approximated from the Newton-Euler method. Similarly, the second order motor dynamics are linearized about an operating point, and then combined with the quadcopter modelling [2]. Another approach for the creation of the dynamic model is the Euler-Lagrange Method, demonstrated by [5]. The final dynamics are modelled as the summation of the inertial, Coriolis, centrifugal, and gravitational torques acting upon the quadcopter [5]. The drawback of the above methods is their failure to compensate for any uncertainties of the system in the dynamical model.

While these methods work for the case of a simple quadcopter, they fail to account for cases where the center of mass, payload, and inertial properties are unknown. This proves particularly limiting for quadcopters that must interact with their environment, since the equations of motion of the control algorithm fail to react to system changes [1]. By using a Newton-Euler approach, the equations of motion that express the translational and rotational dynamics and kinematics of the UAV, while capturing the uncertainties of the center of mass and inertia matrix with respect to some reference point can be developed [1]. According to L'Afflitto, this is more suitable to design complex control algorithms able to guarantee satisfactory results in cases where the payload is non-rigidly attached or the vehicle's mass changes over time [1].

Quadcopters are characterized by six degrees of freedom and four control inputs; where the degrees of freedom are the position of the center of mass and the Euler angles identifying its actuation, and the control inputs are the total thrust force and the torques exerted by each propeller. Therefore, quadcopters are underactuated. To overcome this complexity, one solution is to actuate the vehicle's rotors, obtaining

so-called tiltrotors. The alternative is to design autopilots comprised of an outer loop and an inner loop. The outer loop determines the pitch and roll angles needed to rotate the vehicle and use the horizontal output of the thrust force to reach some desired position in the horizontal plane. The inner loop determines the total thrust force and the torque needed to track the reference pitch and roll angles outlined by the outer loop and the reference altitude and yaw angle chosen by the user. The outer loop and the inner loop are therefore cascaded control systems, which can be designed using, for example: PID, backstepping, sliding mode, and adaptive control laws.

One of the simplest control methods is the PID algorithm, as used by [6–10]. To this goal, the equations of motion of the quadcopter are linearized, and a PID control law is used in the outer loop to generate the reference roll and pitch angles. Similarly, in the inner loop, a PID control law is used to track the reference Euler angles and the desired altitude.

PID control laws rely on linearized models of the quadcopter dynamics and hence, are effective in arbitrarily small neighborhoods of the equilibrium condition. However, since the domain of attraction is unknown, the effective range of PID controllers must be estimated in a conservative manner. To overcome this difficulty and allow maneuvers in the presence of substantial uncertainties, nonlinear control techniques should be employed. However, tuning the linear controllers is a significantly faster process due to the availability of heuristic tuning methods such as the Zeigler-Nichols technique or the Tyreus & Luyblen method [11]. Tuning the control parameters in nonlinear algorithms, such as backstepping, sliding mode, and adap-

tive control is a considerably more difficult task for which there is no systematic approach.

The first of these nonlinear control methodologies to examine is backstepping. To create this controller, the dynamical model is rewritten into one that separates the linear translations and the angular rotations [12], [13]. It can be observed that the rotational dynamics of a quadcopter are independent of the translational dynamics, but the translational dynamics depend on the rotational dynamics. This leads to a similar inner loop outer loop structure as described previously, where the outer loop controls translation and the inner loop controls rotation. To control the vehicle, virtual inputs are created, and used to find the thrust force along the z-axis. Once this step is complete, the virtual inputs found are passed along to extract the other three control inputs for the rotation. While the backstepping controller does offer a systematic and continuous way to construct the control inputs, it is not robust to uncertainties, an issue corrected by other control techniques.

One method that accounts for the uncertainties in the system is sliding mode control (SMC). The dynamics of the system are taken and transformed into a pair of differential equations with a virtual input and uncertainties added [14], [15, Ch. 14]. These uncertainties are considered to have an upper bound that can be determined empirically by knowledge of the quadcopter and the potential flight conditions [14]. A sliding surface is created using the trajectory tracking error of the system, and along with the virtual control are used in the Lyapunov equation to find the control input that ensures system stability. From the virtual control, the thrust force control is found in the outer loop translation controller, and the three angular control forces

are found in the rotational controller. The issue with this control methodology is its discontinuity, due to its use of the *signum* function in the formulation of the virtual control, resulting in chattering because of the finite sampling rate [15, Ch. 14]. A solution to this issue is the use of higher order sliding mode control [16] or adaptive control algorithms, such as adaptive sliding mode control (ASMC) [17–19]. The benefit of the ASMC approach is that the control gains are adaptive, reducing the oscillations resulting from the sliding mode controller's chattering behavior.

Another adaptive algorithm for quadcopter control is the model reference adaptive control (MRAC). In order to implement MRAC into the control systems design, one methodology is to use feedback linearization on the inner loop dynamics [18]. Nonlinearities and uncertainties are captured by means of a regressor vector. This control method is faster to implement than ASMC and provides smoother control input that SMC [18]. For these reasons, MRAC was selected to design the quadcopter autopilot in this thesis, and will be examined in great detail in a later chapter.

2.3. Literature Review on Tiltrotors

While the previously examined literature dealt with control strategies for standard quadcopters, a tiltrotor variation of the quadcopter has come into research focus in recent years. Tiltrotors increase the number of available control inputs, and in some configurations are fully actuated [20]. This is achieved through the actuation of the motors and propellers, allowing them to rotate about an axis and change the direction of the thrust generated. There are three main types of quadcopter tiltrotor configurations: the H-configuration demonstrated by [21], the X-configuration

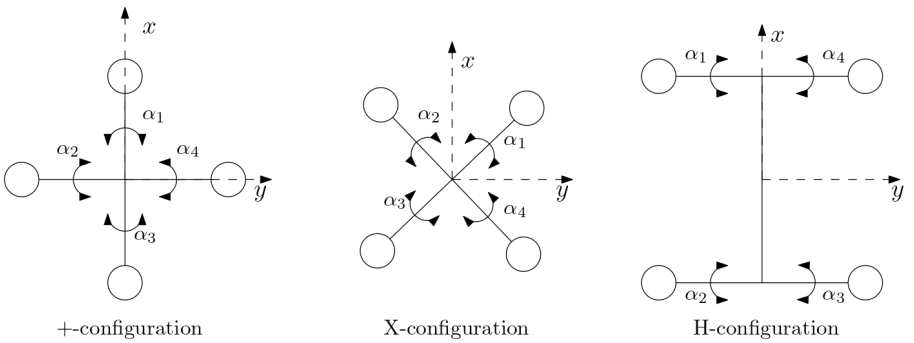

Figure 2.1: Types of Tiltrotor Configuration

demonstrated by [20], and the +-configuration demonstrated by [22]. These configurations can all be seen in Figure 2.1. Since the dynamics of these vehicles are substantially different from one another, autopilots for tiltrotors strongly depend on the configuration case. The +-configuration is fully actuated, that is, characterized by as many degrees of freedom as control inputs. Indeed, it is possible to tilt each propeller and regulate their angular rotation to produce forces along and torques around each of the vehicle's axes. The X and H-configurations are instead under-actuated, having six degrees of freedom and five control inputs. These new control inputs and increased actuation over standard quadcopter systems result in new potential uses for quadcopter systems [23].

2.4. Equations of Motion of a Tiltrotor Quadcopter

Most of the same control strategies applied to standard quadcopters are also applied to tiltrotors, with one notable exception. PID control is not used for the entire vehicle, but rather to control the propellers' tilt angles in [20],[24]. Sliding mode control algorithms for tiltrotors are demonstrated by [20],[25]. For ASMC, successful implementation was achieved by [23],[24]. MRAC control applied to tiltrotor vehi-

cles is presented in [26],[27], and is the basis upon which this thesis is constructed.

2.5. Literature Review on Lightweight Two-Link Robotic Manipulators

Another research area of increasing popularity over the last few decades is mobile robotic manipulation. While robotic manipulation itself dates back to the 1950's, it has recently seen a resurgence thanks to its ability to be integrated into other platforms to increase their potential uses [28, Ch. 1]. In this section, lightweight two-link manipulators and their control will be examined, due to their suitability for aerial manipulation. Two-link manipulators have a wide range of uses due to their low power consumption, light weight, and large range of motion [29]. These advantages pair well with different vehicles, giving the manipulator a platform to increase its range of effect while providing the vehicle with a way to interact with its surroundings [30]. Lightweight manipulators are especially important for aerial manipulation, due to the thrust constraints on the vehicle, so minimizing the manipulator weight increases the flight time and movement capabilities of the quadcopter. One final point about two-link manipulators is that their dynamics and control formulation dates are well-assessed, with most of the current research taking the form of neural network control and pairing the manipulator with computer vision applications. These topics, however, are outside the scope of this thesis.

Control of a robotic manipulator can be broken into three parts: dynamic model formulation, trajectory generation, and control algorithm formulation. In general, the robotic manipulator is modelled as a system of rigid bodies represented as a combination of joints and links [31]. Links are the rigid body lengths between the

joints, and joints are the connection between two links, and are most commonly revolute joints, rotating about one axis [31]. This representation of the manipulator system is ubiquitous in the literature, however, variations still exist in the way it is implemented. One representation is using the Euler-Lagrange method, representing the entire system as generalized motor torques for each joint, with the links themselves modelled as Euler-Bernoulli beams [32–34]. This representation of the dynamical model is very useful for torque control of the manipulator. Alternatively, Newton-Euler method can be applied to finding the dynamics, which is useful for direction joint angle control of the manipulator, demonstrated in [35–37].

Once the dynamic model is established, the range of possible configurations for the arm is known. This is where trajectory generation is beneficial, to allow the manipulator to move end-effector positions in an optimal path while avoiding obstacles and singular configurations [38]. For lightweight two-link manipulators, the most common trajectory generation methodologies are waypoint planning and polynomial planning, where the planning defines the position, velocity, and acceleration of each joint. For waypoint planning, a series of points for the end-effector to pass through are generated, then the desired joint angles solved through the manipulator kinematics. This method also can solve for the joint velocities at all time, but not the acceleration, which can only be set as an end goal. Once the joint positions at the waypoints are found, some optimal trajectory can be found, whether it be minimum time or minimum cost, where minimum cost usually refers to minimizing the control input on the manipulator [38]. Various methods of computing the optimal trajectory are present in the literature, such as dynamic programming [38],

geometric programming [39], or genetic algorithms [40].

For polynomial trajectory generation, the most common polynomial used is a quintic polynomial , due to its ability to create a system of equations for all states of a joint simultaneously [41]. This method offers the advantage of not only generating a smooth position path, but velocity and acceleration as well. This smooth trajectory minimizes the jerk in the system, which in turn proves to track faster and more accurately than non-smooth trajectories [42]. Polynomial trajectory generation only requires the starting state and desired final end state be known, then the coefficients of the quintic polynomial found by solving the system of 6 equations generated [28, Chap. 8], [43]. There are multiple potential poses resulting from the given end-effector positions, something which the user needs some methodology to determine which is more desirable. Once this criteria is determined, it allows all states to be known for all time during the trajectory motion. Furthermore, constraints can be added to any state, ensuring that the trajectory follows some desired behavior for the system.

After defining the desired trajectory of the manipulator, control algorithms can be applied to minimize the tracking error. Many of the control algorithms for manipulators are the same as those used for quadcopter systems, just applied to the robotic manipulator dynamical model instead. There is no need for an outer loop inner loop control design, since every joint can be controlled directly. For both position and torque control, the error is defined as the difference between the desired and actual joint states for a given point along the trajectory [28, Ch. 14], [44], [45].

For torque control methods, a PID controller is able to prove the manipulator

system asymptotically stable [44], [46]. This is done through applying the control to the torque input of the system to account for the error. However, in the case of an unknown mass being picked up by the system, the constraint on the torque bounds cannot be assumed to be met, requiring other control techniques. A similar constraint exists on PID algorithms for position control. If the payload's mass is unknown, it cannot be guaranteed that the system will be stable in all configurations, requiring more robust techniques in order to verify stability [47].

The presence of unknown mass and inertia uncertainties in robotic manipulators requires the use of robust techniques in the formulation of control algorithms. Commonly used in the literature are SMC, ASMC, and MRAC methods. Backstepping control is generally not used for robotic manipulators due to failing to compensate for the unmatched uncertainties [47]. Similarly, feedback linearization is rarely used due to the need to calculate the entire dynamical model in real time, while depending on system parameters which are sensitive to uncertainties [48]. Sliding mode control is used to guarantee system asymptotic stability as long as the desired trajectory is bounded, a condition that can be guaranteed through careful trajectory generation and constraint formulation [48], [49]. It has the benefit of being insensitive to uncertainties in the system model, provided the sliding manifold is designed in such a way as to compensate for the unknown part of the system dynamics [49]. The issue with using SMC for torque control units is that the chattering in the system is magnified by the use of Pulse-Width Modulation (PWM) inputs, resulting in systems that heavily oscillate about the goal point. While less extreme on position control systems, the strain placed on the actuators by the chattering effect promotes

the implementation of adaptive control methods [48].

One final control method to discuss for lightweight manipulators is MRAC control. MRAC control is fairly straightforward to implement in two-link manipulators due to the ideal system dynamics at a set trajectory point being relatively easy to find [33]. This provides a relatively simple calculation, unlike the high computational load required to calculate the actual system dynamics in real time [33]. However, many modelling assumptions must be made, such as known payload mass and perfect joint structure, resulting in a large error between the actual and ideal system responses that must be corrected by the MRAC algorithm. Furthermore, the linearization of the ideal system may not be valid for all manipulator configurations, due to straying too far from the operating point [50]. This in turn limits the valid operating range of the manipulator, reducing its efficacy in the tasks asked of it. Hence, ASMC is the much more popular control algorithm for robotic manipulation [50].

ASMC provides the chattering reduction that MRAC does while also providing a wider stability region. For this reason, they are much more common in the literature. One method to achieve fast convergence without chattering is to set the control gains to be proportional to the sliding variables while at some distance away from the manifold, whereas in the neighborhood of the sliding manifold the control gains are set to be inversely proportional to the sliding variables [45]. This greatly reduces chattering, at the cost of some of the tracking speed, which is why the addition of a pole placement controller in addition to the ASMC algorithm is recommended [45]. Alternatively, the adaptive gains can be calculated outside the

sliding mode algorithm, increasing the implementation cost of the algorithm, but providing faster convergence and more robust performance [51]. ASMC algorithms have been successfully implemented into many lightweight two-link manipulators, as evidenced by [45],[51–53].

Finally, there are other control methods that have become increasingly popular in robotic manipulation in recent years. These methods include neural networks [54], [55], fuzzy-logic controllers [56], [57], and machine learning algorithms [58], [59]. While results from these control methods are promising, they are highly specialized topics requiring in-depth study to successfully implement, and as such are outside the scope of this thesis.

2.6. Literature Review of Aerial Manipulation

Combining the two topics previously examined creates a new area of study, aerial manipulation. While the use of UAVs as aerial platforms for carrying systems such as cameras, sensors, and payloads is not new, the use of quadcopters as carriers for robotic manipulators has only taken off in the last decade. Currently, there are two main categories of aerial manipulation. First, the use of one degree of freedom grippers used to transport objects [60]. In this system, the gripper is rigidly attached to the vehicle, and requires the vehicle to be in a specific position to grasp an object, due to the fixed configuration of the gripper. The second method is the use of a robotic manipulator, such as the two-link manipulators previously discussed, attached to a quadcopter allowing the vehicle to interact with the environment, due to the manipulator being actuated and able to move independently of the vehicle

[60]. It is this application of aerial manipulation that we are interested in, due to the variety of tasks that can be performed, thanks to the wider range of system configurations granted by the robotic manipulator. These tasks include object manipulation, tool operation, and environment interaction. As with quadcopters and manipulators, the dynamic model of the aerial manipulator system must be found before control synthesis can occur.

The first method of creating the equations of motion for the system is to model the quadcopter as previously demonstrated, and treat the manipulator as a disturbance [61]. The first step of this process is to find the equations of motion of the manipulator, which is well-defined for two-link lightweight manipulators as previously demonstrated. Following from this derivation, the manipulator mass, moments of inertia, and movement states are all added as a disturbance to the quadcopter model [61]. The disturbance due to the motion of the manipulator is modelled as a non-rigidly attached payload disturbance, due to the changing of the inertia and center of mass of the combined system due to its movements. This, in turn, allows payloads carried by the manipulator to simply be folded into this disturbance without requiring exact knowledge of the payloads mass properties [61]. This method is extremely limiting for the increased levels of robustness required of the quadcopter control algorithm, and proves to highly impede the movement and manipulation capabilities of the aerial manipulator

The second method for formulation of the equations of motion is to treat the arm and quadcopter as one system, demonstrated by [62], [63]. Here, the dynamic model of the entire system is found by combining the dynamic model of the robotic

manipulator with the dynamic model of the quadcopter. This method allows for better control of the aerial manipulator, due to the coupling between the two systems' dynamics being accounted for in this model.

This coupling between the quadcopter's and manipulator's dynamical models mean's that the vehicle's stability depends on the manipulator configuration [64]. To account for the coupled dynamics, the entire system is considered when constructing the quadcopter control laws in [63],[64]. However, this approach is computationally expensive and complex, so an alternative approach controlling the system is provided in [30],[65]. In this method, the computation of the dynamics of the quadcopter and manipulator are performed separately, but with consideration for the other system. This requires that each system has knowledge of the others' position, velocity, and acceleration within some bounded uncertainty, and is the method that will be employed in this thesis. This assumption is easier to make with the introduction of the tiltrotor platform to the system, due to the increase in possible flight configurations to counteract the position of the manipulator.

When controlling an aerial manipulator system, examples in the literature exist for using ASMC [65], [66], MRAC [61], and inverse dynamic techniques [63], [67] to control the vehicle, while PID and SMC techniques are used to control the manipulator. These systems all demonstrate simple trajectory tracking like circle tracking and hovering while performing manipulation. However, none of these methods account for the interaction of the manipulator with the environment. Instead, the reaction forces of the environment on the manipulator are assumed negligible. Impedance control offers a way to account for the reaction forces due to its active elasticity,

which in turns provides a method to reduce interaction force on the vehicle frame during manipulator contact with the environment [68]. The control method works to actively minimize the reaction force through minimization of the force output of the manipulator. However, this methodology requires the assumption that the quadcopter is at or near a hovering flight configuration during manipulator motion, in turn allowing the simplifying assumption that aerodynamic forces are not acting on the manipulator during its actuation [68]. The handicap of this method is its assumption that the manipulator mass is negligible compared to the vehicle mass, an assumption not made by any of the previously examined literature.

To attempt to overcome the small manipulator handicap for impedance control, Cartesian Impedance Control (CIC) can applied to the aerial manipulation system [62]. The uniqueness of this method is its independence from external force measurements usually required in impedance control, meaning that disturbances on the system such as unknown mass of the payload can be accounted for in the control algorithm [62]. This provides a minimization of reaction forces on the aerial manipulator during interaction with the environment, but does not allow the system to fly at a states that are not near hover.

Finally, it was attempted to find literature sources for the implementation of robotic manipulators on tiltrotor UAVs, however, there currently is no literature on these sort of applications. While tools have been rigidly attached to tiltrotors, as demonstrated by [69], [70], no system more complex than a 1 degree of freedom gripper can be found. While many authors point out that the increased control and range of motion provided by tiltrotors would be a benefit in manipulating the

environment, none go so far as to implement such a system. In fact, all the aerial manipulation examples above rely on the vehicle being at or near a hovering state to actuate the manipulator, due to the vehicle not being able to account for the manipulator motion in its own control. This is a weakness that this thesis hopes to overcome with its use of a tiltrotor system. Due to the lack of literature on tiltrotor aerial manipulators, this thesis hopes to provide a starting point for future writings on this subject.

3. Control of a Tiltrotor Quadcopter

3.1. Introduction

In this chapter we derive the equations of motion for a H-configuration tiltrotor quadcopter. Following this, a robust MRAC control law is presented for the control of the vehicle. Finally, a numerical example illustrates the theoretical results discussed. The contents of this chapter are primarily based on [26], [27], and all theoretical results are from the above sources, unless otherwise cited.

3.2. Equations of Motion of a Tiltrotor Quadcopter

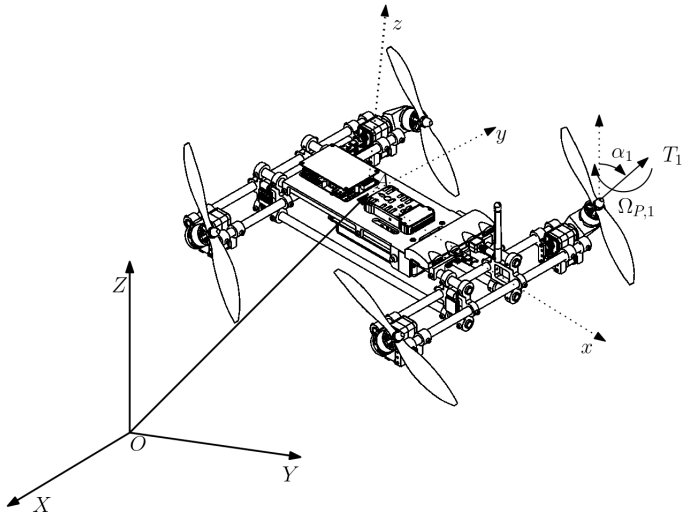

Figure 3.1: Representation of the Tiltrotor System

First, the H-configuration tiltrotor is considered to be comprised of a frame, four propellers, and a payload. The frame itself is considered as a rigid body, and the four propellers are considered to be independent from one another, that is, each can be tilted without consideration to another.

Finally, the payload is considered to be non-rigidly attached to the tiltrotor frame, resulting in an unknown position of the center of mass and an unknown inertia matrix, both of which vary in time. Furthermore, the mass of the payload is considered to be significant with respect to the mass of the body, meaning the variations in the center of mass and inertia matrix are not negligible.

The position of the tiltrotor's frame can be defined through the creation of an inertial orthonormal reference frame $\mathbb{I} = \{O; X, Y, Z\}$, where $O \in \mathbb{R}^3$ is the origin and $X, Y, Z \in \mathbb{R}^3$ are the axes; along with a body reference frame $\mathbb{J}(\cdot) = \{A(\cdot); x(\cdot), y(\cdot), z(\cdot)\}$ originating at some chosen reference point on the quadcopter frame $A : [t_0, \infty) \to \mathbb{R}^3$, with axes $x, y, z : [t_0, \infty) \to \mathbb{R}^3$ [71, pg. 11].

The inertial reference frame $\mathbb{I}$ will be oriented such that the force due to gravity is defined as $F_g^{\mathbb{I}} = -mgZ$, where m denotes the mass of the vehicle and g denotes the acceleration due to gravity. The mass m and gravitational acceleration g are both assumed to be greater than zero and constant. The body frame is oriented such that the $y(\cdot)$ is parallel to the axis about which the propellers rotate, and $x(\cdot)$ and $z(\cdot)$ are oriented so if $x(t) = X$ and $y(t) = Y$, $t \geq t_0$ then $z(t) = Z$.

Next, let us define the translation and orientation between the body and inertial reference frame. The position between the inertial frame origin O and body reference point $A(\cdot)$ is defined by the vector $r_A^{\mathbb{I}} : [t_0, \infty) \to \mathbb{R}^3$, where the superscript $\mathbb{I}$ denotes

that this vector is expressed in the inertial frame. Similarly, the velocity of $A(\cdot)$ with respect to the inertial reference frame is given by $v_A^{\mathbb{I}} : [t_0, \infty) \to \mathbb{R}^3$. The orientation of the body frame with respect to the inertial frame can be captured by a 3-2-1 rotation sequence. The angles of rotation are the roll angle $\phi : [t_0, \infty) \to [0, 2\pi)$, pitch angle $\theta : [t_0, \infty) \to (-\frac{\pi}{2}, \frac{\pi}{2})$, and yaw angle $\psi : [t_0, \infty) \to [0, 2\pi)$ [71, pg. 11]. Finally, let $\Omega_i : [t_0, \infty) \to \mathbb{R}$ define the angular position of the ith propeller about its spin axis, where $i = 1, \ldots, 4$, due to the number of propellers. Furthermore, a new term is added to the equations of motion, the propeller tilt angle, represented by $\alpha_i(\cdot)$, $i = 1, \ldots, 4$. From the above, all the terms needed to capture the equations of motion for an H-configuration tiltrotor are defined.

3.2.1. Kinematic Equations of a Tiltrotor Quadcopter

First, let the position and orientation of the tiltrotor be defined by the vector of independent generalized coordinates

$$q(t) \triangleq \left[(r_A^{\mathbb{I}}(t))^{\mathrm{T}}, \phi(t), \theta(t), \psi(t) \right]^{\mathrm{T}}, t \geq t_0, \tag{3.1}$$

In turn, let the angular velocity of the body reference frame with respect to the inertial frame be captured by $\omega : \mathcal{D} \times \mathbb{R}^6 \to \mathbb{R}^3$. Let $\mathcal{D}$ be defined as follows

$$\mathcal{D} \triangleq \mathbb{R}^3 \times [0, 2\pi) \times \left(-\frac{\pi}{2}, \frac{\pi}{2} \right) \times [0, 2\pi), \tag{3.2}$$

therefore, from [71, Th. 1.7], the final angular velocity of $\mathbb{J}$ with respect to $\mathbb{I}$ can be captured as

$$\omega(q(t), \dot{q}(t)) = \begin{bmatrix} 1 & \sin\phi(t)\tan\theta(t) & \cos\phi(t)\tan\theta(t) \\ 0 & \cos\phi(t) & -\sin\phi(t) \\ 0 & \sin\phi(t)\sec\theta(t) & \cos\phi(t)\sec\theta(t) \end{bmatrix}^{-1} \begin{bmatrix} \dot{\phi}(t) \\ \dot{\theta}(t) \\ \dot{\psi}(t) \end{bmatrix} \qquad t \geq t_0. \quad (3.3)$$

The matrix in Equation (3.3) is invertible, due to $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$, and henceforth let it be known as $\Gamma(q(t))$. From the above equation the kinematic equations of motion can be built as in [71, pg. 18-19]

$$\dot{q}(t) = \begin{bmatrix} v_A^{\mathbb{I}}(t) \\ \dot{\phi}(t) \\ \dot{\theta}(t) \\ \dot{\psi}(t) \end{bmatrix}, \qquad q(t_0) = \begin{bmatrix} r_{A,0}^{\mathbb{I}} \\ \phi_0 \\ \theta_0 \\ \psi_0 \end{bmatrix}, \qquad t \geq t_0. \quad (3.4)$$

3.2.2. Dynamic Equations of a Tiltrotor Quadcopter

First, the translational dynamic equation will be defined, starting with the 3-2-1 rotation sequence from the inertial frame to the body frame [1]

$$R(q) \triangleq \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}, \qquad q \in \mathcal{D}. \quad (3.5)$$

From here, the standard translational dynamic equations as illustrated in [1] is created

$$
\begin{aligned}
&m\dot{v}_A^{\mathbb{I}}(t) + mR(q(t))r_C^{\times}(t)\dot{\omega}(q(t),\dot{q}(t)) \\
&\qquad = R(q(t))\begin{bmatrix} u_5(t) \\ 0 \\ u_1(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} - mR(q(t))\left[\ddot{r}_C(t) + 2\omega^{\times}(q(t),\dot{q}(t))\dot{r}_C(t)\right. \\
&\qquad \left. +\omega^{\times}(q(t),\dot{q}(t))\omega^{\times}(q(t),\dot{q}(t))r_C(t)\right], \qquad v_A^{\mathbb{I}}(t_0) = v_{A,0}^{\mathbb{I}}, \qquad t \geq t_0,
\end{aligned}
\tag{3.6}
$$

where $u_5 : [t_0, \infty) \to \mathbb{R}$ denotes the force component along the $x(\cdot)$ axis and is a new term introduced for the H-configuration tiltrotor. The force component along the $z(\cdot)$ axis is denotes as $u_1 : [t_0, \infty) \to \mathbb{R}$, gravitational acceleration denoted by $g > 0$, and vehicle center of mass with respect to the reference point denoted by $A(\cdot)$, $r_C(\cdot) : [t_0, \infty) \to \mathbb{R}^3$, are all standard terms in the translational dynamic equation [1]. To model the rotational dynamic equation, the standard equation set forth in

[1] is used,

$$
\begin{aligned}
&I(t)\dot{\omega}(q(t),\dot{q}(t)) + mr_C^{\times}(t)R^{\mathrm{T}}(q(t))\dot{v}_A^{\mathbb{I}}(t) \\
&\qquad = \begin{bmatrix} u_2(t) \\ u_3(t) \\ u_4(t) \end{bmatrix} - \omega^{\times}(q(t),\dot{q}(t))I(t)\omega(q(t),\dot{q}(t)) - \dot{I}(t)\omega(q(t),\dot{q}(t)) \\
&\qquad - \omega^{\times}(q(t),\dot{q}(t))\sum_{i=1}^{4} I_{P_i}(t)\omega_{P_i}(t) - \sum_{i=1}^{4}\left[I_{P_i}(t)\dot{\omega}_{P_i}(t) + \omega_{P_i}^{\times}(t)I_{P_i}(t)\omega_{P_i}(t)\right] \\
&\qquad\qquad + r_C^{\times}(t)F_g(q(t)), \qquad \omega(t_0) = \omega_0, \qquad t \geq t_0.
\end{aligned} \tag{3.7}
$$

In the previous equation the vehicles weight due to gravity is denoted by [71, p. 26]

$$
F_g(q) = R^{\mathrm{T}}(q)\begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix}, \qquad q \in \mathcal{D}, \tag{3.8}
$$

the moments of force applied to the propellers denoted by $[u_2, u_3, u_4]^{\mathrm{T}} : [t_0, \infty) \to \mathbb{R}^3$, and the inertia matrix with respect to the reference point $A(\cdot)$ of the vehicle given by

$$
I(t) \triangleq -\int_{\mathcal{V}} r_{mA}^{\times}(t) r_{mA}^{\times}(t)\delta m, \qquad t \geq t_0, \tag{3.9}
$$

where $\mathcal{V} \subset \mathcal{R}^3$ is a volume containing the vehicle and $r_{mA} : [t_0, \infty) \to \mathcal{V}$ is the position of an infinitesimal mass δm with respect to the $A(\cdot)$. Furthermore, the

inertia matrix of the ith propeller is given by

$$I_{P_i}(t) \triangleq -\int_{\mathcal{P}_i} r_{mA}^{\times}(t) r_{mA}^{\times}(t) \delta m, \tag{3.10}$$

where $\mathcal{P}_i \subset \mathcal{R}^3$ denotes a volume containing exclusively the propeller, and $i = 1, \ldots, 4$ denotes the propeller number.

From (3.6) and (3.7), the final equations of motion are formulated as follows

$$\mathcal{M}(t, q(t)) \begin{bmatrix} \dot{v}_A^{\mathbb{I}}(t) \\ \dot{\omega}(q(t), \dot{q}(t)) \end{bmatrix} = \begin{bmatrix} f_{\text{tran}}(t, q(t), \dot{q}(t)) \\ f_{\text{rot}}(t, q(t), \dot{q}(t)) \end{bmatrix} + \hat{G}(q(t)) u(t),$$
$$\begin{bmatrix} v_A^{\mathbb{I}}(t_0) \\ \omega(t_0) \end{bmatrix} = \begin{bmatrix} v_{A,0}^{\mathbb{I}} \\ \omega_0 \end{bmatrix}, \qquad t \geq t_0. \tag{3.11}$$

In the above equation, $\mathcal{M}(t, q)$ is the generalized mass matrix and given by

$$\mathcal{M}(t, q) \triangleq \begin{bmatrix} m\mathbf{1}_3 & -mR(q) r_C^{\times}(t) \\ m r_C^{\times}(t) R^{\mathrm{T}}(q) & I(t) \end{bmatrix}, \qquad (t, q) \in [t_0, \infty) \times \mathcal{D}, \tag{3.12}$$

where $\mathbf{1}_n$ denotes the identity matrix of size $\mathbb{R}^{n \times n}$, $f_{\text{tran}}(t, q, \dot{q})$ is the translational dynamic equation given by

$$f_{\text{tran}}(t, q, \dot{q}) \triangleq [0, 0, -mg]^{\mathrm{T}} - mR(q)[\ddot{r}_C(t) + 2\omega^{\times}(q, \dot{q}) \dot{r}_C(t) + \omega^{\times}(q, \dot{q}) \omega^{\times}(q, \dot{q}) r_c^{\mathrm{T}}], \tag{3.13}$$

the rotational dynamic equation $f_{\text{rot}}(t, q, \dot{q})$ given by

$$\begin{aligned} f_{\text{rot}}(t, q, \dot{q}) \triangleq & -\omega^{\times}(q, \dot{q}) I(t) \omega(q, \dot{q}) - \dot{I}(t) \omega(q, \dot{q}) \\ & - \sum_{i=1}^{4} [I_{P_i}(t) \dot{\omega}_{P_i}(t) + \omega_{P_i}^{\times}(t) I_{P_i}(t) \omega_{P_i}(t)] - \omega^{\times}(q, \dot{q}) \sum_{i=1}^{4} I_{P_i}(t) \omega_{P_i}(t) \\ & + m r_C^{\times}(t) \omega^{\times}(q, \dot{q}) v_A(t) + r_C^{\times}(t) F_g(q), \end{aligned} \tag{3.14}$$

$\hat{G}(q)$ is given by

$$\hat{G}(q) = \begin{bmatrix} R(q) \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} & 0_{3\times 3} \\ 0_{3\times 2} & \mathbf{1}_3 \end{bmatrix}, \tag{3.15}$$

and finally, the control input $u(t)$ is given by

$$u = [u_5, u_1, u_2, u_3, u_4]^{\mathrm{T}}. \tag{3.16}$$

Of note in the above equations is that the propeller inertia matrix I_{P_i} is modelled as a thin disk, resulting in the following

$$I_{\text{disk}} = m_{\text{prop}} \frac{\rho_{\text{prop}}^2}{4} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \tag{3.17}$$

where ρ_{prop} is the radius of the propeller and m_{prop} is the mass of the propeller. This

in turn leads us to the final equation for the propeller inertia matrix,

$$I_{P_i}(t) = R_2(\alpha_i(t))I_{\text{disk}}R_2^{\text{T}}(\alpha_i(t)) + m_{\text{prop}}r_{\text{prop},i}^{\text{T}}r_{\text{prop},i}\mathbf{1}_3 - m_{\text{prop}}r_{\text{prop},i}r_{\text{prop},i}^{\text{T}},$$
$$t \geq t_0. \quad (3.18)$$

where the $R_2(\cdot)$ term is given by

$$R_2(\alpha_i(t)) = \begin{bmatrix} \cos(\alpha_i(t)) & 0 & -\sin(\alpha_i(t)) \\ 0 & 1 & 0 \\ \sin(\alpha_i(t)) & 0 & \cos(\alpha_i(t)) \end{bmatrix}, \qquad t \geq t_0, \tag{3.19}$$

and $r_{\text{prop},i}$ is the distance from the center of the ith propeller to the reference point $A(\cdot)$.

3.2.3. Tiltrotor Gyroscopic effect

In standard quadcopter dynamics, the term $\sum_{i=1}^{4} I_{P_i}(t)\dot{\omega}_{P_i}(t)$, $t \geq t_0$ is known as the inertial counter-torque, and $\omega^{\times}(t)\sum_{i=1}^{4} I_{P_i}(t)\omega_{P_i}(t)$, $t \geq t_0$ is the gyroscopic effect. Both of these terms can be seen in Equation (3.14). However, in standard quadcopter dynamics, the term $\sum_{i=1}^{4} \omega^{\times}(t)I_{P_i}(t)\omega_{P_i}(t)$ is ignored, due to $\alpha_i(t) \equiv 0$ for non-tilting propellers, resulting in $\sum_{i=1}^{4} \omega^{\times}(t)I_{P_i}(t)\omega_{P_i}(t) \equiv 0$. For the tiltrotor, due to $\alpha_i(t)$ not being identically equal to 0 by assumption, this term cannot be ignored, resulting in the naming of a new term, the tiltrotor gyroscopic effect [26]. This term, while neglected in conventional quadcopter dynamics, cannot be for tiltrotor vehicles, due to the addition of propeller rotation.

3.3. Robust Control Formulation for a Tiltrotor

The goal of the robust control algorithm is to allow the vehicle to track some reference trajectory of the unknown dynamical system while maintaining to some user defined tracking error and control input constraints.

3.3.1. Modeling the Dynamical System

First thing in the formulation of the control algorithm is the creation of the unknown dynamical system,

$$\dot{x}(t) = Ax(t) + B[u(t) + \Theta^{\mathrm{T}}\Phi(x(t))] + \xi(t), \qquad x(t_0) = x_0, \qquad t \geq t_0. \tag{3.20}$$

In the above equation the known parameters are the system trajectory, control input, and regressor vector, represented by $x(t) \in \mathcal{D} \subseteq \mathbb{R}^n$, $t \geq t_0$, $u(t) \in U \subseteq \mathbb{R}^m$, $t \geq t_0$, and $\Phi : \mathcal{D} \to \mathbb{R}^N$, respectively, as well as $B \in \mathbb{R}^{n\times m}$. The unknowns are A, Θ, and $\xi(t)$, where $A \in \mathbb{R}^{n\times n}$ and $\Theta \in \mathbb{R}^{N\times m}$ capture the parametric and matched uncertainties, respectively, and $\xi(t) : [t_0, \infty) \to \mathbb{R}^n$ is continuous and captures the unmatched uncertainties where the upper bound $||\xi(t)|| \leq \xi_{\max}$, $t \geq t_0$ is known.

Furthermore, the reference dynamical model is represented by

$$\dot{x}_{\mathrm{ref}} = A_{\mathrm{ref}}x_{\mathrm{ref}}(t) + B_{\mathrm{ref}}r(t), \qquad x_{\mathrm{ref}} = x_{\mathrm{ref},0}, \qquad t \geq t_0, \tag{3.21}$$

where $x_{\mathrm{ref}}(t) \in \mathbb{R}^n$, $A_{\mathrm{ref}} \in \mathbb{R}^{n\times n}$ is Hurwitz, $B_{\mathrm{ref}} \in \mathbb{R}^{n\times m}$, and $(A_{\mathrm{ref}}, B_{\mathrm{ref}})$ controllable. Additionally, the command input $r(t) \in \mathbb{R}^m$ is bounded, and $t \geq t_0$. Next,

matching conditions are created as follows

$$A_{\text{ref}} = A + BK_x^{\text{T}}, \tag{3.22}$$

$$B_{\text{ref}} = BK_r^{\text{T}}, \tag{3.23}$$

where $(K_x, K_r) \in \mathbb{R}^{n\times m} \times \mathbb{R}^{m\times m}$ are assumed to exist, as is common in the literature [18],[26]. Equation (3.22) and Equation (3.23) imply that in the absence of uncertainties, then the actual system trajectory would converge to the ideal system trajectory with zero error, where the error is defined as

$$e(t) \triangleq x(t) - x_{\text{ref}}(t), \qquad t \geq t_0. \tag{3.24}$$

However, since A and Θ are unknown, then it follows that $K \triangleq [K_x^{\text{T}}, K_r^{\text{T}}, -\Theta^{\text{T}}], \in \mathbb{R}^{m\times(n+m+N)}$ is similarly unknown. Instead, let an estimate of K be defined, $K_e \in \mathbb{R}^{m\times(n+m+N)}$ such that the Frobenius norm, denoted by $||\cdot||_{\text{F}}$, of their difference fits within some bound $\epsilon \geq 0$, as shown

$$||K_e - K||_{\text{F}} \leq \epsilon. \tag{3.25}$$

To demonstrate, consider that there is a feedback control law

$$\phi(\pi, \hat{K}) = \hat{K}\pi, \qquad (\pi, \hat{K}) \in \mathbb{R}^{n+m+N} \times \mathbb{R}^{m\times(n+m+N)}, \tag{3.26}$$

where $\pi(t) \triangleq [x^{\text{T}}(t), r^{\text{T}}(t), \Phi^{\text{T}}(x(t))]$, and the adaptive gain matrix is given by $\hat{K}(t)$:

$[t_0, \infty) \to \mathbb{R}^{m\times(n+m+N)}$. This allows the creation of the estimated and actual adaptive gain error, defined as $\Delta K \triangleq \hat{K}(t) - K_e$, $t \geq t_0$ and $\widetilde{\Delta K}(t) \triangleq \hat{K}(t) - K$, $t \geq t_0$, respectively. Recall Equation (3.25), and applying the same methodology, note that

$$||\widetilde{\Delta K}(t) - \Delta K||_{\mathrm{F}} \leq \epsilon, \qquad t \geq t_0. \tag{3.27}$$

3.3.2. Constraint Formulation and Properties

Now, a compact, connected constraint set is created

$$\mathcal{C} \triangleq \{(e, \Delta K) \in \mathbb{R}^n \times \mathbb{R}^{m\times(n+m+N)} : h(e^{\mathrm{T}}Me, \Delta K\Gamma^{-1}\Delta K^{\mathrm{T}} \geq 0)\}. \tag{3.28}$$

In the above equation, $e(t)$ is the trajectory tracking error $x(t) - x_{\mathrm{ref}}(t)$, $t \geq 0$, and $M \in \mathbb{R}^{n\times n}$ is user defined to weight the constraints on e where $M = M^{\mathrm{T}} > 0$. Similarly, $\Gamma \in \mathbb{R}^{(n+m+N)\times(n+m+N)}$ is the user defined constraints on ΔK, where $\Gamma = \Gamma^{\mathrm{T}} > 0$. Finally, $h : \mathbb{R} \times \mathbb{R}^{m\times m} \to \mathbb{R}$ is both continuously differentiable and such that $h(0, 0) > 0$. The reason $\mathcal{C}$ is given as both compact and connected is so that it is both able to capture bounded constraint sets, and that there always is a subset of the interior $\mathring{\mathcal{C}}$ that simultaneously contains both $(e(t_0), \Delta K(t_0))$ and $(0_n, 0_{m\times(n+m+N)})$ that exists without the ability to be expressed as two disjoint non-empty sets.

Next,

$$h_e(e^{\mathrm{T}}Me, \Delta K\Gamma^{-1}\Delta K^{\mathrm{T}}) \triangleq \left.\frac{\partial h(\beta, X)}{\partial \beta}\right|_{\substack{\beta=e^{\mathrm{T}}Me,\\ X=\Delta K\Gamma^{-1}\Delta K^{\mathrm{T}}}}, \tag{3.29}$$

$$h_X(e^{\mathrm{T}}Me, \Delta K\Gamma^{-1}\Delta K^{\mathrm{T}}) \triangleq \left.\frac{\partial h(\beta, X)}{\partial \beta}\right|_{\substack{\beta=e^{\mathrm{T}}Me,\\ X=\Delta K\Gamma^{-1}\Delta K^{\mathrm{T}}}} \tag{3.30}$$

where $h_e : \mathbb{R} \times \mathbb{R}^{m\times m} \to \mathbb{R}$ and $h_X : \mathbb{R} \times \mathbb{R}^{m\times m} \to \mathbb{R}^{m\times m}$ for all $(e, \Delta K) \in \mathring{\mathcal{C}}$. Furthermore, it is assumed that

$$h_e(e^{\mathrm{T}}Me, \Delta K\Gamma^{-1}\Delta K^{\mathrm{T}}) \leq 0, \tag{3.31}$$

$$h_X(e^{\mathrm{T}}Me, \Delta K\Gamma^{-1}\Delta K^{\mathrm{T}}) \leq 0, \tag{3.32}$$

where $(e, \Delta K) \in \mathring{\mathcal{C}}$. It follows then that

$$(0_n, 0_{m\times(n+m+N)}) = \underset{(e,\Delta K)\in\mathbb{R}^n\times\mathbb{R}^{m\times(n+m+N)}}{\arg\max} h(e^{\mathrm{T}}Me, \Delta K\Gamma^{-1}\Delta K^{\mathrm{T}}), \tag{3.33}$$

and that $h_X(\cdot,\cdot) = h_X^{\mathrm{T}}(\cdot,\cdot) \leq 0$.

The next property of $\mathcal{C}$ to note is that the interior

$$\mathring{\mathcal{C}} = \{(e, \Delta K) \in \mathbb{R}^n, \mathbb{R}^{m\times(n+m+N)} : h(e^{\mathrm{T}}Me, \Delta K\Gamma^{-1}\Delta K^{\mathrm{T}}) > 0\} \tag{3.34}$$

is not empty, due to the previously defined properties of $h(0,0) > 0$ and $h(\cdot,\cdot)$ is continuous. From the above equations, it is seen that $h(e^{\mathrm{T}}Me, \cdot)$ must be a strictly non-increasing function where $e^{\mathrm{T}}Me = ||M^{\frac{1}{2}}e||^2$ and $(e, \Delta K) \in \mathring{\mathcal{C}}$ and $h(\cdot, \Delta K\Gamma^{-1}\Delta K^{\mathrm{T}})$

is chosen such that when $\Delta K = 0$, the maximum of $h(\cdot, \Delta K \Gamma^{-1} \Delta K^{\mathrm{T}})$ is achieved. For example, in [26],

$$h(e^{\mathrm{T}} M e, \Delta K \Gamma^{-1} \Delta K^{\mathrm{T}}) = h_{\max} - ||M^{\frac{1}{2}} e||^2 - ||\Delta K||^2_{\mathrm{F},\Gamma^{-1}},$$
$$(e, \Delta K) \in \mathbb{R}^m \times \mathbb{R}^{m \times (n+m+N)}, \quad (3.35)$$

where $h_{\max} > 0$ is chosen that verifies Equations (3.31)-(3.33). This is achieved through the fact that $h(0,0) = h_{\max} > 0$, $h_e(e^{\mathrm{T}} M e, \Delta K \Gamma^{-1} \Delta K) = -1$, and $h_X(e^{\mathrm{T}}(t) M e, \Delta K \Gamma^{-1} \Delta K^{\mathrm{T}}) = -\mathbf{1}_m$.

3.3.3. Adaptive Law Formulation

The goal of formulating the adaptive law $\hat{K}(\cdot)$ is to ensure that all sources of error maintain boundedness inside the user-defined constraints, despite the presence of uncertainties. The input to do so will be in the form $u(t) = \phi(\pi(t), \hat{K}(t))$, $t \geq t_0$. By combining the actual and ideal dynamical systems along with the feedback control law, the following differential equation is found

$$\dot{e}(t) = A_{\mathrm{ref}} e(t) + B \widetilde{\Delta K}(t) \pi(t) + \xi(t), \qquad e(t_0) = x_0 - x_{\mathrm{ref},0}, \qquad t \geq t_0, \quad (3.36)$$

as well as the function

$$V(e, \Delta K) \triangleq \frac{e^{\mathrm{T}} P e + \mathrm{tr}(\Delta K \Gamma^{-1} \Delta K^{\mathrm{T}})}{h(e^{\mathrm{T}}(t) M e, \Delta K \Gamma^{-1} \Delta K^{\mathrm{T}})}, \qquad (e, \Delta K) \in \mathring{\mathcal{C}}, \quad (3.37)$$

where $V(\cdot,\cdot)$ is positive definite, and $P \in \mathbb{R}^{n\times n}$ is such that $P = P^{\mathrm{T}} > 0$ and is a solution of the algebraic Lyapunov equation. This equation is given as

$$0 = A_{\mathrm{ref}}^{\mathrm{T}} P + P A_{\mathrm{ref}} + Q_1, \tag{3.38}$$

where $Q_1 \in \mathbb{R}^{n\times n}$ has the properties $Q_1 = Q_1^{\mathrm{T}} > 0$. Next, taking

$$\mathcal{S}_\pi \triangleq \{(e, \Delta K) \in \mathbb{R}^n \times \mathbb{R}^{(n+m+N)} : S_\pi(e, \Delta K) > 0\}, \tag{3.39}$$

and by creating some $\alpha > 0$, $\sigma > 0$, $p \in \mathbb{R}$,, and $\pi \in \mathbb{R}^{n+m+N}$, the result is

$$S_\pi(e, \Delta K) \triangleq -\alpha||e||^2 - \sigma||e^{\mathrm{T}} P N||^p ||\Delta K||_{\mathrm{F}}^2 + 2(\epsilon||\pi|| + \xi_{\max})||R(e, \Delta K)||_{\mathrm{F}}, \tag{3.40}$$

where

$$R(e, \Delta K) \triangleq e^{\mathrm{T}}[P - V(e, \Delta K) h_e(e^{\mathrm{T}}(t) M e, \Delta K \Gamma^{-1} \Delta K^{\mathrm{T}}) M] B. \tag{3.41}$$

It also can be noted that $x_{\mathrm{ref}}(t), t \geq t_0$ is bounded, due to the fact that A_{ref} is Hurwitz and $r(t)$ is bounded. Additionally, if the error $e(\cdot)$ is bounded, then $\pi(\cdot)$ exists in some such compact set $\Pi \subset \mathbb{R}^{n+m+N}$ where $\pi(t) = [x^{\mathrm{T}}(t), r^{\mathrm{T}}(t), -\Phi^{\mathrm{T}}(x(t))]^{\mathrm{T}} \in \Pi$, $t \geq t_0$. Now, due to the fact that $S_\pi(\cdot,\cdot)$ is continuous and e, ΔK are both bounded, Weierstrass theorem [72, Th. 2.13] can be used, meaning that some such

$$\pi^* \triangleq \arg\max_{\pi\in\Pi} S_\pi(e, \Delta K), \qquad (e, \Delta K) \in \mathbb{R}^n \times \mathbb{R}^{m\times(n+m+N)}, \tag{3.42}$$

both exists and is finite. Henceforth, for simplicity of notation, let $S_{\pi^*(e,\Delta K)}$ be represented by S_{π^*}.

Finally, the adaptive law can be synthesized as follows

$$\begin{aligned}\dot{\hat{K}}^{\mathrm{T}}(t) = &-\Gamma[\pi(t)e^{\mathrm{T}}(t)(P - V(e(t),\Delta K(t))) \\ &\cdot h_e(e^{\mathrm{T}}(t)Me(t),\Delta K(t)\Gamma^{-1}\Delta K^{\mathrm{T}}(t))M)B \\ &+\sigma||e^{\mathrm{T}}(t)PB||^p\Delta K^{\mathrm{T}}(t)][\mathbf{1}_m - V(e(t),\Delta K) \\ &\cdot h_X(e^{\mathrm{T}}(t)Me(t),\Delta K(t)\Gamma^{-1}\Delta K^{\mathrm{T}}(t))]^{-1}, \qquad \hat{K}(t_0) = \hat{K}_0, \qquad t \geq t_0,\end{aligned} \tag{3.43}$$

where if $S_{\pi^*} \subset \mathring{\mathcal{C}}$ with π^* as defined previously, then the developed adaptive law is such that $(e,\Delta K) \in \mathring{\mathcal{C}}$, $t \geq t_0$. This control law is able to drive the trajectory of the dynamical system to the desired trajectory within some user-defined constraints, as is proved in [73].

3.4. Application of the MRAC Algorithm to the Tiltrotor

Now, using the previously developed equations of motion and MRAC control algorithm, a control strategy will be applied to the tiltrotor vehicle. Starting with the user-defined continuously differentiable reference trajectory, pitch angle, and roll angle, given by $r^{\mathbb{I}}_{\mathrm{ref}} : [t_0,\infty) \to \mathbb{R}^3$, $\theta_{\mathrm{ref}} : [t_0,\infty) \to (-\frac{\pi}{2},\frac{\pi}{2})$, and $\psi_{\mathrm{ref}} : [t_0,\infty) \to [0,2\pi)$, the virtual translational control input $v_{\mathrm{tran}}(t) : [t_0,\infty) \to \mathbb{R}^3$ and reference

roll angle $\phi_{\text{ref}} : [t_0, \infty) \to [0, 2\pi)$ can be defined as

$$v_{\text{tran}}(t) \triangleq R(q_{\text{ref}})[u_5(t), 0, u_1(t)]^{\text{T}}, \tag{3.44}$$

$$\phi_{\text{ref}} \triangleq -\tan^{-1} \frac{\tilde{v}_{\text{tran},2}(t)}{\tilde{v}_{\text{tran},}(t)}, \qquad t \geq t_0. \tag{3.45}$$

In the above equation, $R(\cdot)$ is as previously defined in formulating the tiltrotor dyanmics, $q_{\text{ref}}(\cdot)$ is the vector of generalized reference coordinates, and $\tan^{-1}$ is the signed inverse tangent function. Furthermore,

$$\begin{bmatrix} \tilde{v}_{\text{tran},1}(t) \\ \tilde{v}_{\text{tran},2}(t) \\ \tilde{v}_{\text{tran},3}(t) \end{bmatrix} = \begin{bmatrix} \cos\theta_{\text{ref}} & 0 & -\sin\theta_{\text{ref}} \\ 0 & 1 & 0 \\ \sin\theta_{\text{ref}} & 0 & \cos\theta_{\text{ref}} \end{bmatrix} \begin{bmatrix} \cos\psi_{\text{ref}} & \sin\psi_{\text{ref}} & 0 \\ -\sin\psi_{\text{ref}} & \cos\psi_{\text{ref}} & 0 \\ 0 & 0 & 1 \end{bmatrix} v_{\text{tran}}(t), \qquad t \geq t_0, \tag{3.46}$$

completing Equations (3.44) and (3.45), where, using Equations (3.44) and (3.11), the equations of motion can be rewritten as

$$\begin{aligned} \mathcal{M}(t, q(t)) \begin{bmatrix} \mathbf{1}_3 & 0 \\ 0 & \Gamma^{-1}(q(t)) \end{bmatrix} \ddot{q}(t) = \begin{bmatrix} f_{\text{tran}}(t, q(t), \dot{q}(t)) \\ f_{rot}(t, q(t), \dot{q}(t)) \end{bmatrix} + [\hat{G}(q(t)) - \hat{G}(q_{\text{ref}}(t))]u(t) \\ + \mathcal{M}(t, q(t)) \begin{bmatrix} 0_{3\times 1} \\ \Gamma^{-1}(q(t))\dot{\Gamma}(q(t))\omega(q(t), \dot{q}(t)) \end{bmatrix} + v(t), \\ \begin{bmatrix} q^{\text{T}}(t_0) \\ \dot{q}^{\text{T}}(t_0) \end{bmatrix} = \begin{bmatrix} q_0^{\text{T}} \\ \dot{q}_0^{\text{T}} \end{bmatrix}, \qquad t \geq t_0, \end{aligned} \tag{3.47}$$

where the unmatched disturbance is given by $[\hat{G}(q(t))-\hat{G}(q_{\text{ref}}(t))]u(t)$ and the equivalent control input given by $v(t) = [v_{\text{tran}}^{\text{T}}(t), u_2(t), u_3(t), u_4(t)]^{\text{T}}$. The equivalent control input is formulated in such a way that $q(t)$ tracks $q_{\text{tran}}(t)$ within the previously discussed user defined bounds. Finally, $u(t)$ is found using the reference control input and Equations (3.44), $q_{\text{ref}}(t)$ is found from the ideal model, $\phi_{\text{ref}}(t)$ provides the thrust required to achieve the goal state along the $y(\cdot)$ axis, and the reference control input computed in order to allow $q(t)$ to track $q_{\text{ref}}(t)$.

One of the reasons for the use of an adaptive control algorithm is the assumption that the center of mass and inertia matrix of the tiltrotor vehicle is unknown, so to estimate them, two twice continuously differentiable functions are defined. First, $\overline{r}_C : [t_0, \infty) \to \mathbb{R}^3$, an estimate of the center of mass, in this case provided by the Computer Aided Design (CAD) model used in the development of the numerical simulation. Secondly, $\Delta r_C : [t_0, \infty) \to \mathbb{R}^3$, representing the unknown part of the center of mass. This yields the final result

$$r_C(t) = \overline{r}_C(t) + \Delta r_C(t), \qquad t \geq t_0. \tag{3.48}$$

Likewise, an estimate is developed for the inertia matrix, from the symmetric and continuously differentiable functions $\overline{I}_{\text{quad}} : [t_0, \infty) \to \mathbb{R}^{3\times3}$, which is assumed to be known and found from the CAD model of the tiltrotor, $I_{P_i} : [t_0, \infty) \to \mathbb{R}^{3\times3}$, the inertia matrix of the ith propeller, also considered known due to the propeller's mass being known, and $\Delta I : [t_0, \infty) \to \mathbb{R}^{3\times3}$, capturing the unknown portion of the

inertia matrix. This results in

$$I(t) = \overline{I}_{\text{quad}}(t) + \sum_{i=1}^{4} I_{P_i}(t) + \Delta I(t), \qquad t \geq t_0. \tag{3.49}$$

the total estimate of the inertia matrix.

Next, the new equations of motion developed in (3.11) are feedback linearized, starting from

$$e(t) = \begin{bmatrix} q(t) - q_{\text{ref}}(t) \\ \dot{q}(t) - \dot{q}_{\text{ref}}(t) \end{bmatrix}, \qquad t \geq t_0, \tag{3.50}$$

where $q_{\text{ref}}(t)$ denotes the vector of generalized reference coordinates, and $q(t)$ is the vector of independent generalized coordinates, defined in the section on the equations of motion. Now, the new feedback linearized equations of motion are broken up into the known and unknown components, with the known dynamics given by

$$\overline{\mathcal{M}}(t,q) \triangleq \begin{bmatrix} m\mathbf{1}_3 & -mR(q)\overline{r}_C^{\times}(t) \\ m\overline{r}_C^{\times}(t)R^{\mathrm{T}}(q) & \overline{I}(t) \end{bmatrix}, \qquad (t,q) \in [t_0,\infty) \times \mathcal{D}, \tag{3.51}$$

and the unknown dynamics by

$$\Delta\mathcal{M}(t,q) \triangleq \begin{bmatrix} 0_3 & -mR(q))\Delta r_C^{\times}(t) \\ m\Delta r_C^{\times}(t)R^{\mathrm{T}}(q) & \Delta I(t) \end{bmatrix}, \qquad (t,q) \in [t_0,\infty)\times\mathcal{D}, \tag{3.52}$$

resulting in

$$\mathcal{M}(t,q) = \overline{\mathcal{M}}(t,q) + \Delta\mathcal{M}(t,q), \qquad (t,q) \in [t_0,\infty) \times \mathcal{D}, \tag{3.53}$$

where $\overline{\mathcal{M}}(t,q)$ is invertible. Likewise, we define

$$\begin{aligned}\overline{f}_{\text{tran}}(t,q,\dot{q}) \triangleq -mR(q)[&\ddot{\overline{r}} + 2\omega^{\times}(q,\dot{q})\dot{\overline{r}}_C(t) \\ &+ \omega^{\times}(q,\dot{q})\omega^{\times}(q,\dot{q})\overline{r}_C(t)] + [0,0,-mg]^{\mathrm{T}}, \qquad t \geq t_0, \end{aligned} \tag{3.54}$$

$$\begin{aligned}\Delta f_{\text{tran}}(t,q,\dot{q}) \triangleq -mR(q)[&\ddot{\Delta r} + 2\omega^{\times}(q,\dot{q})\dot{\Delta r}_C(t) \\ &+ \omega^{\times}(q,\dot{q})\omega^{\times}(q,\dot{q})\Delta r_C(t)] + [0,0,-mg]^{\mathrm{T}}, \qquad t \geq t_0, \end{aligned} \tag{3.55}$$

such that

$$f_{\text{tran}}(t,q,\dot{q}) = \overline{f}_{\text{tran}}(t,q,\dot{q}) + \Delta f_{\text{tran}}(t,q,\dot{q}), \tag{3.56}$$

as well as

$$\begin{aligned}\overline{f}_{\text{rot}}(t,q,\dot{q}) \triangleq &- \omega(q,\dot{q})\overline{I}(t)\omega(q(t),\dot{q}) - \dot{\overline{I}}(t)\omega(q(t),\dot{q}) \\ &- \omega^{\times}(q,\dot{q})\sum_{i=1}^{4} I_{P_i}\omega_{P_i}(t) + \overline{r}_C^{\times}(t)F_g(q) \\ &- \sum_{i=1}^{4}[I_{P_i}\dot{\omega}_{P_i}(t) + \omega_{P_i}^{\times}(t)I_{P_i}(t)\omega_{P_i}(t)], \qquad t \geq t_0 \end{aligned} \tag{3.57}$$

$$\begin{aligned}\Delta f_{\text{rot}}(t,q,\dot{q}) \triangleq &\ \omega^{\times}(q,\dot{q})\Delta I(t)\omega(q,\dot{q}) - \Delta\dot{I}(t)\omega(q(t),\dot{q}) \\ &+ \Delta r_C^{\times}(t)F_g(q), \qquad t \geq t_0, \end{aligned} \tag{3.58}$$

such that

$$f_{\text{rot}}(t, q, \dot{q}) = \overline{f}_{\text{rot}}(t, q, \dot{q}) + \Delta f_{\text{rot}}(t, q, \dot{q}). \tag{3.59}$$

Finally, we define our feedback term as

$$\begin{aligned}\alpha(t, q, q_{\text{ref}}, v_2) = \overline{M}(t, q) \begin{bmatrix} \mathbf{1}_3 & 0_{3\times 3} \\ 0_{3\times 3} & \Gamma^{-1}(q) \end{bmatrix} \Bigg(\ddot{q}_{\text{ref}} - \begin{bmatrix} 0_{3\times 1} \\ \dot{\Gamma}(q)\omega(q, \dot{q}) \end{bmatrix} \\ - K_P(q - q_{\text{ref}}) - K_D(\dot{q} - \dot{q}_{\text{ref}}) + v_2 \Bigg) - \begin{bmatrix} \overline{f}^{\text{T}}_{\text{tran}}(t, q, \dot{q}) \\ \overline{f}^{\text{T}}_{\text{rot}}(t, q, \dot{q}) \end{bmatrix}, \qquad t \geq t_0, \end{aligned} \tag{3.60}$$

where the gain matrices $(K_P, K_D) \in \mathbb{R}^{6\times 6}$ have the properties $K_P = K_P^{\text{T}} > 0$ and $K_D = K_D^{\text{T}} > 0$, and the virtual control input $v_2 \in \mathbb{R}^6$.

Next, the error dynamics are found, where if $v(t) = \alpha(t, q(t), q_{\text{ref}}(t), v_2)$ then the dynamics are given by

$$\begin{aligned}\dot{e}(t) = \begin{bmatrix} 0_{6\times 6} & \mathbf{1}_6 \\ -K_P & -K_D \end{bmatrix} e(t) + \begin{bmatrix} 0_{6\times 6} \\ \mathbf{1}_6 \end{bmatrix} v_2(t) + \hat{\xi}(t), \\ e(t_0) = \begin{bmatrix} q^{\text{T}}(t_0) \\ \dot{q}^{\text{T}}(t_0) \end{bmatrix} - \begin{bmatrix} q^{\text{T}}_{\text{ref}}(t_0) \\ \dot{q}^{\text{T}}_{\text{ref}}(t_0) \end{bmatrix}, \qquad t \geq t_0, \end{aligned} \tag{3.61}$$

with the unmatched uncertainties given by $\hat{\xi}(t) = [0_{1\times 6}, \hat{\xi}_{\text{dyn}}^{\text{T}}(t)] \in \mathbb{R}^{12}$ where

$$\hat{\xi}_{\text{dyn}}^{\text{T}}(t) = \begin{bmatrix} \mathbf{1}_3 & 0_{3\times 3} \\ 0_{3\times 3} & \Gamma(q(t)) \end{bmatrix} \overline{\mathcal{M}}^{-1}(t,q) \left(\begin{bmatrix} \Delta f_{\text{tran}}(t,q,\dot{q}) \\ \Delta f_{\text{rot}}(t,q,\dot{q}) \end{bmatrix} + [\hat{G}(q(t)) - \hat{G}(q_{\text{ref}}(t))]u(t) - \Delta\mathcal{M}(t,q) \begin{bmatrix} \mathbf{1}_3 & 0_{3\times 3} \\ 0_{3\times 3} & \Gamma^{-1}(q(t)) \end{bmatrix} \ddot{q}(t) \right), \qquad t \geq t_0. \tag{3.62}$$

However, due to the coupling between the translation and rotational dynamics due to the fact $r_C(t) \not\equiv 0$, $t \geq t_0$ and the uncertainties in the center of mass and inertia matrix, a regressor vector $\Theta^{\text{T}}\Phi(q(t), \dot{q}(t))$ is introduced, making the trajectory tracking error dynamics as follows

$$\dot{e}(t) = \begin{bmatrix} 0_{6\times 6} & \mathbf{1}_6 \\ -K_P & -K_d \end{bmatrix} e(t) + \begin{bmatrix} 0_{6\times 6} \\ \mathbf{1}_6 \end{bmatrix} [v_2(t) + \Theta^{\text{T}}\Phi(q(t), \dot{q}(t))] + \hat{\xi}(t),$$

$$e(t_0) = \begin{bmatrix} q^{\text{T}}(t_0) \\ \dot{q}^{\text{T}}(t_0) \end{bmatrix} - \begin{bmatrix} q_{\text{ref}}^{\text{T}}(t_0), \\ \dot{q}_{\text{ref}}^{\text{T}}(t_0) \end{bmatrix}, \qquad t \geq t_0. \tag{3.63}$$

In the above, we create Θ and Φ such that

$$\Theta^{\text{T}} = \begin{bmatrix} \Theta_{\text{tran}} & 0_{3\times 30} \\ 0_{3\times 9} & \Theta_{\text{rot}} \end{bmatrix}, \tag{3.64}$$

$$\Phi(q,\dot{q}) = \begin{bmatrix} \Phi_{\text{tran}}^{\text{T}}(q,\dot{q}) \\ \Phi_{\text{rot}}(q,\dot{q}) \end{bmatrix}, \qquad (q,\dot{q}) \in \mathcal{D} \times \mathbb{R}^6, \tag{3.65}$$

where, to define Θ_{tran}, Θ_{rot}, Φ_{tran}, Φ_{rot}, two new functions must be defined. First, given an $A \in \mathbb{R}^{n\times m}$, $b \in \mathbb{R}^m$, and $n \in \mathbb{N}$, a function $M_W(b,n)$ exists such that

$$M_W(b,n) \triangleq (b^{\mathrm{T}} \otimes \mathbf{1}_n) \in \mathbb{R}^{n\times nm} \tag{3.66}$$

and a function $W_M(A)$ exists such that

$$W_M(A) \triangleq \sum_{i=1}^{n} [e_{i,m} \otimes (A\mathbf{e}_{i,m})] \in \mathbb{R}^{nm} \tag{3.67}$$

where $\mathbf{e}_{i,m}$ represents the ith vector of the standard basis in $\mathbb{R}^m$, and $\otimes$ denotes the Kronecker product [74, Def. 7.1.2]. Now that these two functions are defined, the following are created,

$$\Theta_{\text{tran}} = M_W(\overline{\overline{r}}_C, 3), \tag{3.68}$$

$$\Theta_{\text{rot}} = [M_W(W_M(\overline{\overline{I}}), 3), \overline{\overline{r}}_C^{\times}], \tag{3.69}$$

$$\Phi_{\text{tran}}(q,\dot{q}) = W_M\left(R(q)\omega^{\times}(q,\dot{q})\omega^{\times}(q,\dot{q})\right), \tag{3.70}$$

$$\Phi_{\text{rot}}(q,\dot{q}) = \begin{bmatrix} -W_M^{\mathrm{T}}\left(\omega^{\times}(q,\dot{q})M_W(\omega(q,\dot{q}))\right) \\ F_g^{\mathrm{T}}(q) \end{bmatrix}, \qquad t \geq t_0 \tag{3.71}$$

where $\overline{\overline{r}}_C \in \mathbb{R}^3$ and $\overline{\overline{I}}_C \in \mathbb{R}^{3\times 3}$ are unknown. The result of the above identities is

that

$$\Theta_{\text{tran}}\Phi_{\text{tran}}(q,\dot{q}) = -mR(q)\omega^{\times}(q,\dot{q})\omega^{\times}(q,\dot{q})\overline{\overline{r}}_C, \tag{3.72}$$

$$\Theta_{\text{rot}}\Phi_{\text{rot}}(q,\dot{q}) = -\omega^{\times}(q,\dot{q})\overline{\overline{I}}\omega(q,\dot{q}) + \overline{\overline{r}}_C^{\times}F_g^{\text{T}}(q). \tag{3.73}$$

Now that the adaptive law can be applied to the tiltrotor's equations of motion in order to regulate the trajectory tracking error, the goal is to have both the trajectory tracking and adaptive gain error fit inside some user-defined constraints. To do so, set $\pi(t) = [q^{\text{T}}(t), \dot{q}^{\text{T}}(t), 0_{1\times 6}, \Phi^{\text{T}}(q(t),\dot{q}(t))]^{\text{T}}$, as well as $\Delta K = \hat{K}(t) - K_e$, $t \geq t_0$, where $\hat{K} : [t_0,\infty) \to \mathbb{R}^{6\times 57}$, and $K_e = -[K_P, K_D, 0_{6\times 6}, \Theta_e^{\text{T}}]$, where Θ_e is an estimate of Θ within some arbitrarily small tolerance $\epsilon \geq 0$ such that $||\Theta_e - \Theta|| \leq \epsilon$ and $\Theta_e \in \mathbb{R}^{39\times 6}$. Also let

$$v_2(t) = \Delta K(t)\pi(t), \qquad t \geq t_0, \tag{3.74}$$

then we can see that our trajectory tracking error dynamics from Equation (3.63) follows the same form as our reference error dynamics from Equation (3.36) where

$$A_{\text{ref}} = \begin{bmatrix} 0_{6\times 6} & \mathbf{1}_6 \\ -K_P & -K_D \end{bmatrix}, \qquad B = \begin{bmatrix} 0_{6\times 6} \\ \mathbf{1}_6 \end{bmatrix}, x_0 = \begin{bmatrix} q_0^{\text{T}} \\ \dot{q}_0^{\text{T}} \end{bmatrix}, \tag{3.75}$$

and $n = 12$, $m = 6$, $N = 39$, $K = -[K_P, K_D, 0_{6\times 6}, \Theta^{\text{T}}]$, and $\widetilde{\Delta K}(t) = \Delta K + K_e - K$, $t \geq 0$. This also means that the matching conditions we discussed previously are

proven with

$$A = \begin{bmatrix} 0_{6\times6} & \mathbf{1}_6 \\ 0_{6\times6} & 0_{6\times6} \end{bmatrix}, \qquad B = \begin{bmatrix} 0_{6\times6} \\ \mathbf{1}_6 \end{bmatrix}, \tag{3.76}$$

as well as $K_r = 0_{6\times6}$ and $K_x^{\mathrm{T}} = -[K_P, K_D]$.

From the computation of the virtual control input and actual control input, the propeller thrust and drag forces are computed with the following

$$T_i(t) = k_T \dot{\Omega}_i^2(t), \qquad i = 1, \ldots, 4, \qquad t \geq t_0, \tag{3.77}$$

$$D_i(t) = k_D \dot{\Omega}_i^2(t), \tag{3.78}$$

Where $T_i(t)$ is the thrust force of the ith propeller, $D_i(t)$ is the drag force on the ith propeller, $k_T > 0$ is the propeller thrust coefficient, $k_D > 0$ is the propeller drag coefficient, and $\dot{\Omega}_i^2(t)$ is the angular velocity of the ith propeller [75]. Also, adjacent propellers spin in opposite directions, a common configuration for quadcopters, and are assumed to be equidistant from the reference point $A(\cdot)$, such that

$$r_{\mathrm{prop},1} = \begin{bmatrix} L_x \\ L_y \\ L_z \end{bmatrix}, \qquad r_{\mathrm{prop},2} = \begin{bmatrix} -L_x \\ L_y \\ L_z \end{bmatrix}, \qquad r_{\mathrm{prop},3} = \begin{bmatrix} -L_x \\ -L_y \\ L_z \end{bmatrix}, \qquad r_{\mathrm{prop},4} = \begin{bmatrix} L_x \\ -L_y \\ L_z \end{bmatrix}, \tag{3.79}$$

Where $L_{(\cdot)} > 0$. Due to the tilting of the propellers about the vehicle y-axis, the thrust components are broken up into $T_{i,x} \triangleq T_i(t)\sin\alpha_i(t)$ and $T_{i,z} \triangleq T_i(t)\cos\alpha_i(t)$ components, where they represent the thrust force component of the ith propeller along the body frame x and z axis, respectively. This gives us the vector of thrust

forces $T \triangleq [T_{1,z}, T_{1,x}, T_{2,z}, T_{2,x}, T_{3,z}, T_{3,x}, T_{4,z}, T_{4,x},]^{\mathrm{T}}$ in the form $u(t) = MT(t), t \geq t_0$ such that

$$\begin{bmatrix} u_5(t) \\ u_1(t) \\ u_2(t) \\ u_3(t) \\ u_4(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ L_y & k_D & L_y & -k_D & -L_y & k_D & -L_y & -k_D \\ -L_x & L_z & L_x & L_z & L_x & L_z & -L_x & L_z \\ k_D & -L_y & -k_D & -L_y & k_D & L_y & -k_D & L_y \end{bmatrix} T(t), \qquad t \geq t_0, \tag{3.80}$$

where, due to L_x, L_y, and k_D all being positive values, the matrix M is full rank and as such is Moore-Penrose invertible. This means that the Moore-Penrose inverse M^+ is defined as in [74, Pro. 6.1.5]

$$M^+ \triangleq [M^{\mathrm{T}} M]^{-1} M^{\mathrm{T}} \tag{3.81}$$

Meaning our computed vector of thrust forces $T^*(t)$ is calculated as

$$T^*(t) = M^+ u(t), \qquad t \geq t_0, \tag{3.82}$$

and the propeller tilt angles from

$$\alpha_i(t) = \tan^{-1} \frac{T^*_{i,x}(t)}{T^*_{i,z}(t)}, \qquad i = 1, \ldots, 4, \qquad t \geq t_0, \tag{3.83}$$

where $\tan^{-1}$ is the signed inverse tangent function.

3.5. Numerical CAD Simulation

To test the theoretical result derived above, a tiltrotor with mass and inertial properties identical to one used in research was modeled in Solidworks CAD, as shown in Figure 3.2.

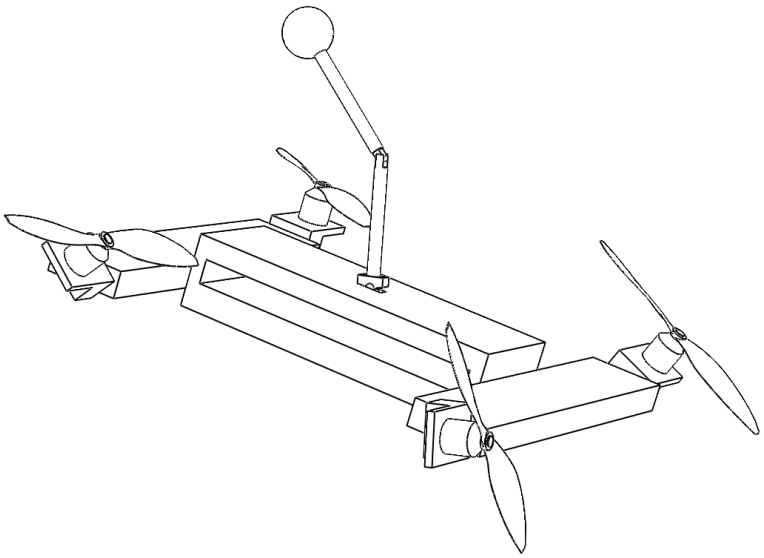

Figure 3.2: CAD model for an H-configuration tiltrotor quadctoper.

From this step, the CAD model shown in Figure 3.2 was then imported into a Matlab Simulink environment, where the tiltrotor is made up of a frame, four actuated rotating arms attached to the motors, four rotating propellers, and an actuated inverted pendulum. The inverted double pendulum was considered as a payload disturbance, where the first pendulum link attached to the tiltrotor frame oscillates at 0.5 Hz with an amplitude of 10 degrees and the second pendulum link oscillates at 1.5 Hz with an amplitude of 20 degrees. The payload's mass is 0.1 kg, and each pendulum section has a length of 0.154 m. Furthermore, the mass of the

tiltrotor frame and propellers is 1.667 kg. Finally, the tiltrotor principle moments of inertia are 0.0317, 0.0643, and 0.0319 kg$(\cdot)$m^2 along the $x(\cdot)$, $y(\cdot)$, and $z(\cdot)$ axes, respectively.

For the simulated trajectory, the reference pitch and yaw angles were set to $\theta_{\text{ref}}(t) = \psi_{\text{ref}}(t) = 0, t \geq t_0$, and the reference position given by $r^{\mathbb{I}}_{\text{ref}}(t) = [2\cos(0.35(t-10)), 2\sin(0.35(t-10)), 2+(t-4)/30]^{\text{T}}$. In addition, the frequency and amplitude of oscillation of the payload was considered as unknown to the controller. Additional disturbances were added in the form of a wind effect of 10 m/s along the X axis of the inertial frame from time $t \geq 10$, and a simulated failure of the inertial measurement system during $t \in [15, 20]$ where the measured roll, pitch, and yaw angles were all set to zero. One final disturbance was an underestimation of the vehicle mass and inertia properties by 10%. The purpose of these disturbances were to show the robustness of the proposed MRAC control law. As can be seen in Figure 3.2, the tiltrotor closely tracks the reference trajectory throughout the entire flight period, despite the introduction of disturbances to the flight model. The root mean square of the trajectory following error is $18{\cdot}10^{-3}$ m.

Of note in the control inputs illustrated in Figure 3.3, the effect of the double pendulum perturbation on the pitching moment $u_3(\cdot)$ can be followed as the control input shows similar profile to the position of the payload on the double pendulum throughout the flight time. Furthermore, the horizontal force $u_5(\cdot)$ and the pitching moment $u_3(\cdot)$ show an increase due to the presence of aerodynamic drag. One final note is that there is as rapid fluctuation in control inputs right after $t = 20$ as the controller compensates for the error induced by the failure of the inertial

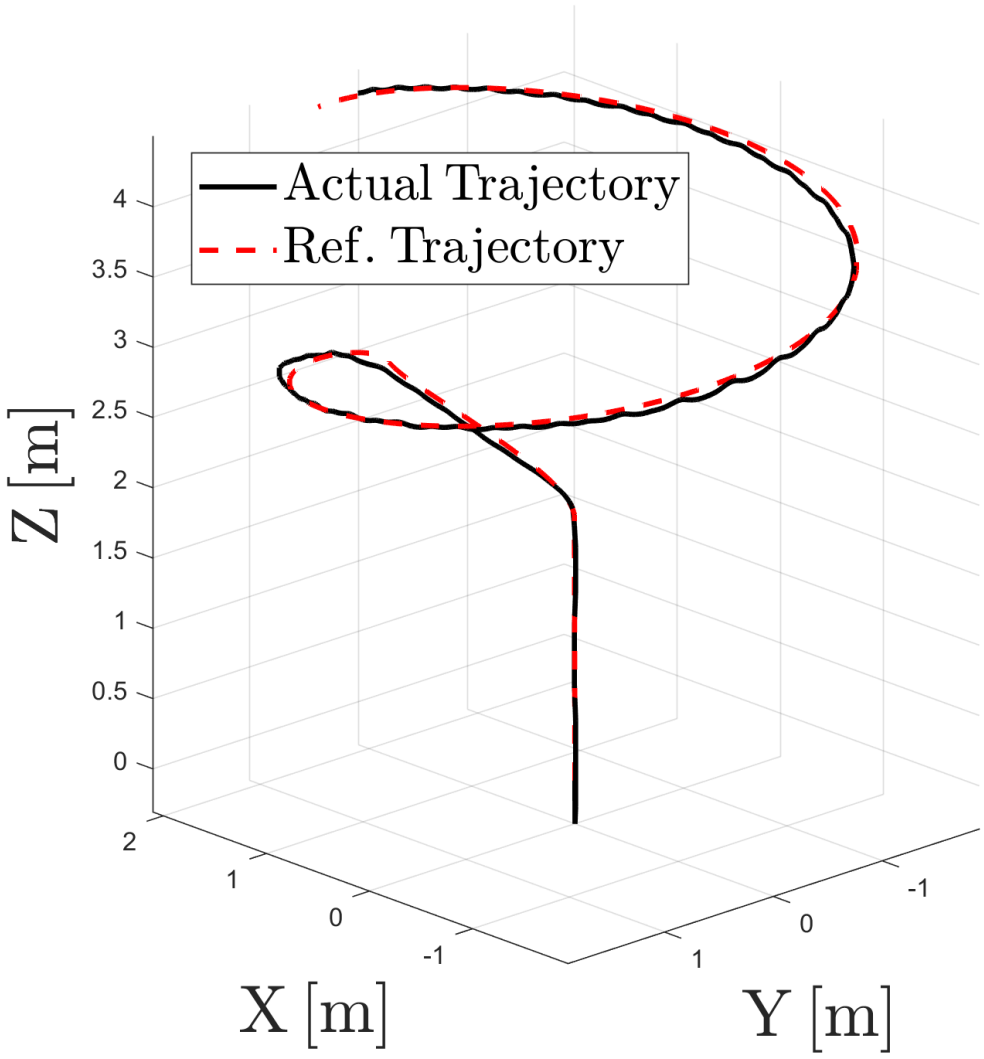

Figure 3.3: Trajectory of a tiltrotor following an ascending spiral reference trajectory.

measurement unit.

Finally, showing that the trajectory tracking and adaptive gain errors meets the user-defined constraints is shown in Figure 3.4. As can be seen, $h(\cdot, \cdot)$ meets the user-defined constraints on the trajectory tracking and adaptive gain error, as defined in Equation (3.35)

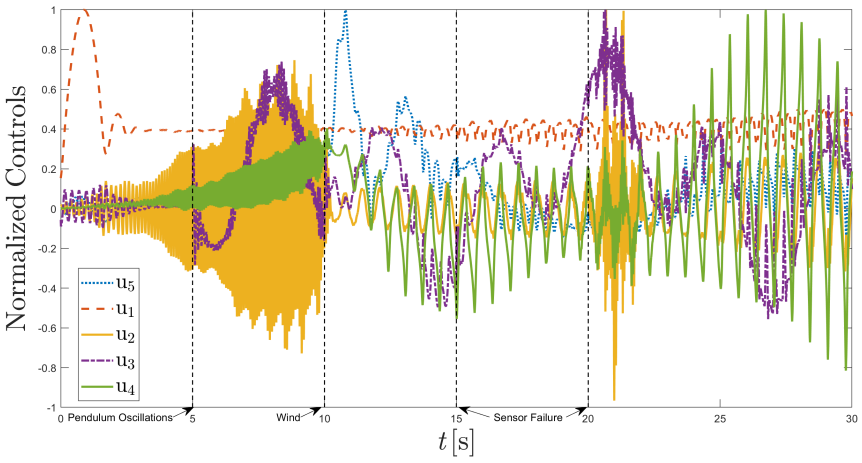

Figure 3.4: Control inputs

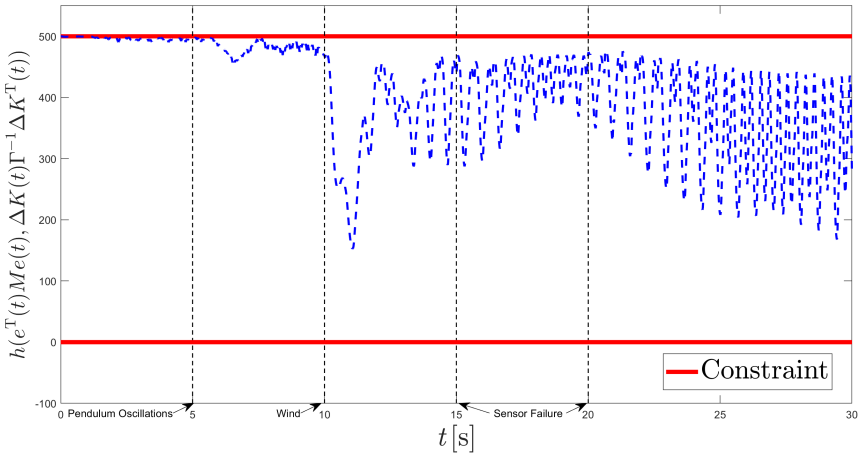

Figure 3.5: Constraints

4. Control of a Two-Link Robotic Manipulator on a Cylindrical Hinge

4.1. Introduction

The goal of this chapter is to outline trajectory generation and control algorithms for a two-link robotic arm, such as the `AX-12A Smart Robotic Arm`, that will be installed on the tiltrotor quadcopter presented in Chapter 3. First, the techniques used to define the trajectory of the robotic arm end-effector will be discussed succinctly, and a PID control tehnique will be illustrated. Finally, the effectiveness of these guidance and control techniques wil be demonstrated by means of a numerical simulation.

4.2. Properties of the Robotic Manipulator

In this thesis, we consider the robotic manipulator similar to the `AX-12A Smart Robotic Arm` by `CrustCrawler Robotics`, whose Solidworks model is shown in Figure 4.1. This robotic arm is modelled as a two-link robotic arm mounted on a cylindrical hinge. Specifically, let ① denote the rotating base, ② denote the first joint and servo, ③ denote the second joint and servo, and ④ denote the end-effector. The first arm link joins ② and ③, the second arm link joins ③ and ④, and the segment joining ① and ② forms a cylindrical hinge. In the following, we will refer to this modelling choice as a *two-link robotic manipulator on a cylindrical joint*.

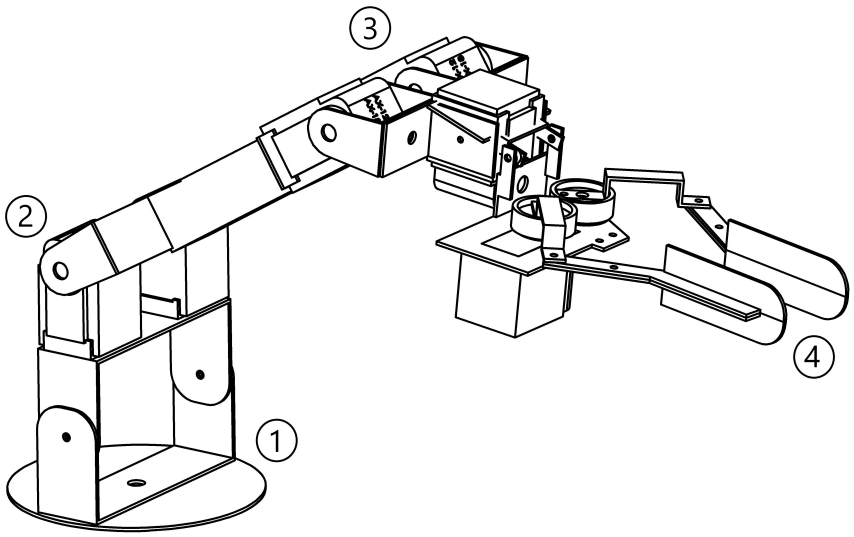

Figure 4.1: CAD model of the `AX-12A Smart Robotic Arm`

4.3. Inverse Kinematics of a Two-Link Robotic Manipulator on a Cylindrical Joint

In order to control a robotic manipulator, two classes of control techniques are generally used. Specifically, *position control techniques* deduce a path for the robotic manipulator to reach the desired position and the resulting control inputs are computed by exploiting the inverse-kinematics [28], [76]. These techniques are usually effective whenever the manipulator's inertial properties, kinematics, and dynamics are well known. Alternatively, *torque control techniques* deduce the required control inputs as functions of the trajectory tracking error [44], [46]. These techniques are usually preferred whenever the manipulator's models are affected by uncertainties or external disturbances may occur. In general, the computational cost of torque-control techniques is higher.

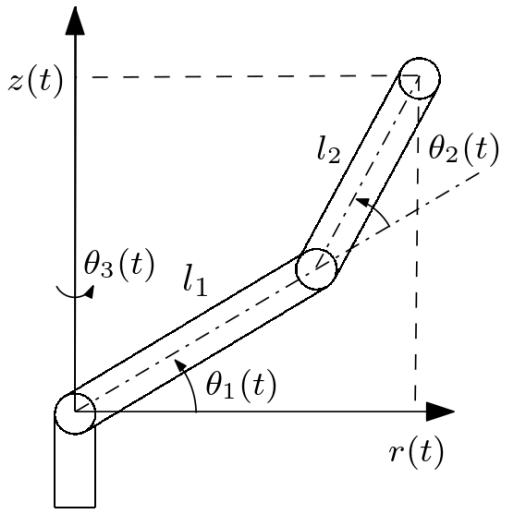

Figure 4.2: Schematic representatino of a two-Link manipulator on a cylindrical joint

In this work, we exploit a position control technique. To this goal, let the desired position of the end-effector be given by $[x(t), y(t), z(t)]^{\mathrm{T}}$, $t \geq t_0$. Moreoever, let $\theta_3(t)$, $t \geq t_0$ denote the angular position of the cylindrical hinge, $\theta_1(t)$, the angular position of the first arm link measured about the axis orthogonal to the cylindrical joint, and $\theta_2(t)$ the angular position of the second arm link. In this case, the position of the end-effector in the plane rotating with the cylindrical joint is given by [77]

$$\begin{bmatrix} r(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} l_1 \cos\theta_1(t) + l_2 \cos(\theta_1(t) + \theta_2(t)) \\ l_1 \sin\theta_1(t) + l_2 \sin(\theta_1(t) + \theta_2(t)) \end{bmatrix}, \qquad t \geq t_0, \tag{4.1}$$

where $l_1 > 0$ denotes the length of the first arm link, $l_2 > 0$ denotes the length of the second arm link, and $l_1 \neq l_2$. Therefore, the position of the end-effector is given

by

$$\begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} r(t)\cos\theta_3(t) \\ r(t)\sin\theta_3(t) \\ \tan^{-1}\left(\frac{x(t)}{y(t)}\right) \end{bmatrix}, \qquad t \geq t_0. \tag{4.2}$$

Given $[x(t), y(t), z(t)]^{\mathrm{T}}$, $t \geq t_0$, it holds that

$$\theta_1(t) = \tan^{-1}\left(\frac{z(t)}{r(t)}\right) - \tan^{-1}\left(\frac{l_1 \sin\theta_2(t)}{l_1 + l_2\sin\theta_2(t)}\right), \tag{4.3}$$

$$\theta_2(t) = \frac{\pm\sqrt{1 - D(t)^2}}{D(t)}, \tag{4.4}$$

$$\theta_3(t) = \tan^{-1}\left(\frac{y(t)}{x(t)}\right), \tag{4.5}$$

where,

$$D(t) = \frac{r^2(t) + z^2(t) - (l_1^2 + l_2^2)}{2l_1 l_2}, \tag{4.6}$$

and $\tan^{-1}$ denotes the signed inverse tangent function [28, Ch. 4]. Equations (4.3) and (4.4) show that there exist two pairs $(\theta_1(\cdot), \theta_2(\cdot))$ that allow the end-effector to reach the same position. Furthermore, note that (4.3) is well-defined since $r(t) > 0$, $t \geq t_0$, and $l_1 \neq l_2$.

4.4. Trajectory Generation for a Two-Link Robotic Manipulator

Once a relation between end-effector position and configuration angles has been established, the next step is to generate continuously differentiable trajectories for each manipulator joint to move the end-effector from some known initial position to some desired end position. Specifically, assume that $\theta_{i,0}$ and $\theta_{i,f}$ are known. The trajectory generation problem is that of finding $\theta_{i,\mathrm{ref}}(t)$, $t \in [t_0, t_f]$ so that

$\theta_{i,\text{ref}}(t_0) = \theta_{i,0}$, $\theta_{i,\text{ref}}(t_f) = \theta_{i,f}$, and additional constraints are satisfied. Examples of constraints to verify are the joint's maximum angular velocity or acceleration. In this thesis, we use a quintic polynomial trajectory planning technique, which is common in the robotic manipulator literature [28], [41–43]. For this method, it is assumed that

$$\theta_{i,\text{ref}}(t_0) = \theta_{i,0}, \qquad i = 1, 2, 3, \tag{4.7}$$

$$\dot{\theta}_{i,\text{ref}}(t_0) = v_{i,0}, \tag{4.8}$$

$$\ddot{\theta}_{i,\text{ref}}(t_0) = \alpha_{i,0}, \tag{4.9}$$

$$\theta_{i,\text{ref}}(t_f) = \theta_{i,f}, \tag{4.10}$$

$$\dot{\theta}_{i,ref}(t_f) = v_{i,f}, \tag{4.11}$$

$$\ddot{\theta}_{i,\text{ref}}(t_f) = \alpha_{i,f}, \tag{4.12}$$

are known, and [28]

$$\theta_{i,\text{ref}}(t) = a_{i,0} + a_{i,1}t + a_{i,2}t^2 + a_{i,3}t^3 + a_{i,4}t^4 + a_{i,5}t^5, \qquad t \in [t_0, t_f], \qquad i = 1, 2, 3, \tag{4.13}$$

where $a_{i,j} \in \mathbb{R},\ i = 1, 2, 3,\ j = 1, \ldots, 6$. The polynomial coefficients $a_{i,j},\ i = 1, 2, 3,$ $j = 1, \ldots, 6$, are computed as follows. First, it follows from Equation (4.12) that

$$\dot{\theta}_{i,\text{ref}}(t) = 0 + a_{i,1} + 2a_{i,2}t + 3a_{i,3}t^2 + 4a_{i,4}t^3 + 5a_{i,5}t^4,\ \ t \in [t_0, t_f],\ \ i = 1, 2, 3, \tag{4.14}$$

$$\ddot{\theta}_{i,\text{ref}}(t) = 0 + 0 + 2a_{i,2} + 6a_{i,3}t + 12a_{i,4}t^2 + 20a_{i,5}t^3, \tag{4.15}$$

and it follows from Equations (4.7) - (4.12) that

$$\begin{bmatrix} \theta_{i,0} \\ v_{i,0} \\ \alpha_{i,0} \\ \theta_{i,f} \\ v_{i,f} \\ \alpha_{i,f} \end{bmatrix} = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_{i,0} \\ a_{i,1} \\ a_{i,2} \\ a_{i,3} \\ a_{i,4} \\ a_{i,5} \end{bmatrix}. \tag{4.16}$$

The determinant of the matrix in Equation (4.15) is equal to $-4(t_0 - t_f)^9$, which is non-zero whenever $t_0 \neq t_f$. Thus, the matrix in (4.16) is invertible whenever $t_0 \neq t_f$.

Therefore, a_i, $i = 1, \ldots, 6$, can be captured as

$$\begin{bmatrix} a_{i,0} \\ a_{i,1} \\ a_{i,2} \\ a_{i,3} \\ a_{i,4} \\ a_{i,5} \end{bmatrix} = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix}^{-1} \begin{bmatrix} \theta_{i,0} \\ v_{i,0} \\ \alpha_{i,0} \\ \theta_{i,f} \\ v_{i,f} \\ \alpha_{i,f} \end{bmatrix}. \tag{4.17}$$

4.5. Control of a Two-Link Robotic Manipulator on a Cylindrical Joint

In this section, the formulation of the control structure for one joint servo will be discussed. The same process can be applied to all joint servos. First, the inductance and resistance of the servo are used to develop the electrical plant transfer function [28, Ch. 10]

$$T_{1,i}(s_1) = \frac{1}{L_i s_1 + R_i}, \qquad s_1 \in \mathbb{C}, \qquad i = 1, 2, 3, \tag{4.18}$$

where $L_i > 0$ denotes the inductance and $R_i > 0$ denotes the resistance of the servo. Second, the mechanical transfer function of the servo is found as [28, Ch. 10]

$$T_{2,i}(s_2) = \frac{1}{J_i s_2 + B_i}, \qquad s_2 \in \mathbb{C}, \qquad i = 1, 2, 3, \tag{4.19}$$

where $J_i > 0$ denotes the servo inertia and $B_i > 0$ denotes the servo damping. Finally, the torque constant $K_{t,i} > 0$, $i = 1, 2, 3$, and back electro-magnetic frequency

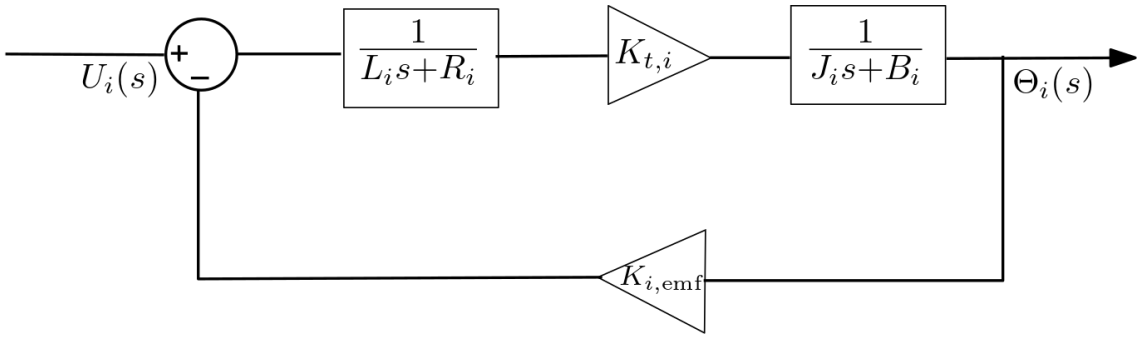

Figure 4.3: Block diagram of a joint servo system

$K_{\mathrm{emf},i} > 0$ are found, and it follows from (4.18) and (4.19) that the full servo model is given by [28, Ch.10, Eqn 10.13]

$$\frac{\Theta_i(s)}{U_i(s)} = \frac{K_{\mathrm{emf},i}}{[(J_i s + B_i)(L_i s + R) + K_{t,i} K_{\mathrm{emf},i}]}, \qquad s \in \mathbb{C}, \qquad i = 1, 2, 3, \qquad (4.20)$$

here $U(\cdot)$ denotes the Laplace transform of the control input and the $\Theta_i(\cdot)$ the Laplace transform of $\theta_i(\cdot)$. This transfer function is shown in Figure 4.3.

Now, let $\Theta_{i,\mathrm{ref}}(\cdot)$ denote the Laplace transform of $\theta_{i,\mathrm{ref}}(\cdot)$. To control the servo, a PID compensator is introduced, yielding the final servo block diagram shown in Figure 4.4. This process is then repeated for every servo joint in the manipulator system.

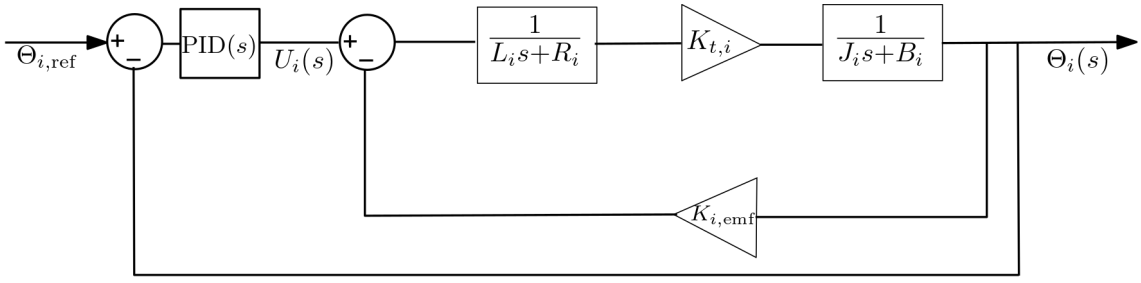

Figure 4.4: Block diagram of servo with PID controller

4.6. Results

The robotic manipulator shown in Figure 4.1 was first modelled in Solidworks, then the CAD model imported into the Simulink Environment in Matlab. Next, the inverse kinematics equations, trajectory generation, and PID controller described previously are implemented in the Simulink environment to ensure the efficacy of the trajectory generation and control law. Additionally, an independent Gaussian white noise feedback disturbance was added to each of the joint servos to simulate electrical disturbances. To test the model, a trajectory involving first a rotation, then a motion in the 2D-plane, then a combined motion was used to demonstrate the manipulator's range. The results for each manipulator joint are shown in Figures 4.5 – 4.7

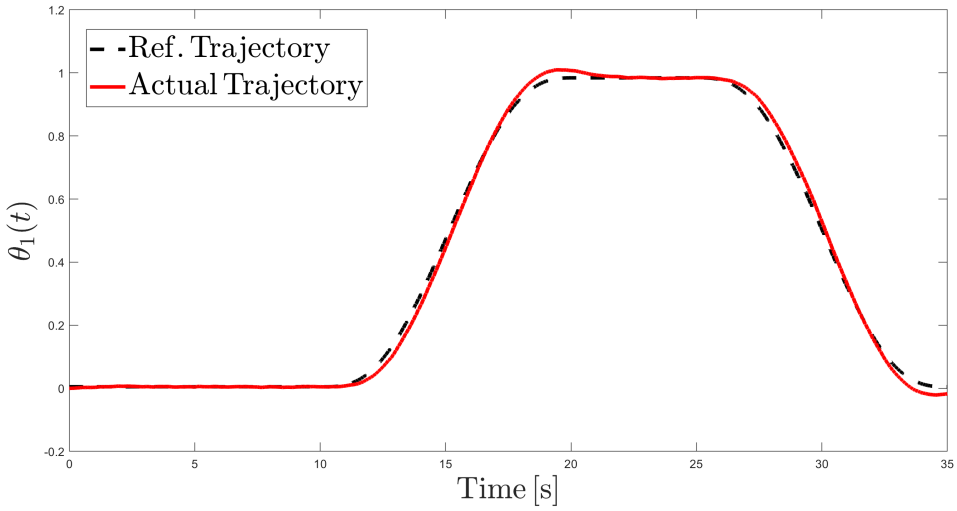

Figure 4.5: Closed-loop trajectory of the first joint

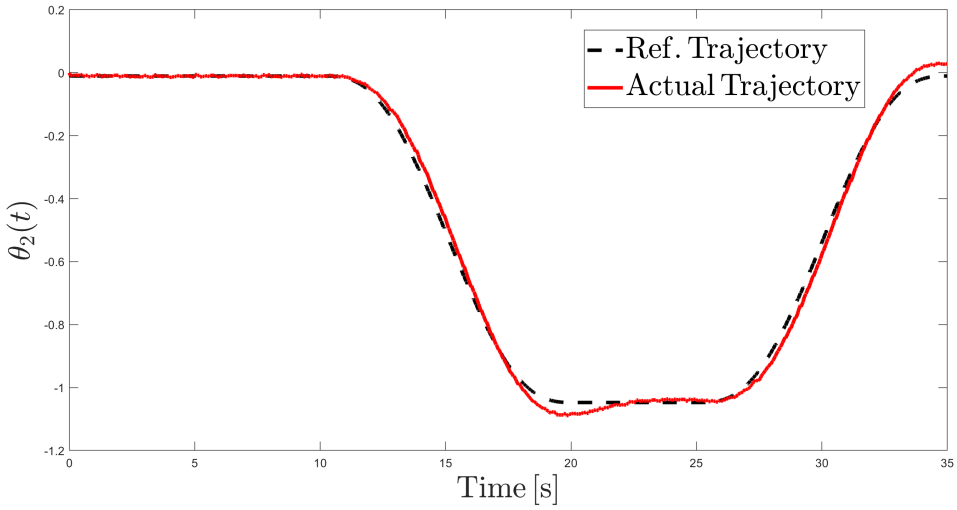

Figure 4.6: Closed-loop trajectory of the second joint

As can be seen, the actual trajectory closely follows the desired trajectory, where the root mean square of the end-effector trajectory following error is $13 \cdot 10^{-3}$ m. Future work direction consists of using robust adaptive control laws to control the robotic arm.

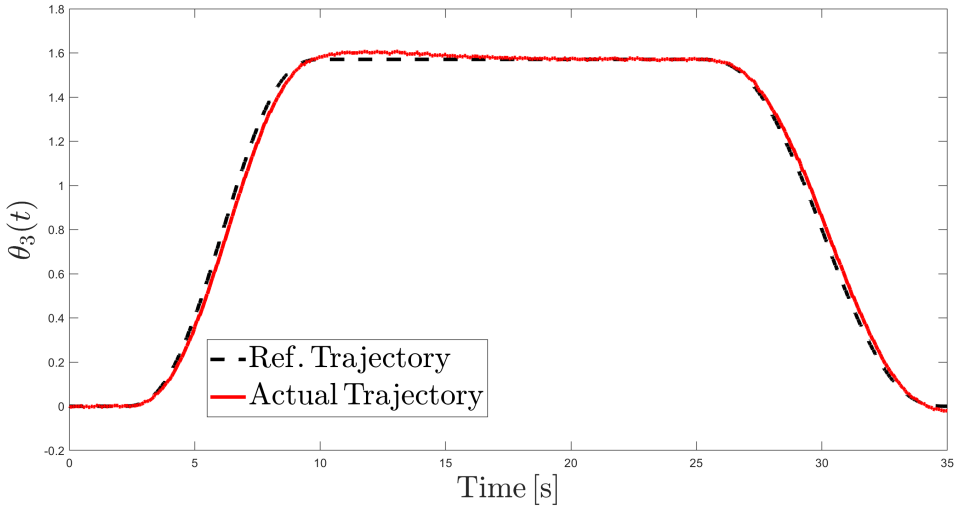

Figure 4.7: Closed-loop trajectory of the third joint

5. Control of an Aerial Manipulator

5.1. Introduction

In this chapter, the numerical results obtained by integrating a tiltrotor quadcopter and a two-link robotic manipulator are presented. These results have been achieved employing an original Simulink simulator, whose features are presented in detail.

5.2. Modelling the Aerial Manipulator

The first step in the creation of a virtual reality simulator to test the ability of the robust MRAC law presented in Chapter 3 was to create the combined CAD model of the aerial manipulator system. While both the tiltrotor and manipulator had been separately modelled, they needed to be combined into a single CAD model to be imported into Simulink. As previously discussed the tiltrotor model was formulated based on the actual measured dimensions of a tiltrotor vehicle used for flight experiments, and the manipulator CAD model based on the `AX-12A Smart Robotic Arm` used in robot vision and control experiments.

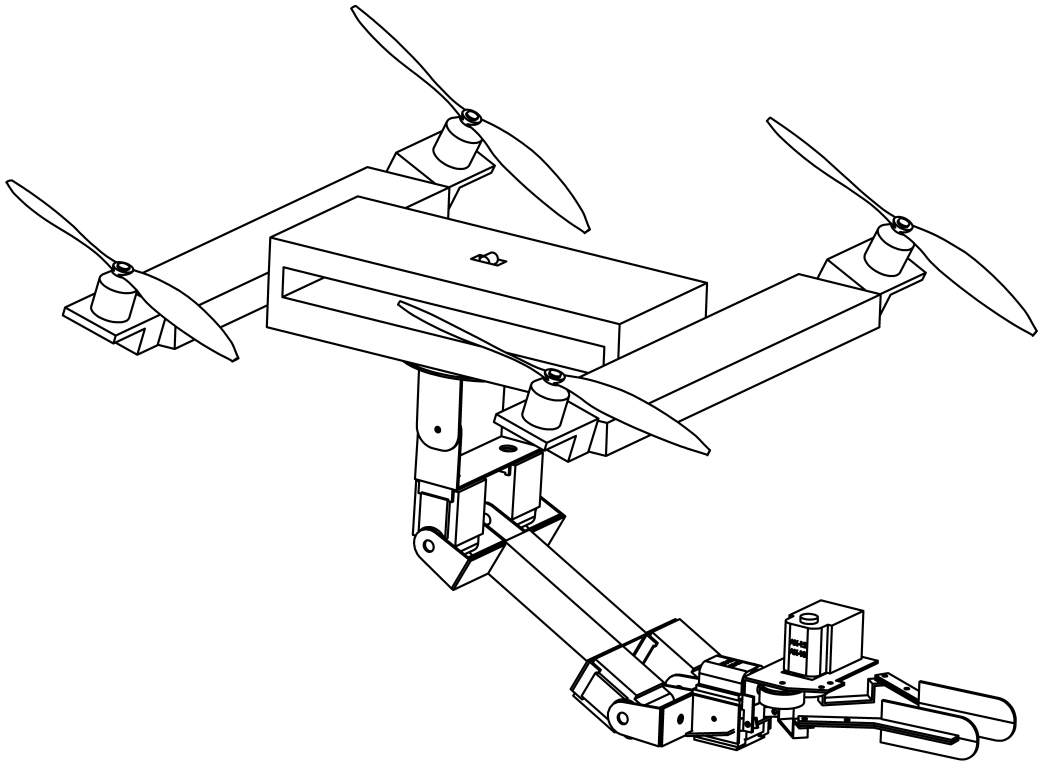

Figure 5.1: CAD model of the aerial manipulator system

This resulted in the CAD model illustrated in Figure 5.1. This CAD model was then imported into the Simulink environment (see Appendix A). In this model, the robotic manipulator is considered by the quadcopter's control law as a payload disturbance of unknown mass, position, and inertial properties. Additionally, a 10 m/s wind disturbance is introduced, in order to further demonstrate the robustness of the MRAC control law. The results of this simulation will be presented after a detailed description of the simulator.

5.3. Setup and Initialization

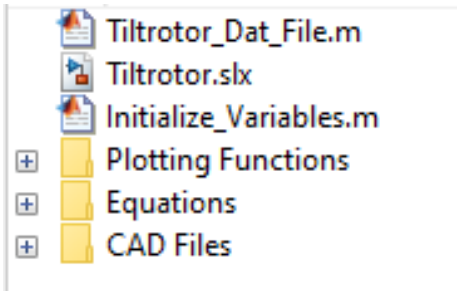

Figure 5.2: Necessary files for simulation

The first thing to do to operate the simulator is to ensure that all the necessary files are in place. In the main Matlab screen, the file list on the left side should look as in Figure 5.2, where `Tiltrotor_Dat_File.m` contains the necessary mass, position, and inertial properties for the aerial manipulator based on the Solidworks model,

`Tiltrotor.slx` is the simulation itself, `Initialize_Variables.m` provides the Matlab script to implement the user defined tuning parameters into the model, the scripts to produce the plots resulting from the simulation are stored in the folder `Plotting Functions`, the image functions used in the simulation masks stored in the `Equations` folder, and the CAD files responsible for providing visuals for the simulation stored in the `CAD Files` folder. Once these files and folders are verified to be in place, the `Tiltrotor.slx` file can be opened to start interacting with the simulator. Upon opening the simulation, the user will see the interface shown in Figure 5.3.

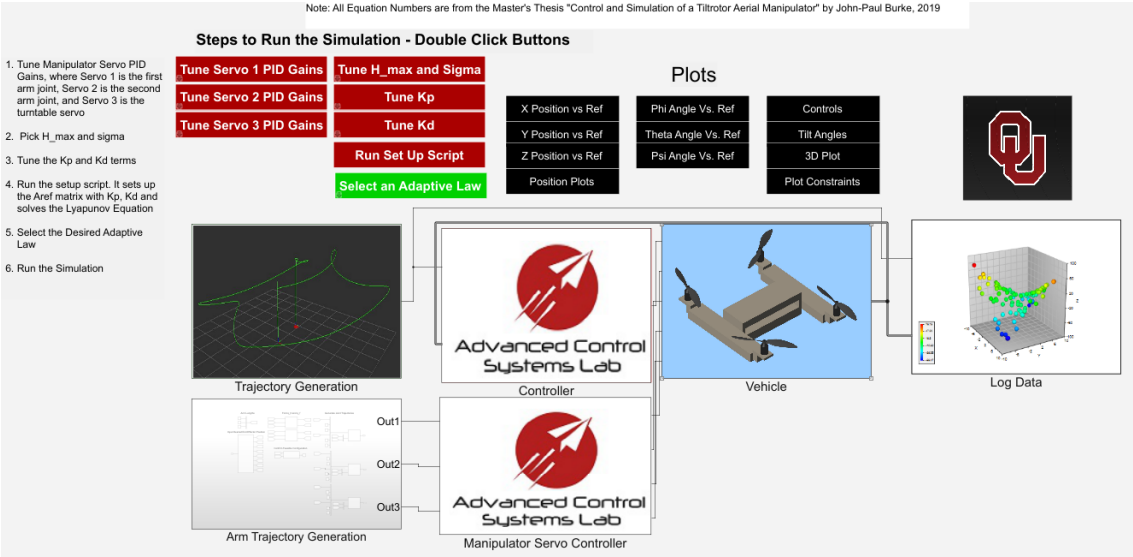

Figure 5.3: Simulator Graphical User Interface (GUI)

This is the top level of the aerial manipulator simulation, providing its general structure. First, by selecting from the menu on top of the Simulink workspace `View > Model Explorer > Model Workspace`, where the user can verify that the physical properties of the aerial manipulator are properly implemented; see Figure 5.4.

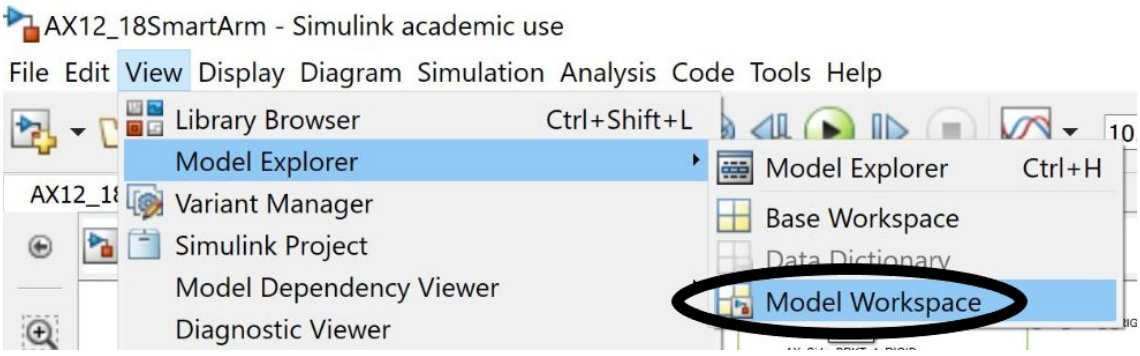

Figure 5.4: Opening the model workspace

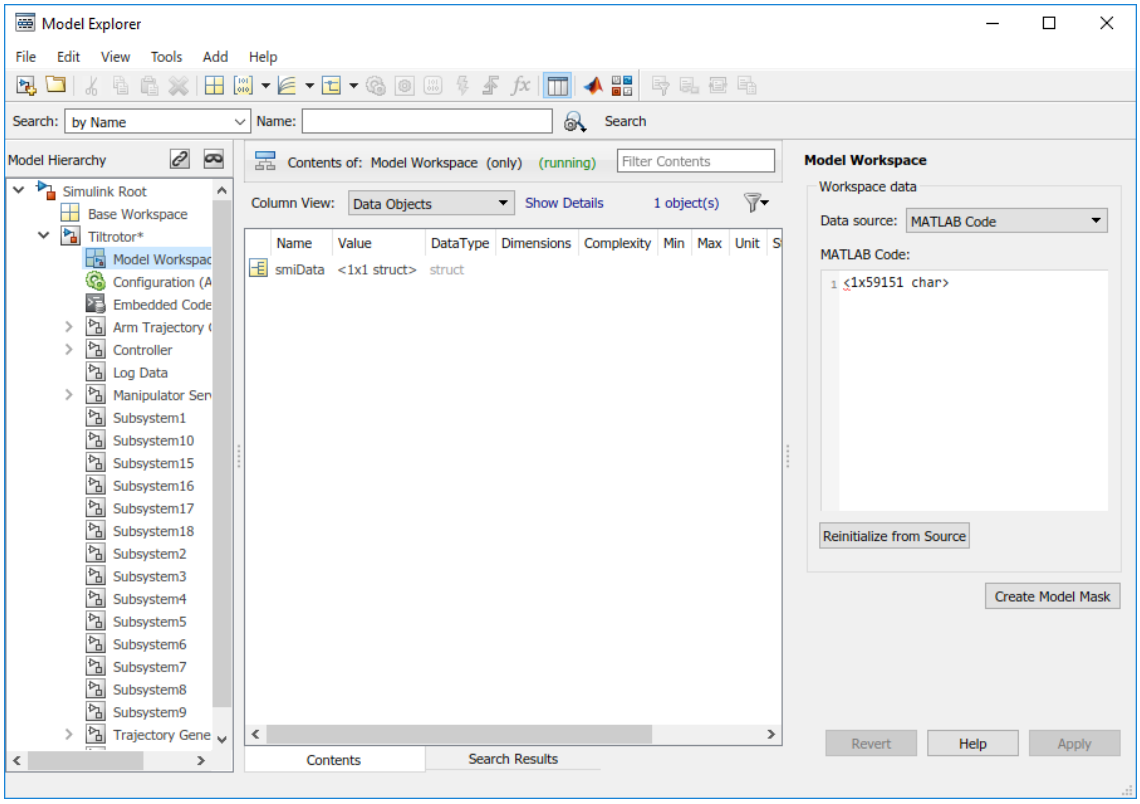

Figure 5.5: Model workspace

The `Model Workspace` window should look as in Figure 5.5. If it does not match the figure, or a "Model Workspace not found" error displays when the simulation is run, then the user should reinitialize the model workspace as follows. First, navigate back to the `Model Workspace` window, select the `Data Source` option on the upper right side of the display, select `Matlab File` from the drop down menu, then navigate through the file list and select `Tiltrotor_Dat_File.m`.

Successively, select `Reinitialize from Source`, then close the window. This will ensure the import of the CAD model properties into the simulink environment. At the top left of the simulation, as seen in Figure 5.3, are a group of eight blocks in two columns, three on the left and five on the right; double-clicking any of these

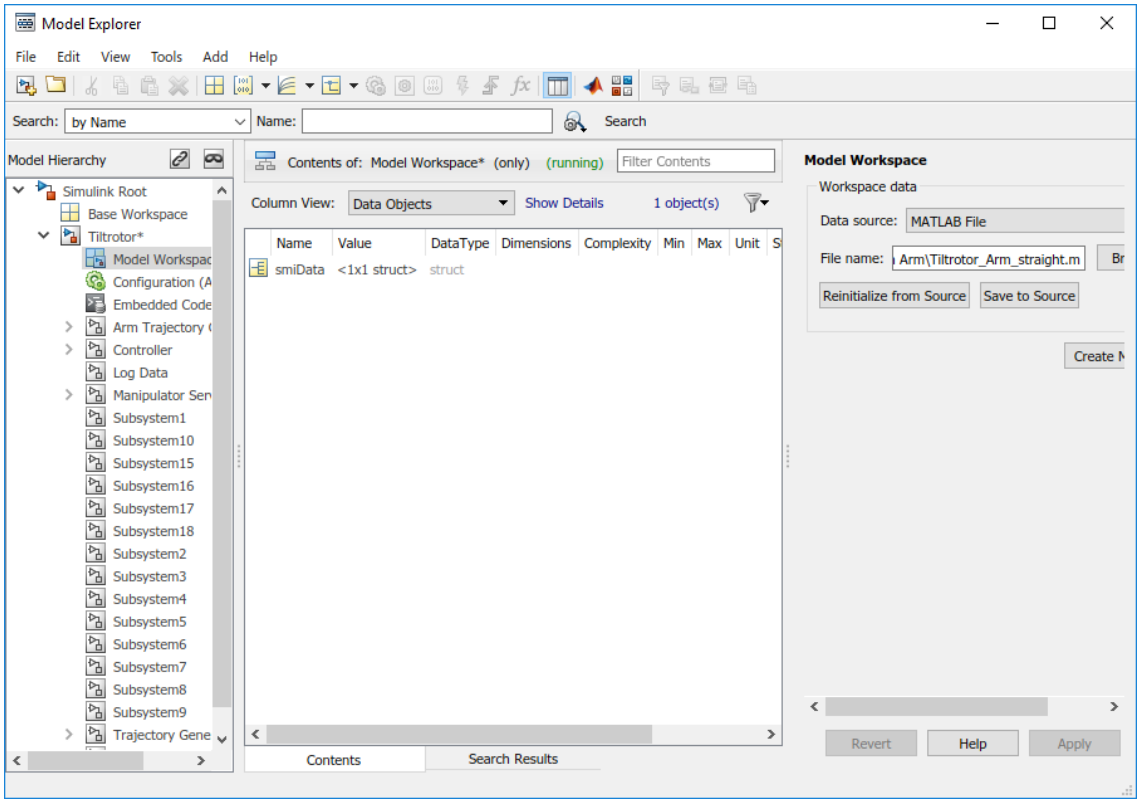

Figure 5.6: Model workspace reinitialization

blocks activates a pop-up menu with sliders to tune the selected parameters. The three blocks in the left column each tune a particular manipulator joint servo. In order from top to bottom, they tune the first arm joint servo, the second arm joint servo, and base turntable servo. Double-clicking on any of these results in the pop-up shown in Figure 5.7,where the sliders tune the K_P, K_I, and K_D gains for the controller of the servo selected.

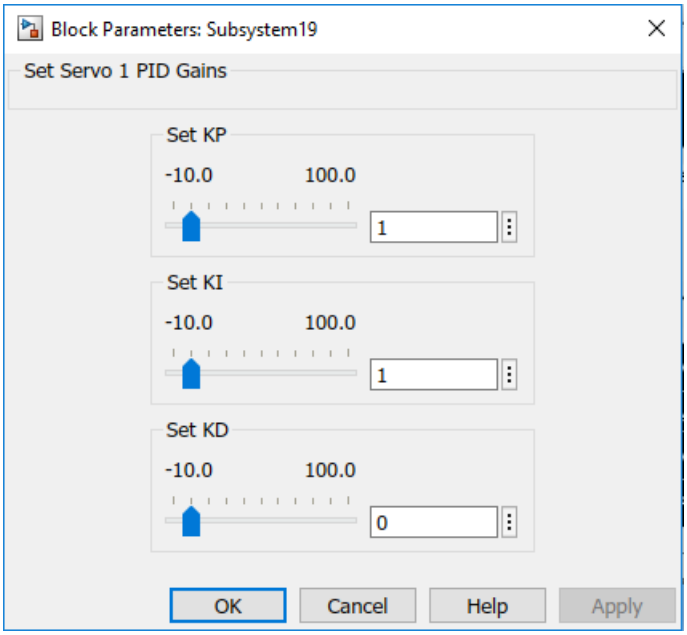

Figure 5.7: Tuning the manipulator servos PID gains

Moving on to the right column, double-clicking the top block `Tune H_max and sigma` generates the interface shown in Figure 5.8, where $h_{\max}$ and σ are presented in (3.35) and (3.43), respectively.

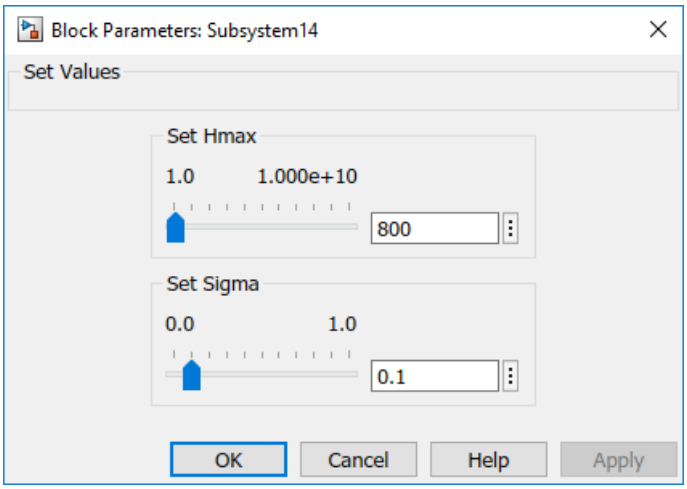

Figure 5.8: Tuning h_{max} and σ

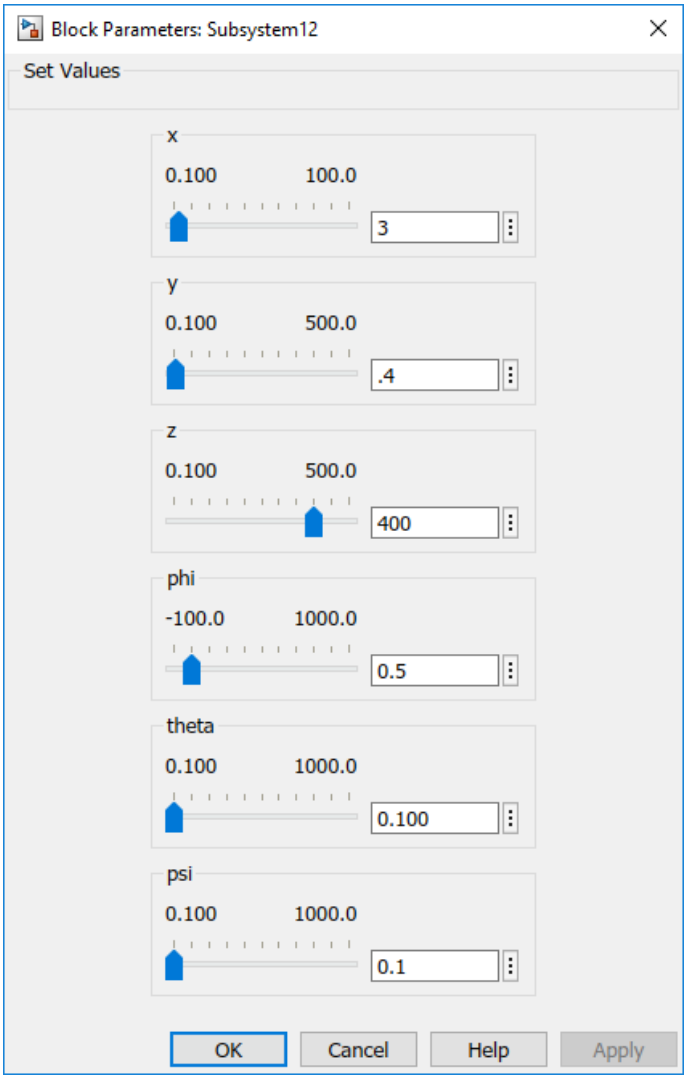

Figure 5.9: Tuning gains K_P and K_D

Next, by double-clicking either `Tune K_P` or `Tune K_D` the interface shon in Figure 5.9 appear. In this interface, the user inputs the diagonal terms of the gain matrices K_P and K_D for the MRAC control law, in order to tune the system. Each gain is labelled with the parameter it effects. Finally, by double-clicking `Run Set Up Script`, the user-selected tuning parameters in `Tune K_P` and `Tune K_D` are used to generate the A_{ref} matrix, which is used in turn to solve the Lyapunov equation fo the matrix P, which will be used later in the simulation.

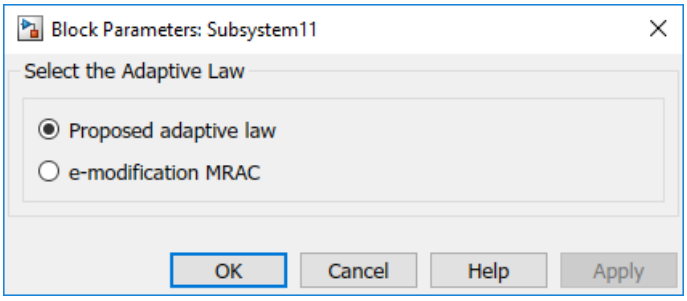

Figure 5.10: Selecting an adaptive law

Finally, by double-clicking the `Select an Adaptive Law` button, the prompt in Figure 5.10 appears, where the user can either select the MRAC control law presented in Chapter 3 and in [26], [27], or the e-modification MRAC control algorithm presented in [78].

5.4. Trajectory Formulation

Following the initialization and tuning of the simulation, the next step is to define the trajectories of both the tiltrotor and the manipulator end-effector. First, the method for generating the tiltrotor trajectory will be discussed, followed by the manipulator trajectory formulation.

5.4.1. Tiltrotor Trajectory Generation

To setup the tiltrotor trajectory, double-click the trajectory generation box from the main simulator screen, which should result in figure 5.11, which takes the time input and passes it into the user defined trajectory functions, and outputs $x_{\text{ref}}(\cdot), y_{\text{ref}}(\cdot), z_{\text{ref}}(\cdot), \theta_{\text{ref}}(\cdot)$, and $\psi_{\text{ref}}(\cdot)$. Double-clicking the blue box results in Figure 5.12, where the user defines the time dependent functions for $z_{\text{ref}}, x_{\text{ref}}, y_{\text{ref}}, \theta_{\text{ref}}$, and ψ_{ref}, in order from top to bottom. In this example, the $\theta_{\text{ref}}(t) = 0,\ t \geq t_0$

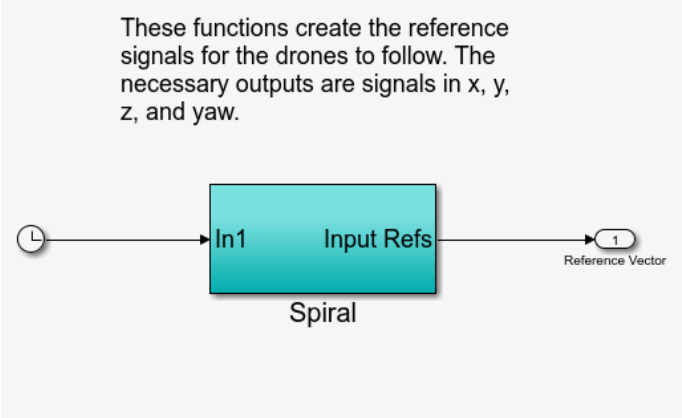

Figure 5.11: Vehicle trajectory generation

and $\psi_{\text{ref}}(t) = 0$, hence their assignment as a constant here, however, to define a time dependent trajectory function, double-click one of the Matlab function blocks pictured in Figure 5.12 to see the function shown in Figure 5.13, where the desired trajectory for each time interval is defined.

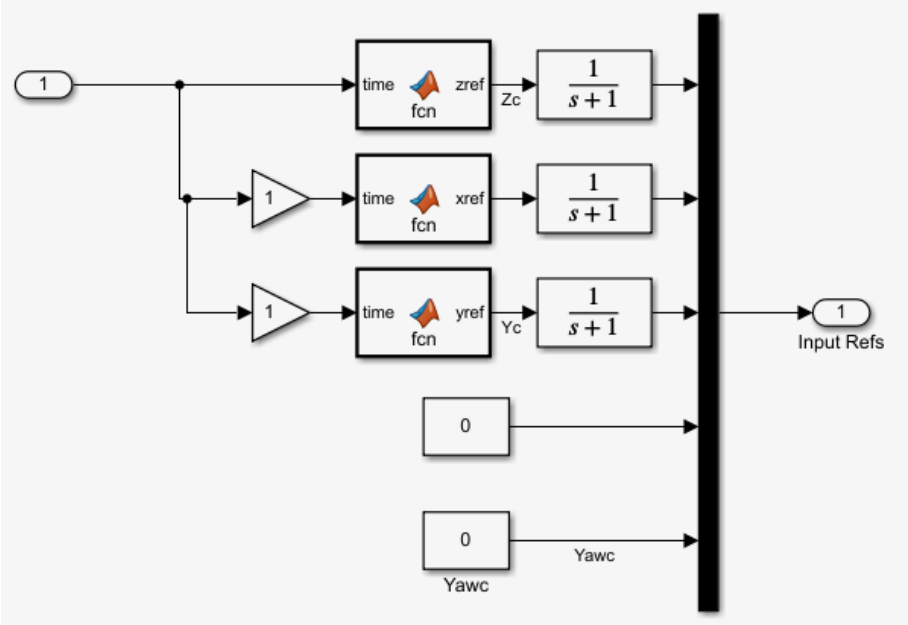

Figure 5.12: User defined trajectory functions

```
function xref = fcn(time)

xref = 0;
if (time >=6) && (time < 10)
    xref = 2*(time-6)/4;
end
if (time >=10) && (time < 50)
   xref = 2*cos((2*pi)/40*(time-10));
end
if (time >=50) &&(time < 65)
    xref = 2*cos((2*pi)/40*(40));
end
if (time >= 65)
     xref = 2*cos((2*pi)/40*(time-25));
end
```

Figure 5.13: Inside the user defined trajectory functions

These trajectory functions take the current simulation time, input it into the appropriate trajectory segment, and then pass into the simulation the desired position for that time. These functions can be edited by the user to have the tiltrotor track any chosen trajectory.

5.4.2. Manipulator Trajectory Generation

Similar to the tiltrotor trajectory creation, by navigating to the `Arm Trajectory Generation` block from the top level view, the arm trajectory can also be edited. Once the screen pictured in Figure 5.14 is visible, by double-clicking the Matlab function under `Input Desired end-effector Position` to setup up the desired positions, the function shown in Figure 5.15 will be displayed, where the initial conditions and the desired start and end points for the manipulator can be input for each time interval.

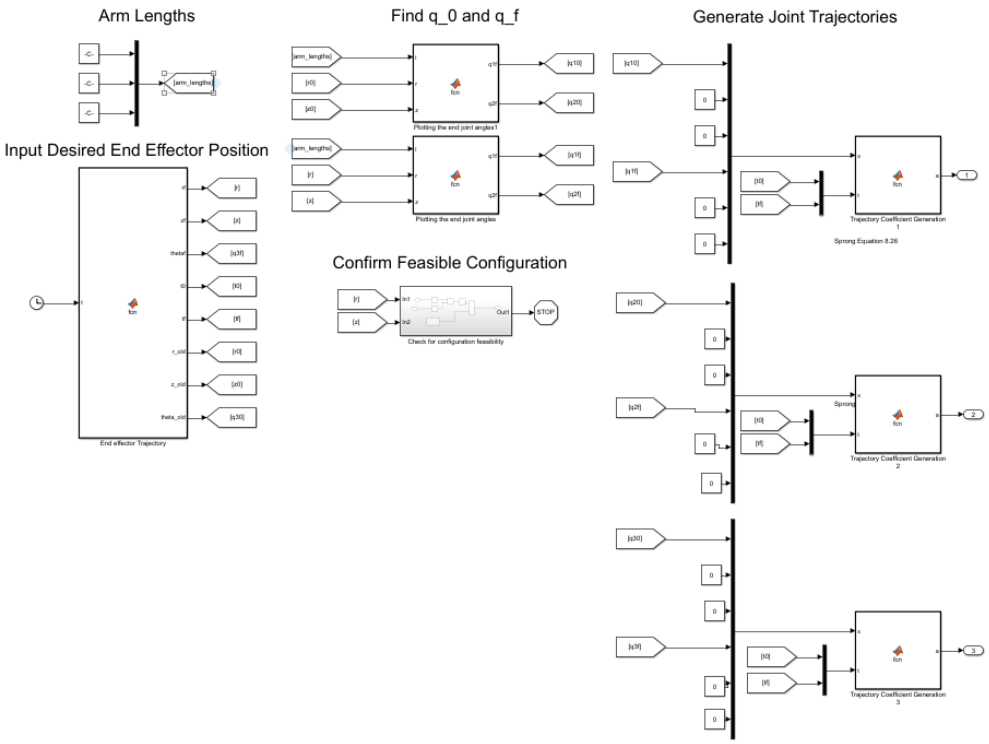

Figure 5.14: Manipulator trajectory generation

```
function [rf, zf, thetaf, t0,tf ,r_old,z_old,theta_old]  = fcn(t)
%define the end effector position of the arm system in cylindrical
%coordinates at each endpoint of arm motion  The position stated is the
%desired final position of the arm at the end of the segment of time
%indicated
    rf = 1.204; %ft
    zf = 0; %ft
    thetaf = 0; %rad
    t0 = 0;
    tf = 2;
    r_old = 1.204;
    z_old = 0;
    theta_old = 0;
if t > 0 &&t < 2
    r_old = 1.204;
    z_old = 0;
    theta_old = 0;
    t0 = 0; %initial time start
    tf = 2; %time end
    rf = 1.204; %ft
    zf = 0; %ft
    thetaf = 0; %rad

end
```

Figure 5.15: User defined manipulator trajectory

In each segment, update `r_old`, `z_old`, and `theta_old` with the final end-effector position from the previous time segment, set `t_0` to be the starting time of the current segment, `t_f` to be the ending time of the current segment, and `rf`, `zf`, and `thetaf` to be the desired final end-effector position for this time segment in cylindrical coordinates. These coordinates are considered to have their origin at the center of the first manipulator joint. Following the coordinate inputs, the function outputs the variables for the correct time step, which are then passed to the Matlab functions under `Find q_0 and q_f`, where the required joint angles will be found for the desired end-effector positions. Due to most end-effector positions having two possible configurations, this Matlab function is designed to choose the one that maximizes the manipulator distance from the body.

The `Confirm Feasible Configuration` function verifies that the desired end-effector position is within the feasible configuration space for the manipulator. Finally, the three functions under `Generate Joint Trajectories` apply the quintic polynomial trajectory generation method previously discussed in order to find the desired position of the manipulator joints for all time. In this section, the only two pieces that should require user modification are the `Arm Lengths` constants, if a new manipulator is installed in the simulator, and the `Desired end-effector Position` function, in order to generate a different manipulator trajectory.

5.5. Control of the Aerial Manipulator

Moving to the right of the two trajectory generation blocks in the top level view of the simulation, there are two blocks titled `Controller` and `Manipulator Servo`

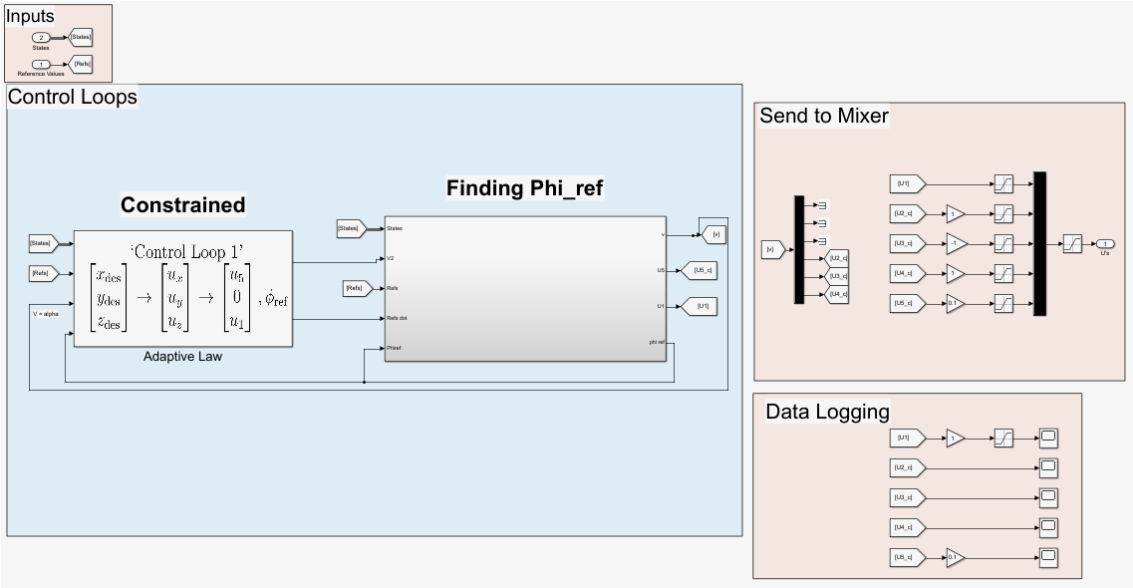

Figure 5.16: Tiltrotor MRAC control law

`Controller`. These blocks are the control laws for the tiltrotor and manipulator are implemented in the simulation.

5.5.1. Control of the Tiltrotor

To control the tiltrotor vehicle, double-click on the `Controller` block from the main simulation screen, which results in the interface shown in Figure 5.16. This block takes the state data found either through initialization or the last iteration of the simulation as well as the ideal trajectory of the system, and passes it first through the control law and then finds the $\phi_{\text{ref}}(\cdot)$ necessary to allow the vehicle to meet its trajectory goals. This results in the necessary control inputs in order to find the required motor thrust and tilt angles, calculated in the vehicle block. First, by double-clicking in the `Constrained` block, the user sees all the inputs being passed into the control law captured in Figure 5.17,

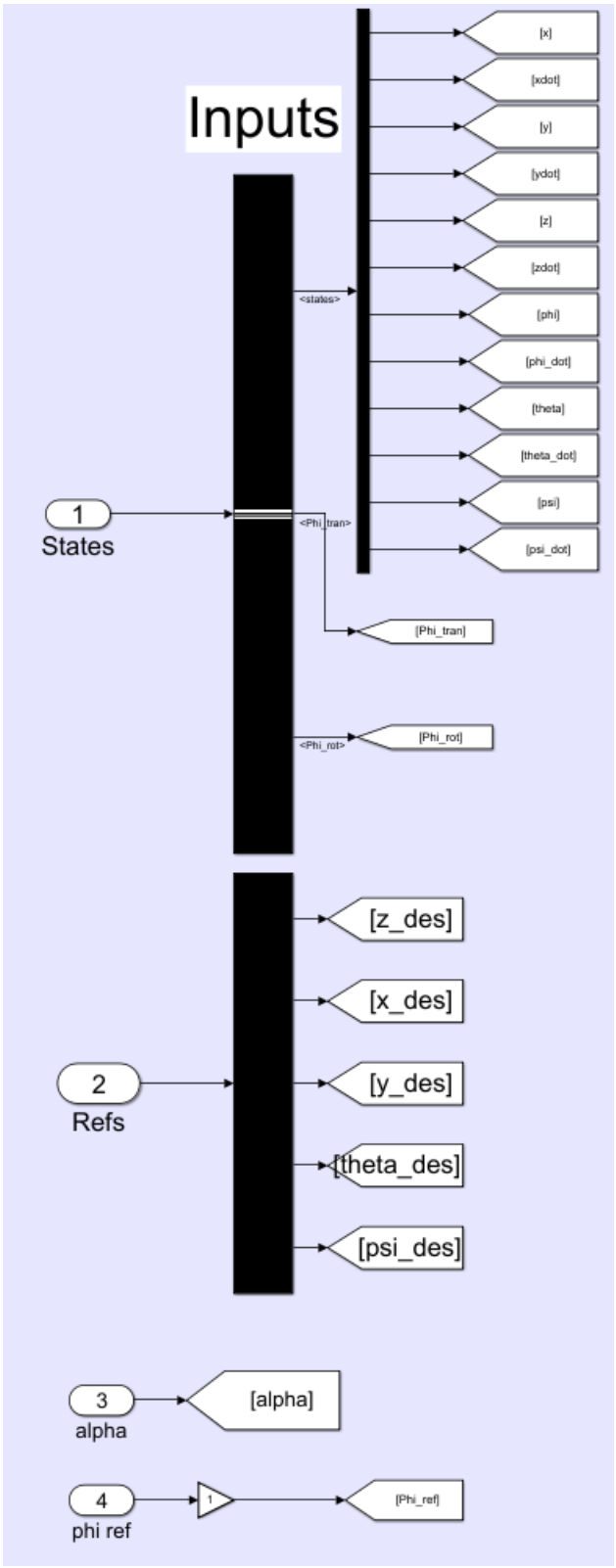

Figure 5.17: Inputs from simulator to the control law

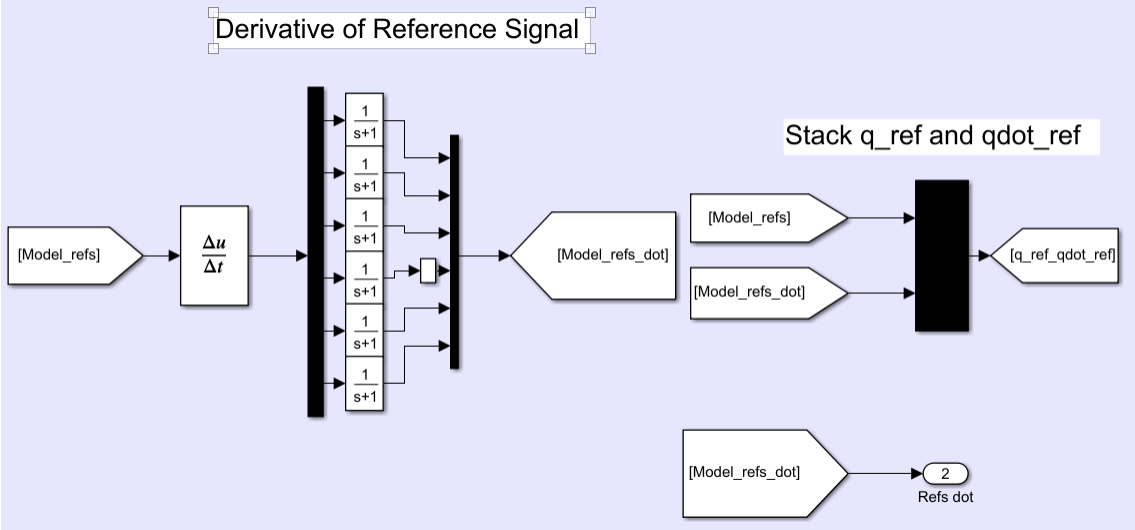

Figure 5.18: Derivation and stacking of the reference values

followed by the derivatives of the reference signal being taken and then stacked with the reference signal values (see Figure 5.18), and finally, the calculation of the error between the reference and actual values (see Figure 5.19).

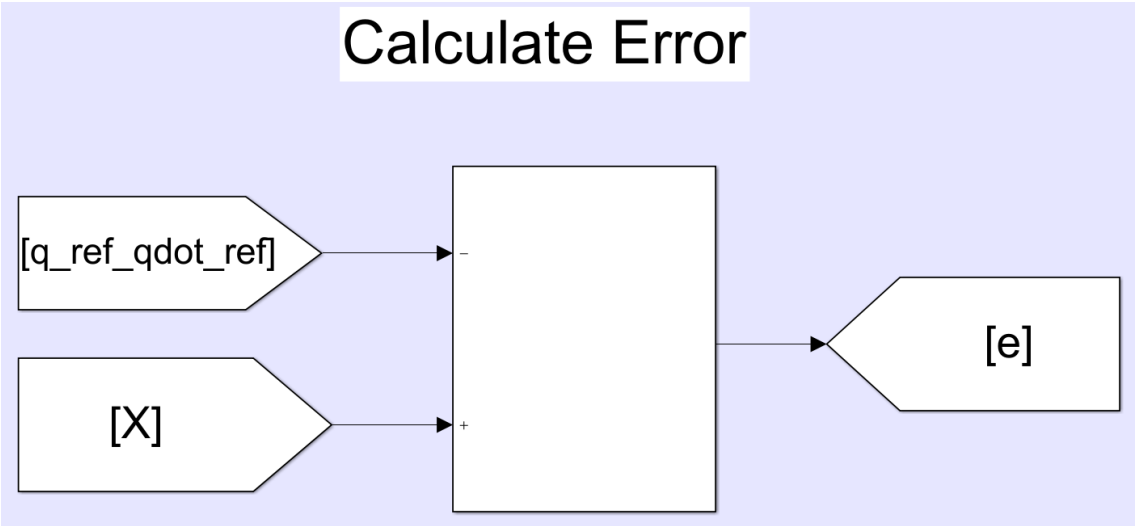

Figure 5.19: Calculation of trajectory tracking error

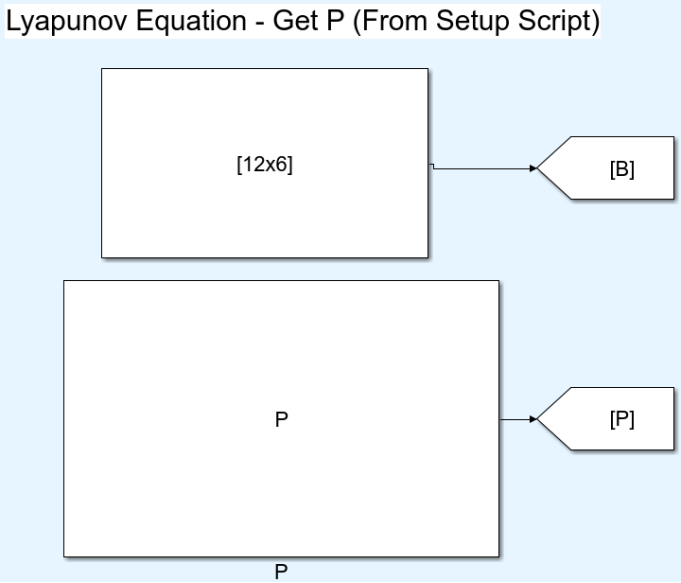

Figure 5.20: Introduction of Lyaponuv equation solution

From here, the solution of the Lyapunov equation from (3.70) is introduced (see Figure 5.20), leading to the calculation of our $h(\cdot, \cdot)$ value from Equation (3.35), shown in Figure 5.21.

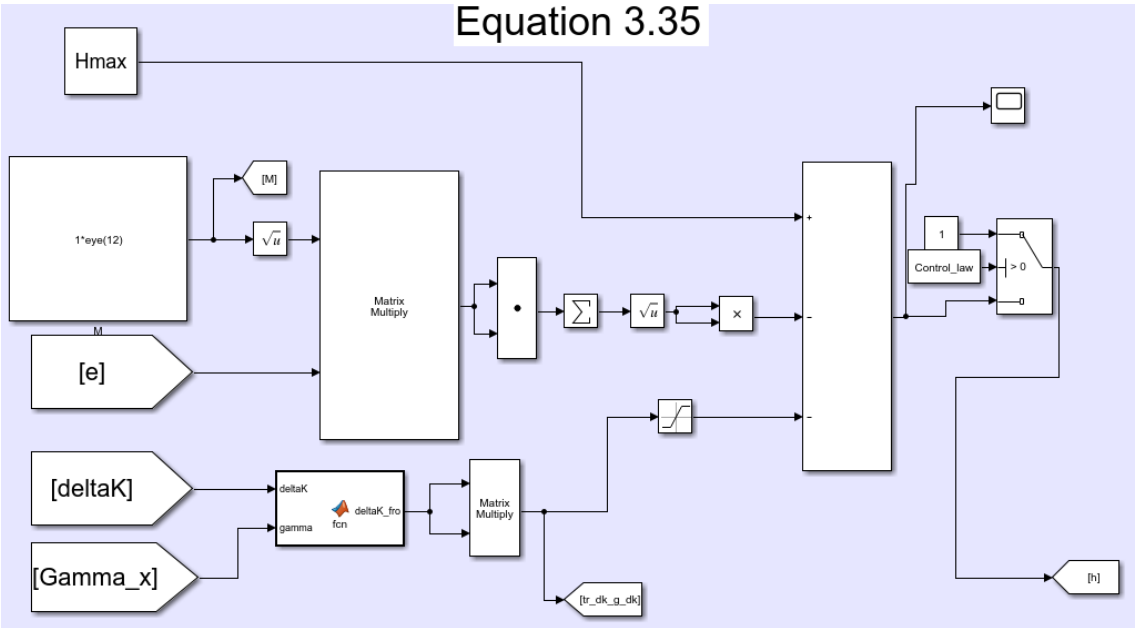

Figure 5.21: Calculation of $h(\cdot, \cdot)$

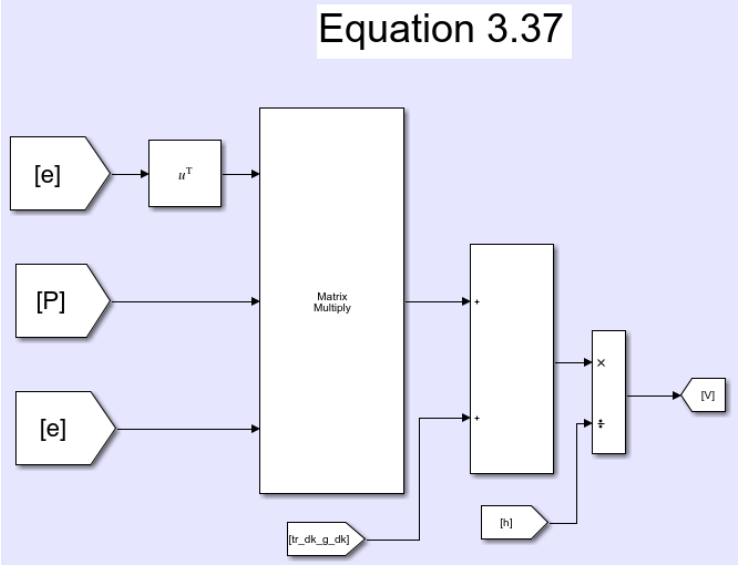

Figure 5.22: Calculation of $V(\cdot,\cdot)$

Next, $V(\cdot,\cdot)$ is computed from (3.37), shown in Figure 5.22, and used to calculate the adaptive gain $\hat{K}(\cdot)$ as in (3.43), shown in Figure 5.23. Of note is that the `Gamma_x` function, circled in Figure 5.23, can be edited by the user in order to change the

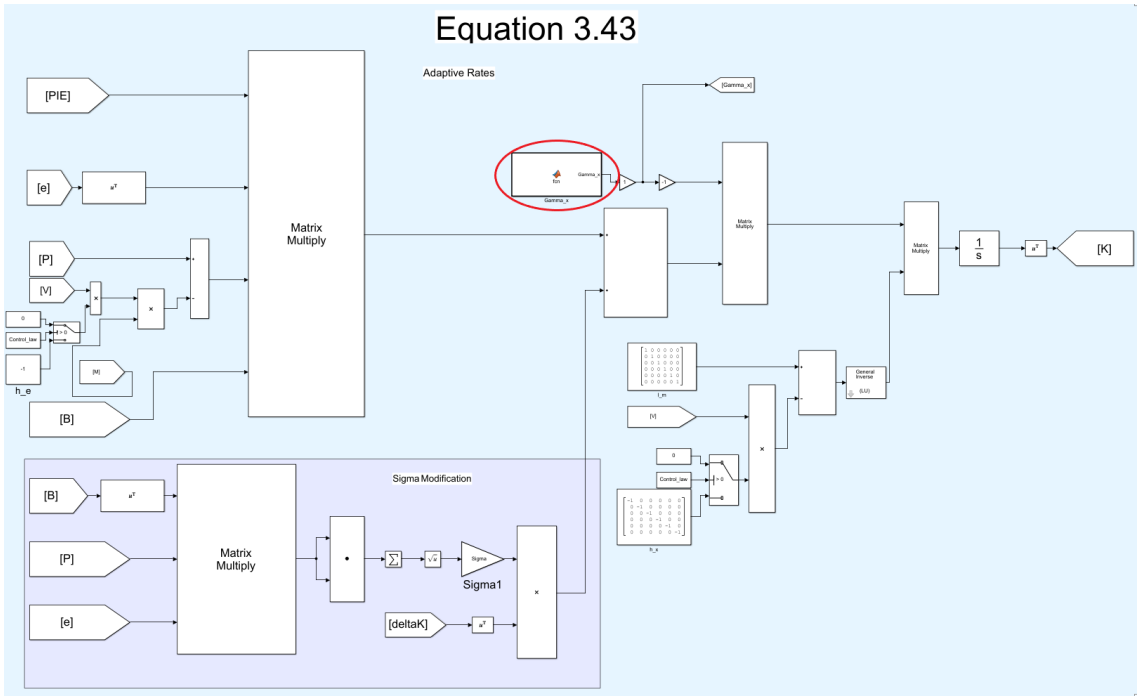

Figure 5.23: Computation of the adaptive gain $\hat{K}(\cdot)$

```
function Gamma_x = fcn()
%This function allows the user to tune the weighting the adaptive law gives
%to each parameter of the adaptive law
Gamma_x = eye(57);
Gamma_x(1,1) = 3; %x (what corresponds to pi(t))
Gamma_x(2,2) = 10; %y
Gamma_x(3,3) = 250;%z
Gamma_x(4,4) = 60;%phi
Gamma_x(5,5) = 25;%theta
Gamma_x(6,6) = 30; %psi
Gamma_x(7,7) = 1; %xdot
Gamma_x(8,8) = 5; %ydot
Gamma_x(9,9) = 75;%zdot
Gamma_x(10,10) = 60;%1 phidot
Gamma_x(11,11) = 25;%thetadot
Gamma_x(12,12) = 25;%psidot
Gamma_x(13,13) = 1; %Reference (set to zero)
```

Figure 5.24: Function to allow user to tune the weighting matrix Γ_x

weighting of specific states in the adaptive law. The specific state each parameter effects is commented into the function, illustrated in Figure 5.24. Continuing from here, $\Delta K(\cdot)$ is then calculated in the block shown in Figure 5.25.

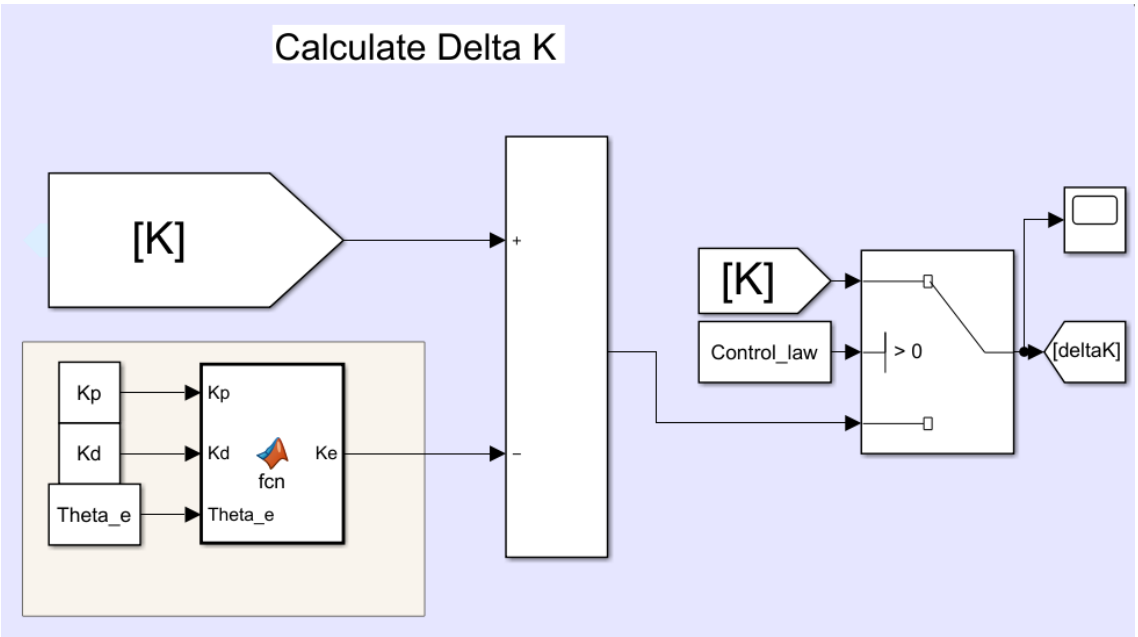

Figure 5.25: Computation of $\Delta K(\cdot)$

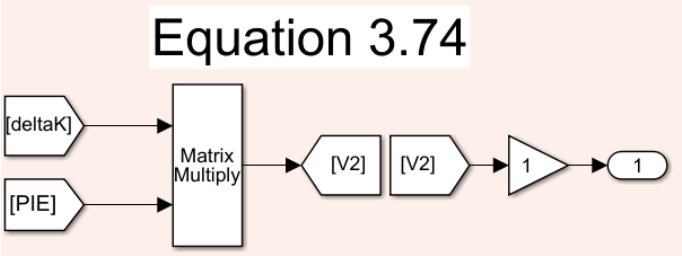

Figure 5.26: Computation of $v_2(\cdot)$

Finally, $v_2(\cdot)$ is captured using Equation (3.74) as shown in Figure 5.26 and then passed to `Finding Phi_ref`. Going back out into the `Controller` Subsystem and then double-clicking on the `Finding Phi_ref` block, the value of $v_2(\cdot)$, the actual state values, reference state values, center of mass, and inertia properties are imported, as shown in Figure 5.27.

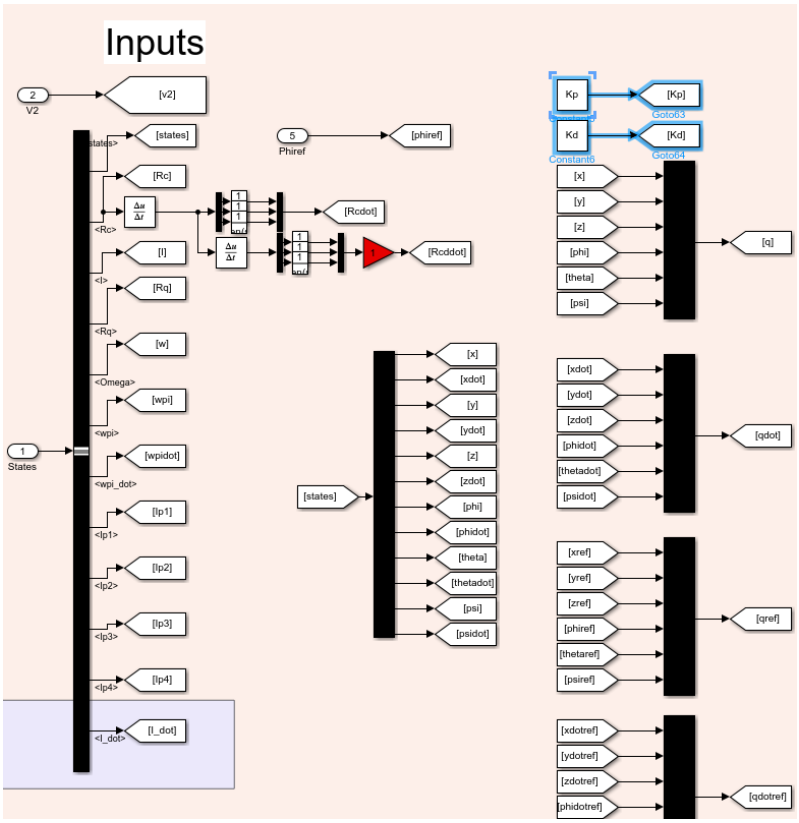

Figure 5.27: Inputting parameters to find ϕ_{ref}

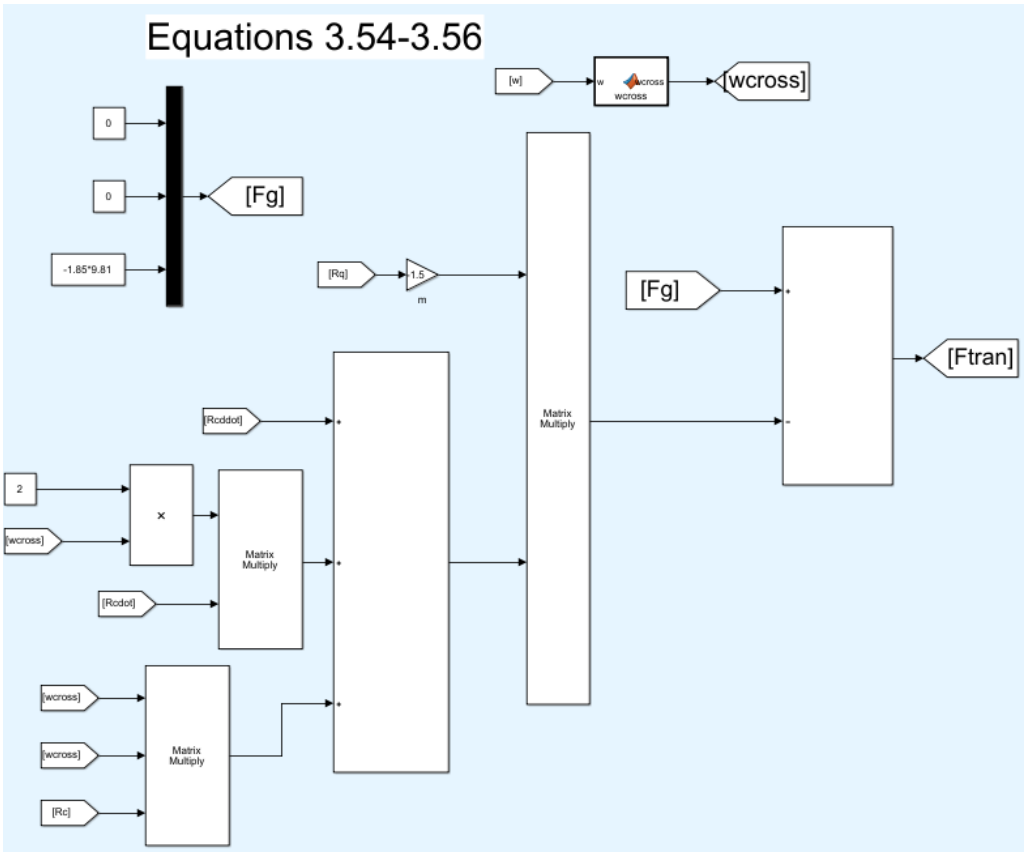

Figure 5.28: Computation of $f_{\text{tran}}(\cdot, \cdot, \cdot)$

Next, $f_{\text{tran}}(\cdot, \cdot, \cdot)$ and $f_{\text{rot}}(\cdot, \cdot, \cdot)$ are calculated in (3.54)-(3.56) and (3.57)-(3.59), respectively, and captured as shown in Figures 5.28 and 5.29.

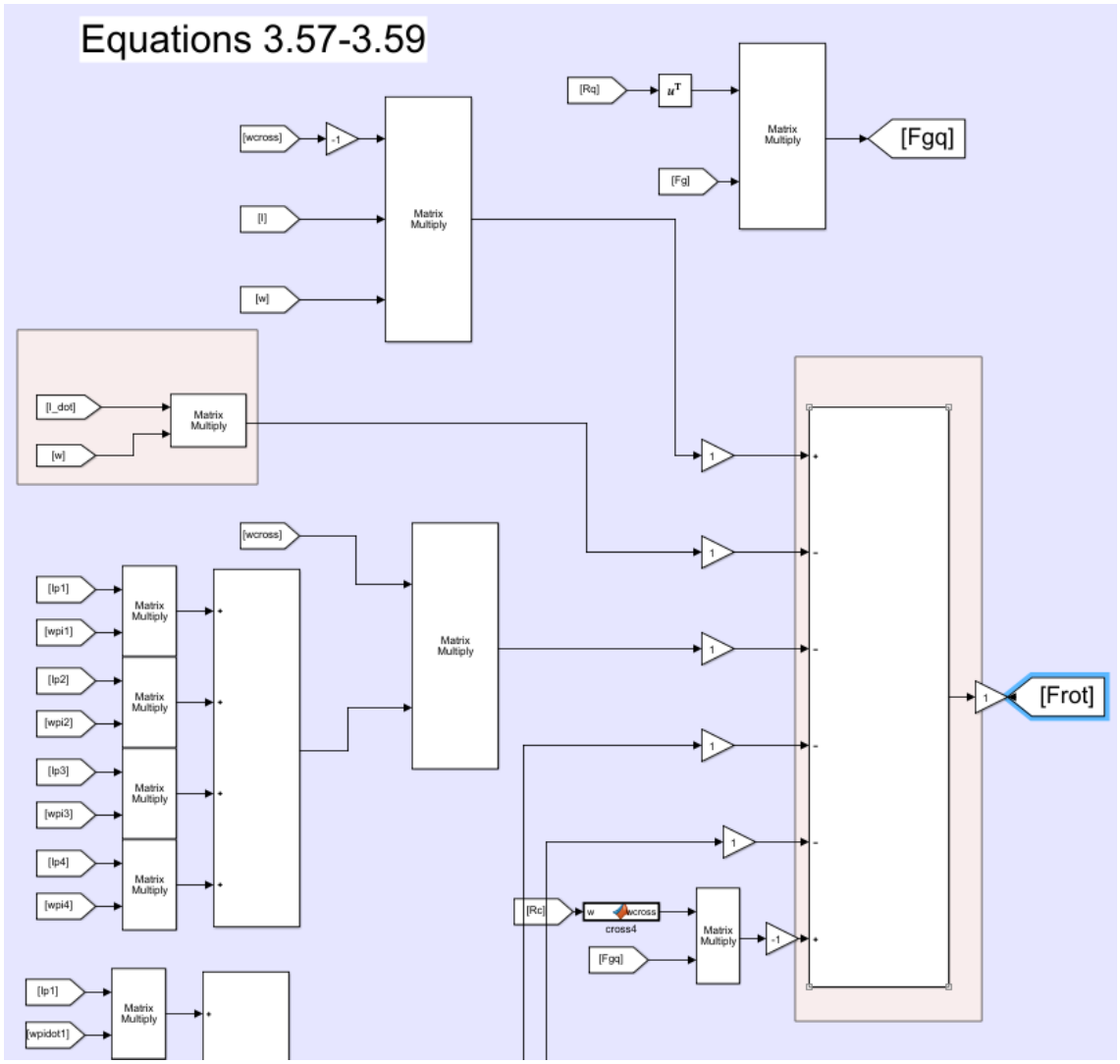

Figure 5.29: Computation of $f_{\text{rot}}(\cdot, \cdot, \cdot)$

The feedback term is then found using Equation (3.60) as shown in Figure 5.30.

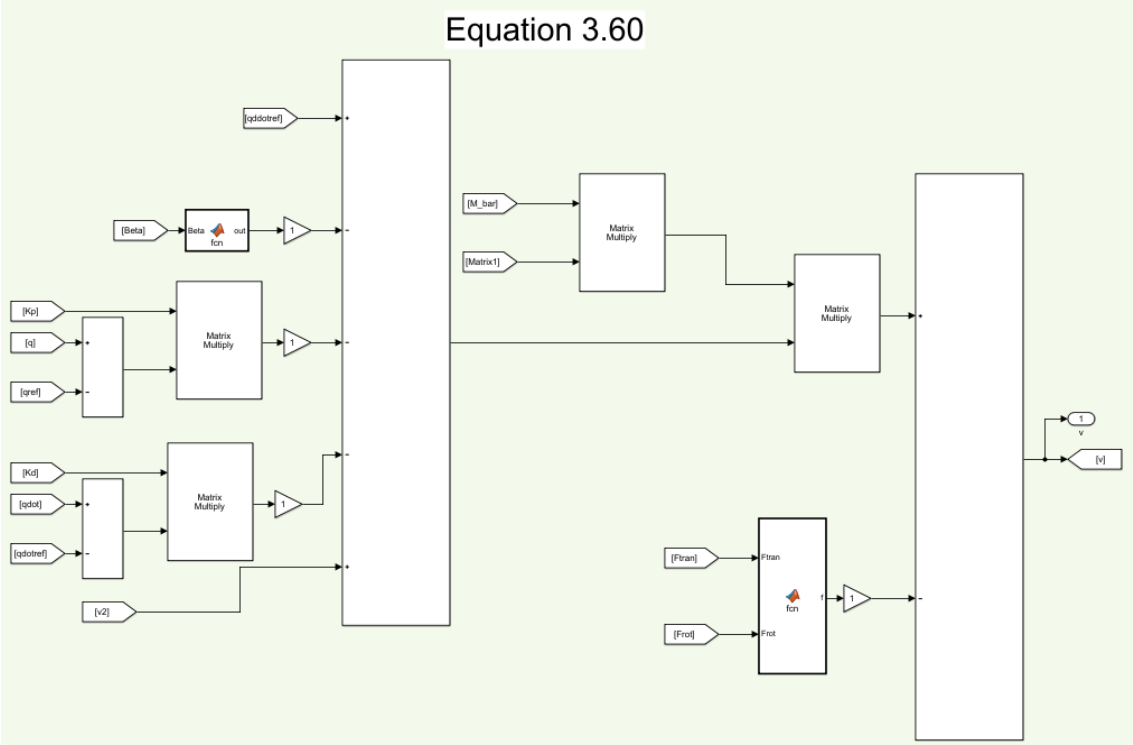

Figure 5.30: Computation of $\alpha(\cdot)$

Finally, $\phi_{\text{ref}}(\cdot)$ is found as captured in Figure 5.31, and passed back to the `Constrained` block as well as to the `Vehicle` block

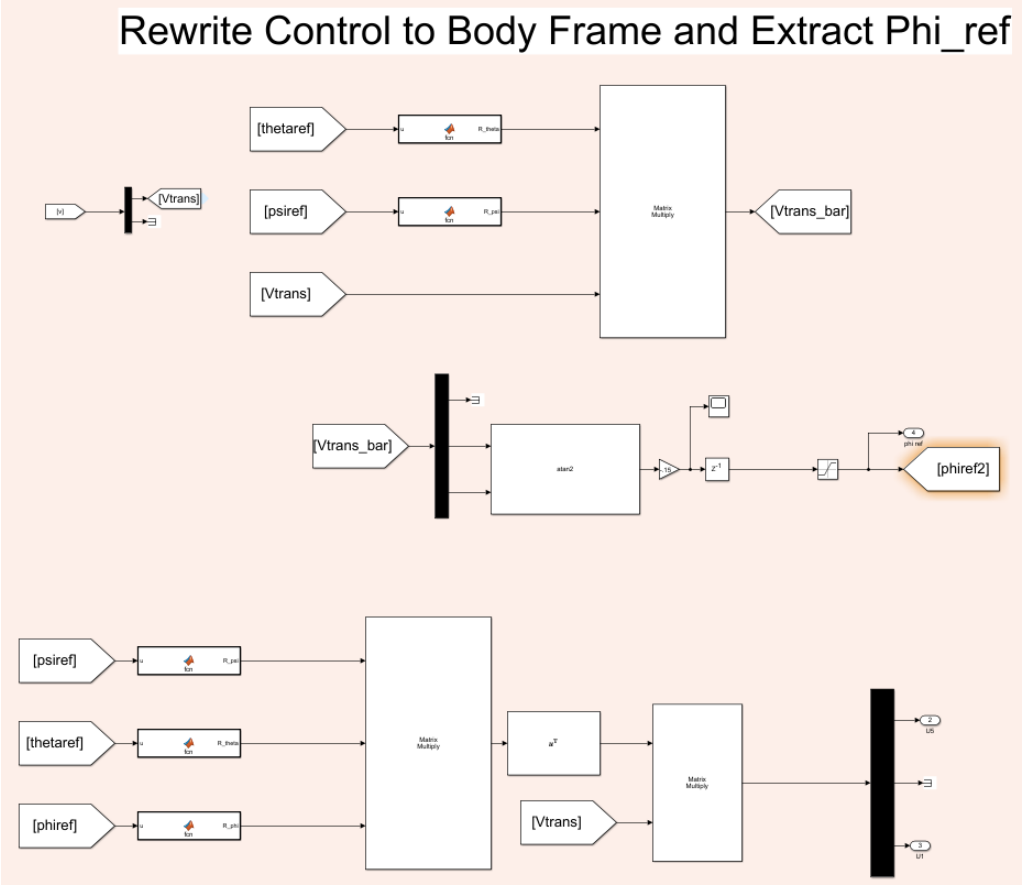

Figure 5.31: Extraction of ϕ_{ref}

5.5.2. Control of the Robotic Manipulator

From the top level view, by double-clicking the Manipulator Servo Controller block, the screen shown in Figure 5.32 should appear. This interface shows the control blocks for each of the manipulator servos. Due to each joint using `Dynamixel A18` servos, all the servo subsystems have the same properties. Double-clicking in any of the servo subsystem blocks, the content of Figure 5.33 is shown. Proceeding from the left, this subsystem takes the quintic polynomial coefficients calculated in the trajectory generation subsystem, and the current time, and outputs to the

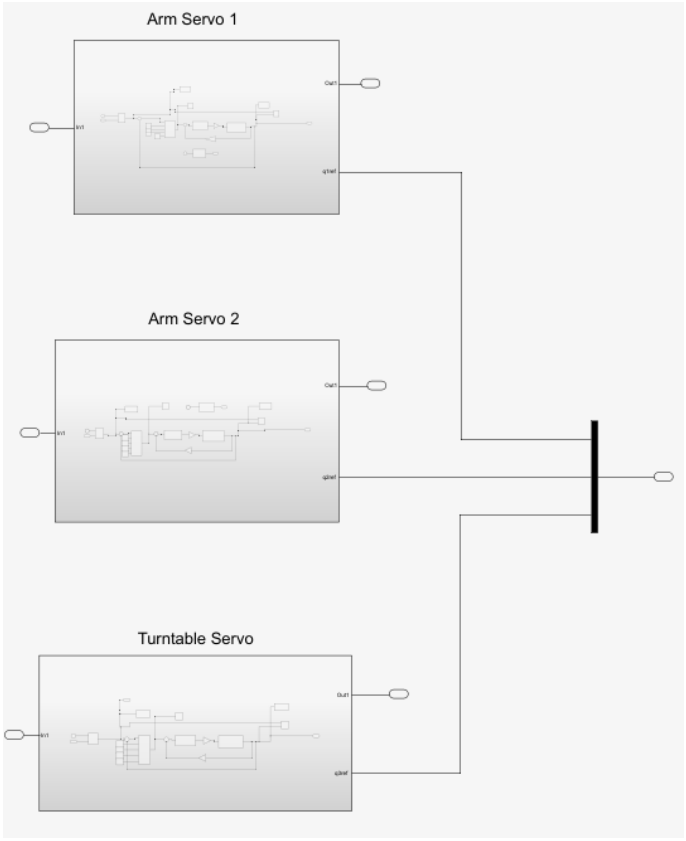

Figure 5.32: Manipulator servo controller

system the current desired joint angle from the `q computation` function. This desired position then gets passed through the PID controller and the mechanical and electrical transfer functions of the servo. The first transfer function from the left is the electrical transfer function given by (4.17) and the rightmost transfer function

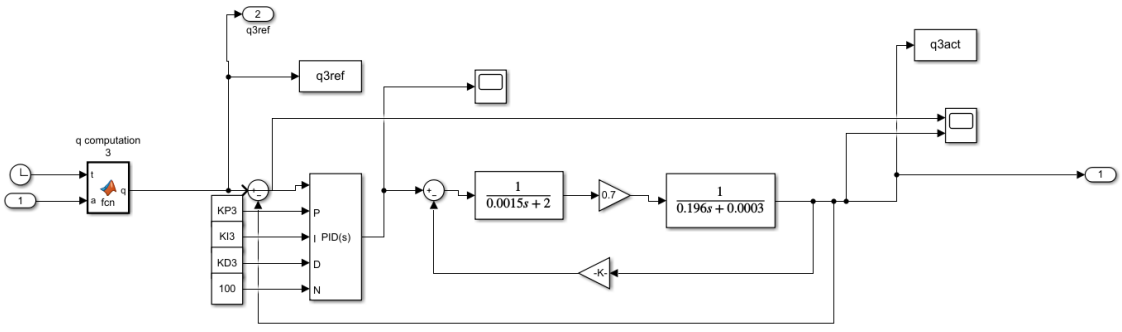

Figure 5.33: Servo model and PID controller

captures (4.18). Finally, the feedback gain in the model is the back-electromagnetic frequency of the servo. The actual joint angle position of the servo is then passed out of the subsystem, in addition to the reference position given by the `q computation` function. This process is performed for all three of the servo subsystems, resulting in the actual and reference servo positions being passed to the manipulator model.

5.6. Aerial Manipulator Force and Moment Calculations

Following the control formulation, the forces and moments acting on the aerial manipulator system are calculated by the simulation. By double-clicking on the `Vehicle` block, the user should see the interface in Figure 5.34. Starting with the `Main Vehicle` subsystem, by double-clicking on the block, the main vehicle Simmechanics blocks will be displayed as in Figure 5.35.

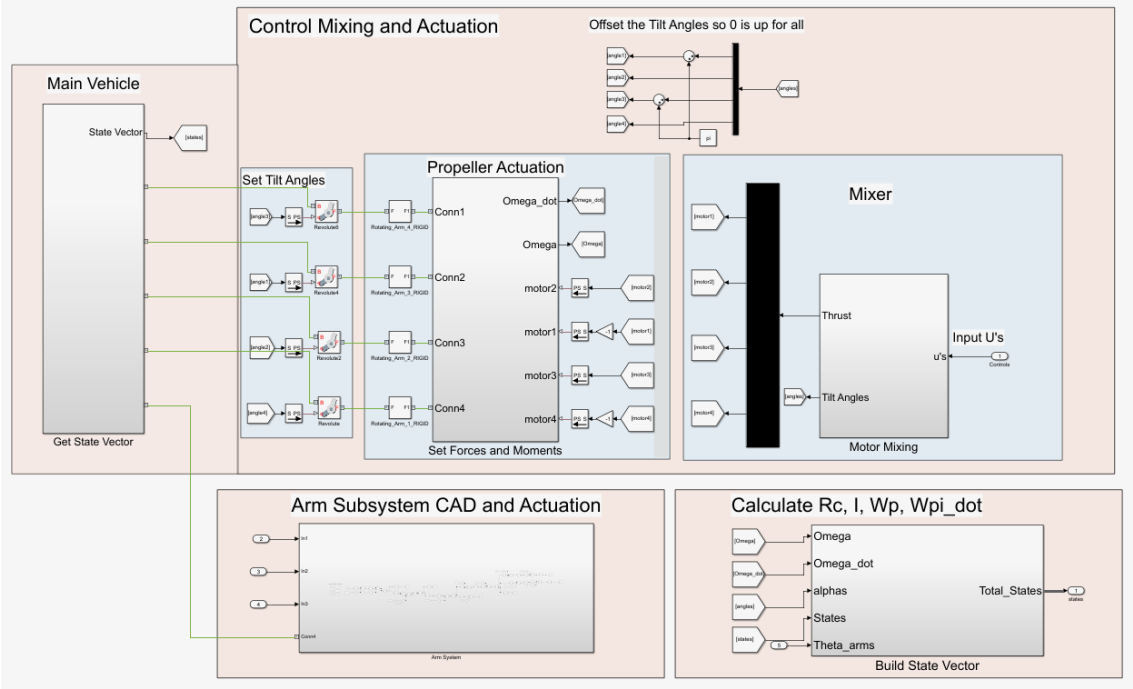

Figure 5.34: Vehicle Subsystem simulation block

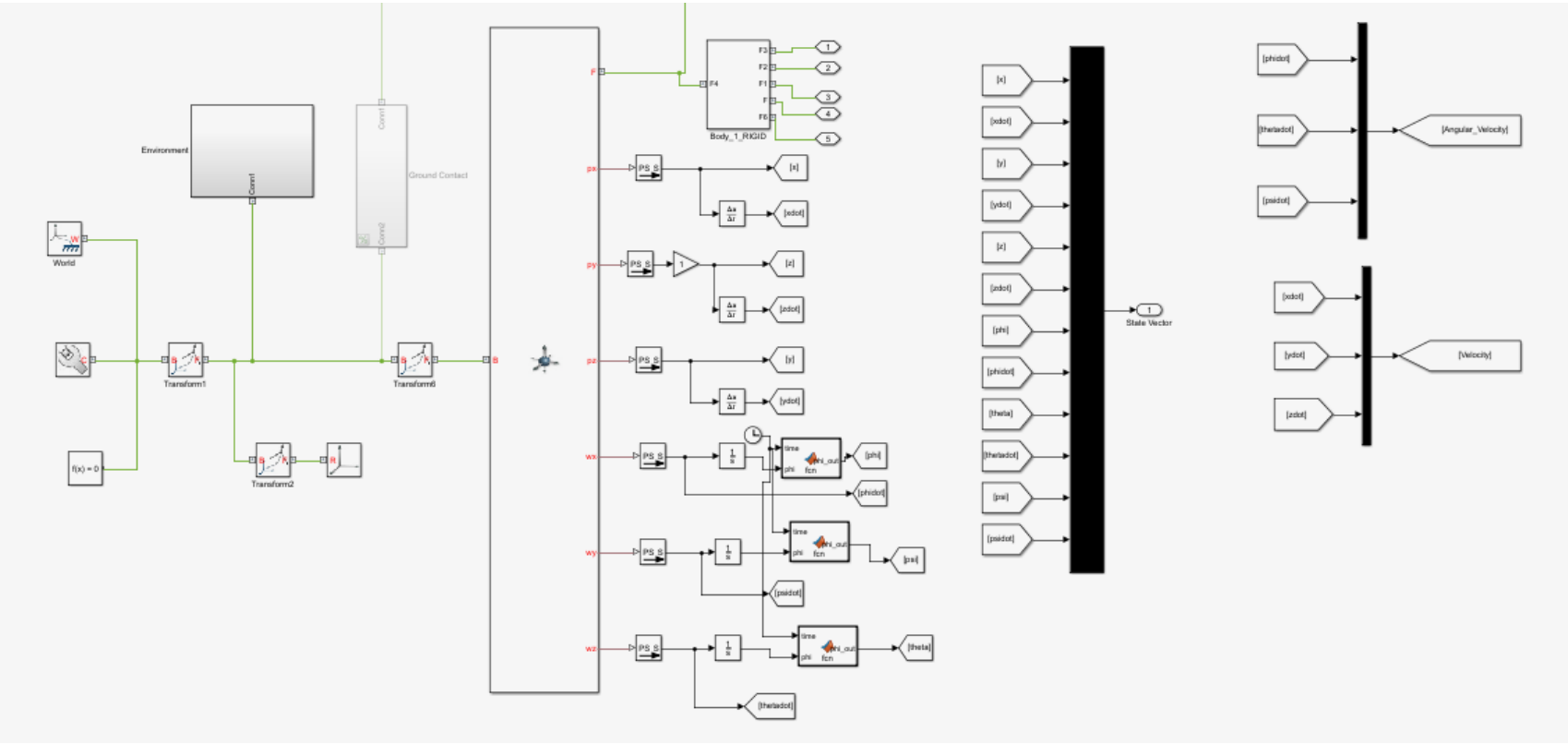

Figure 5.35: Main Vehicle simulation block

At the far left, the world reference frame, simulation parameters, and simulation solver blocks are stacked vertically. These all will be generated automatically by Simulink upon the importing of the CAD model into the Simulink environment, and so will not need to be touched by the user. The `Environement` subsystem contains the building and ground visuals for the simulation, and the large `6-dof` block is where all the forces and moments on the aerial manipulator are calculated by the system, where they are then passed to the state vector for use elsewhere. Finally, the transform blocks throughout this model are what define the position of each vehicle part with respect to the previous part, and are all defined through the model workspace previously discussed. Once again, this is something that is done during the first time the model is imported in Simulink and will not need to be changed by the user. Similarly, back in the `Vehicle` subsystem, clicking on the `Arm Subsystem` displays all the arm Simulink blocks, where, from left to right, the turn table and first link blocks are shown in Figure 5.36, followed by the first arm joint and second arm joints and their attached links in Figures 5.37 and 5.38. The blocks

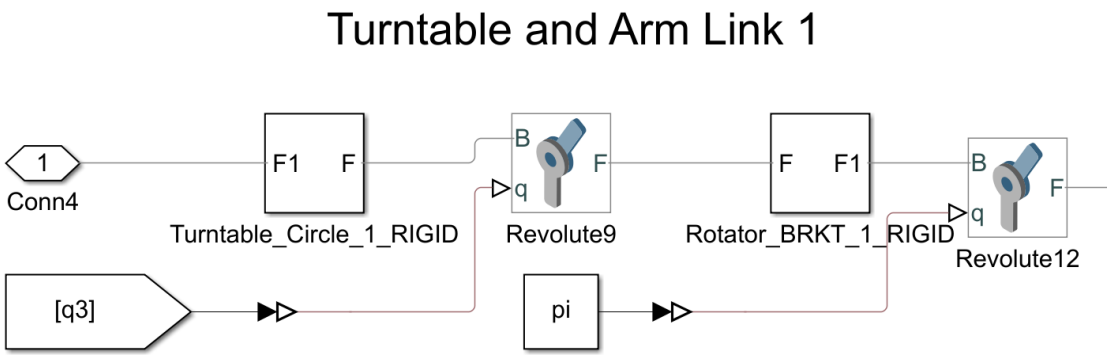

Figure 5.36: Turntable and first arm link

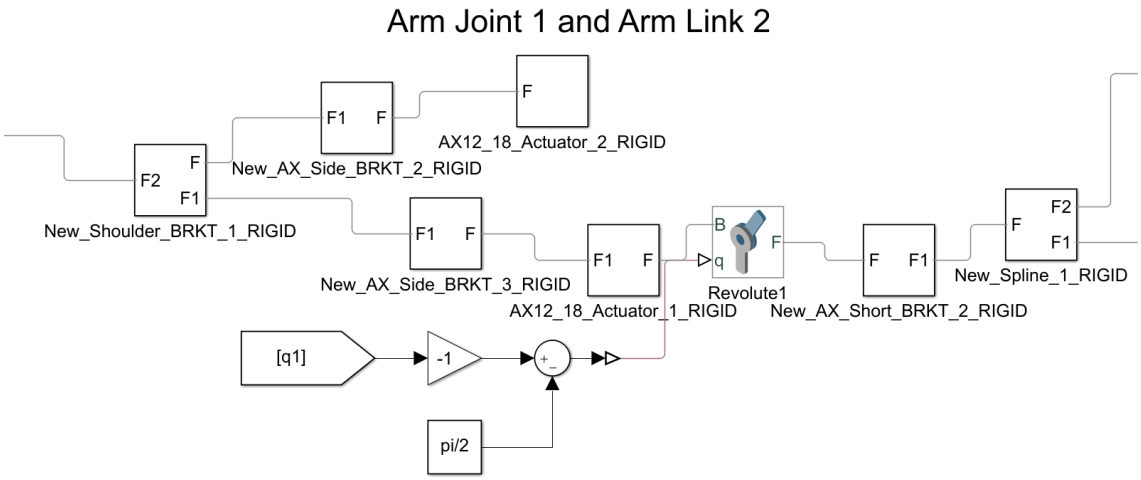

Figure 5.37: First arm joint and second arm link

that capture the wrist joint and gripper claws are shown in Figures 5.39 and 5.40.

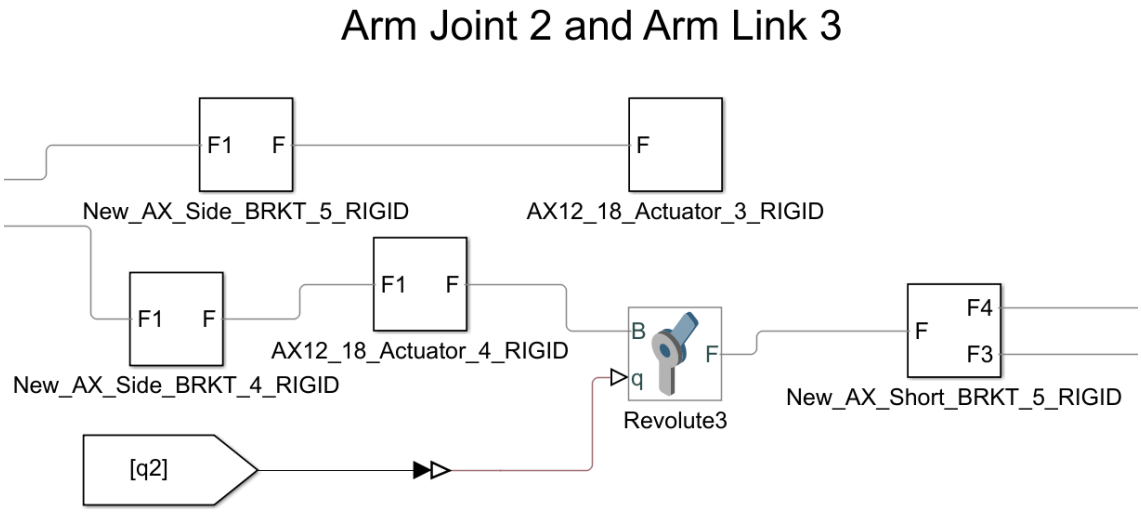

Figure 5.38: Second arm joint and third arm link

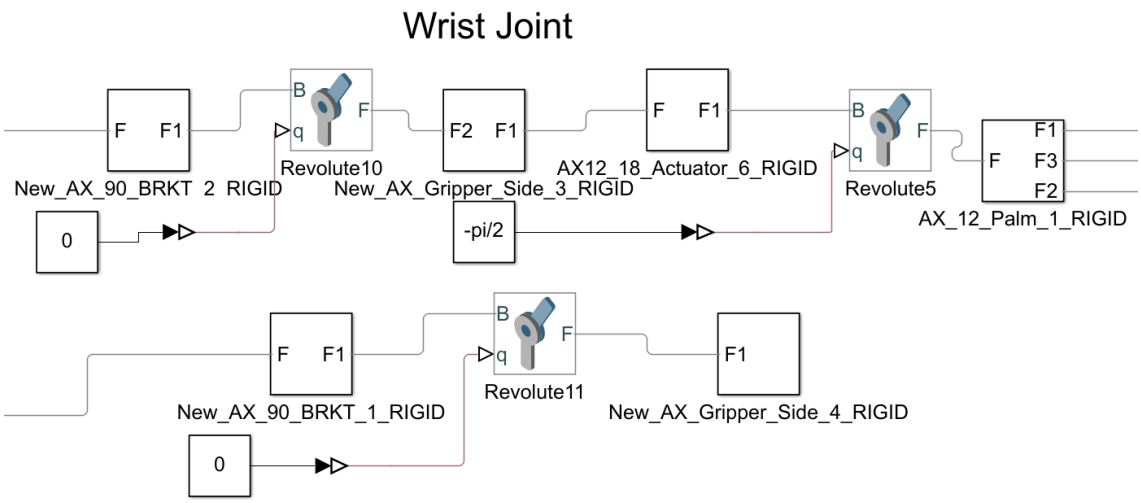

Figure 5.39: Wrist joint and palm

These blocks establish the position of each part in the CAD model in relation to the others, as well as the angles of each of the revolute joints in the manipulator system. One final consideration is that each block is named after its respective CAD file, so any errors in the system visualization can be easily found to correct in the

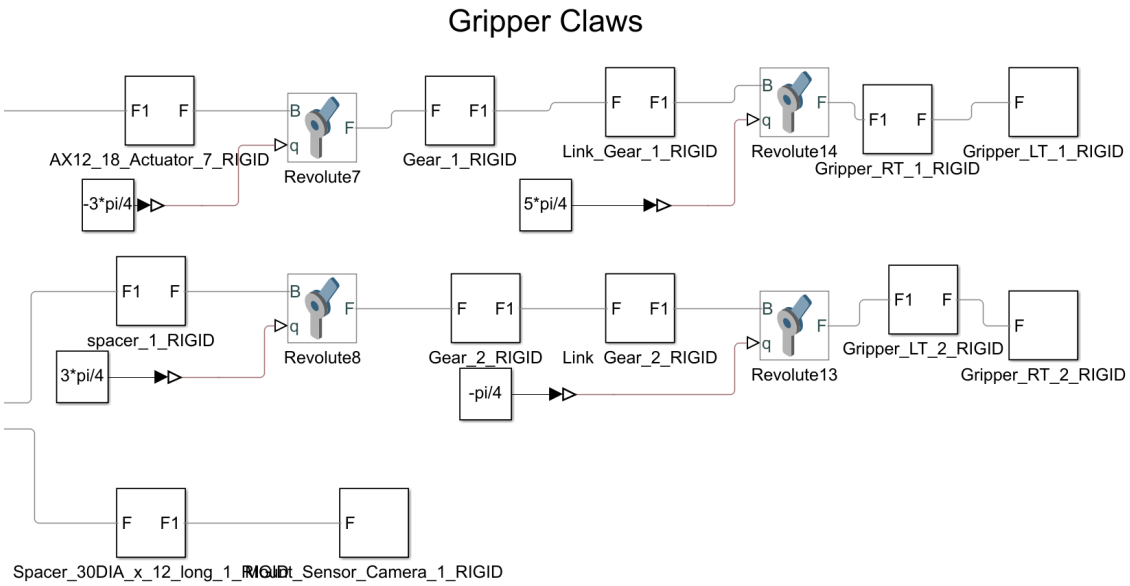

Figure 5.40: Gripper claws and actuator

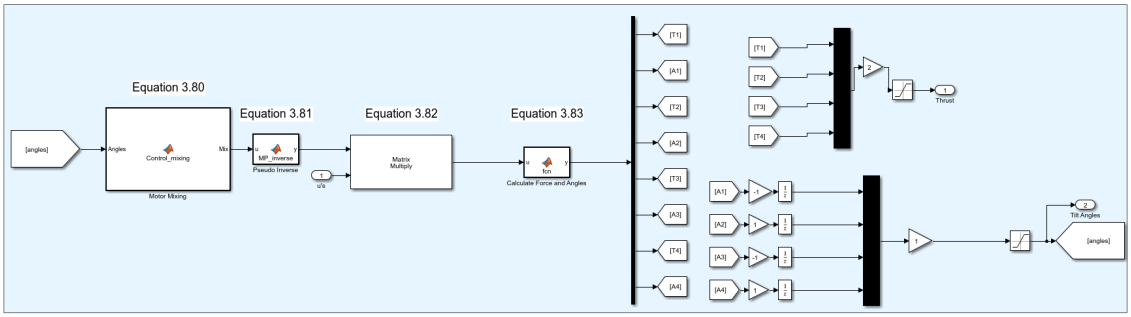

Figure 5.41: Motor thrust and tilt angle calculation

Solidworks model.

Next, going back to the `Vehicle` block, double-clicking on `Motor Mixing` subsystem displays the group shown in Figure 5.41, where the control inputs from the vehicle controller are converted into the necessary motor thrusts and propeller tilt angles, using Equations (3.80)-(3.83). These are then passed back out into the `Vehicle` where they undergo a rotational offset to correspond with the vehicle reference frame, then the tilt angles fed into the Simulink model while the motor thrusts go into the `Propeller Actuation` system shown in Figure 5.42, where the thrust and drag force of each propeller is calculated and the results passed to the Simulink model. Finally, the mass and inertia properties, angular velocities, and regressor vector are calculated in the `Calculate Rc, I, Wp, Wpi_dot` block, shown in Figure 5.43

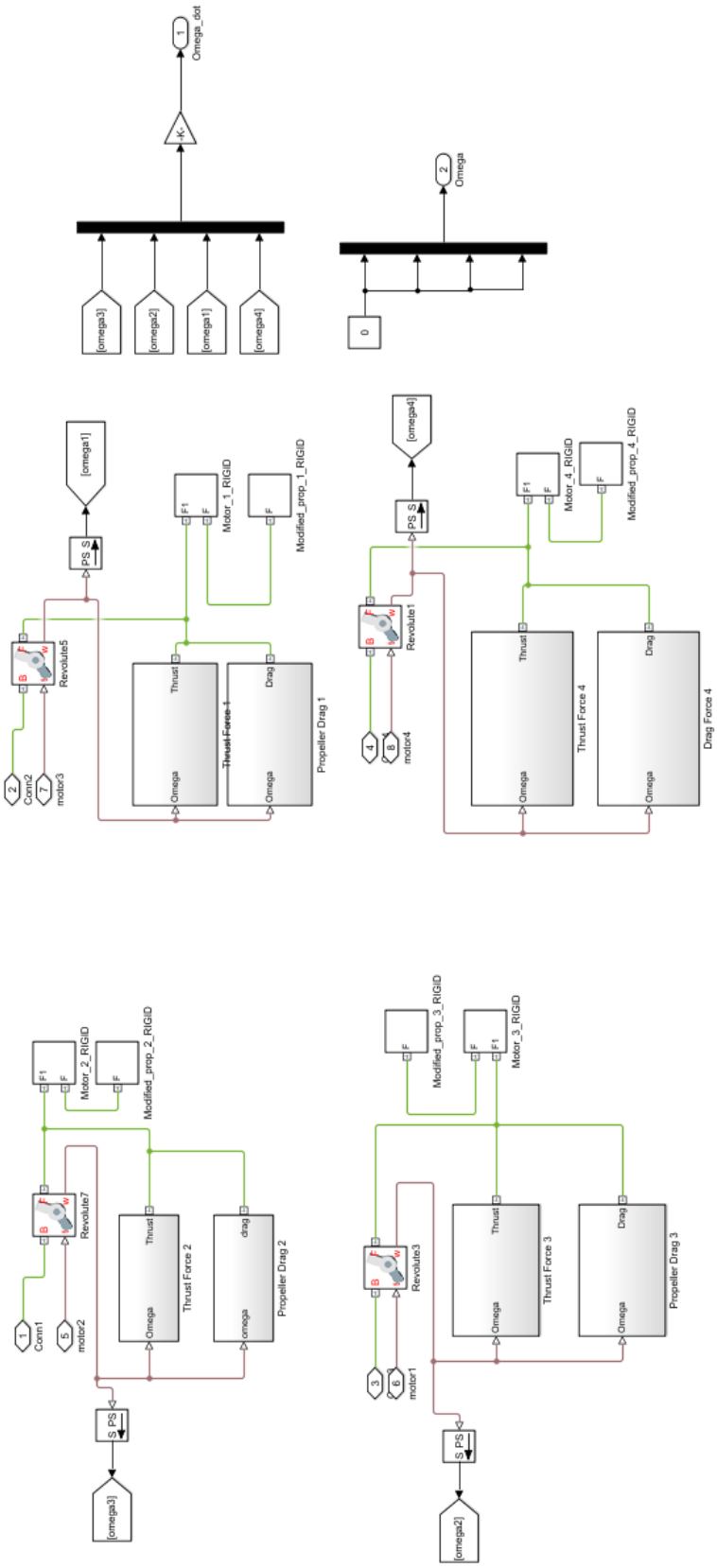

Figure 5.42: Actuation of the propellers

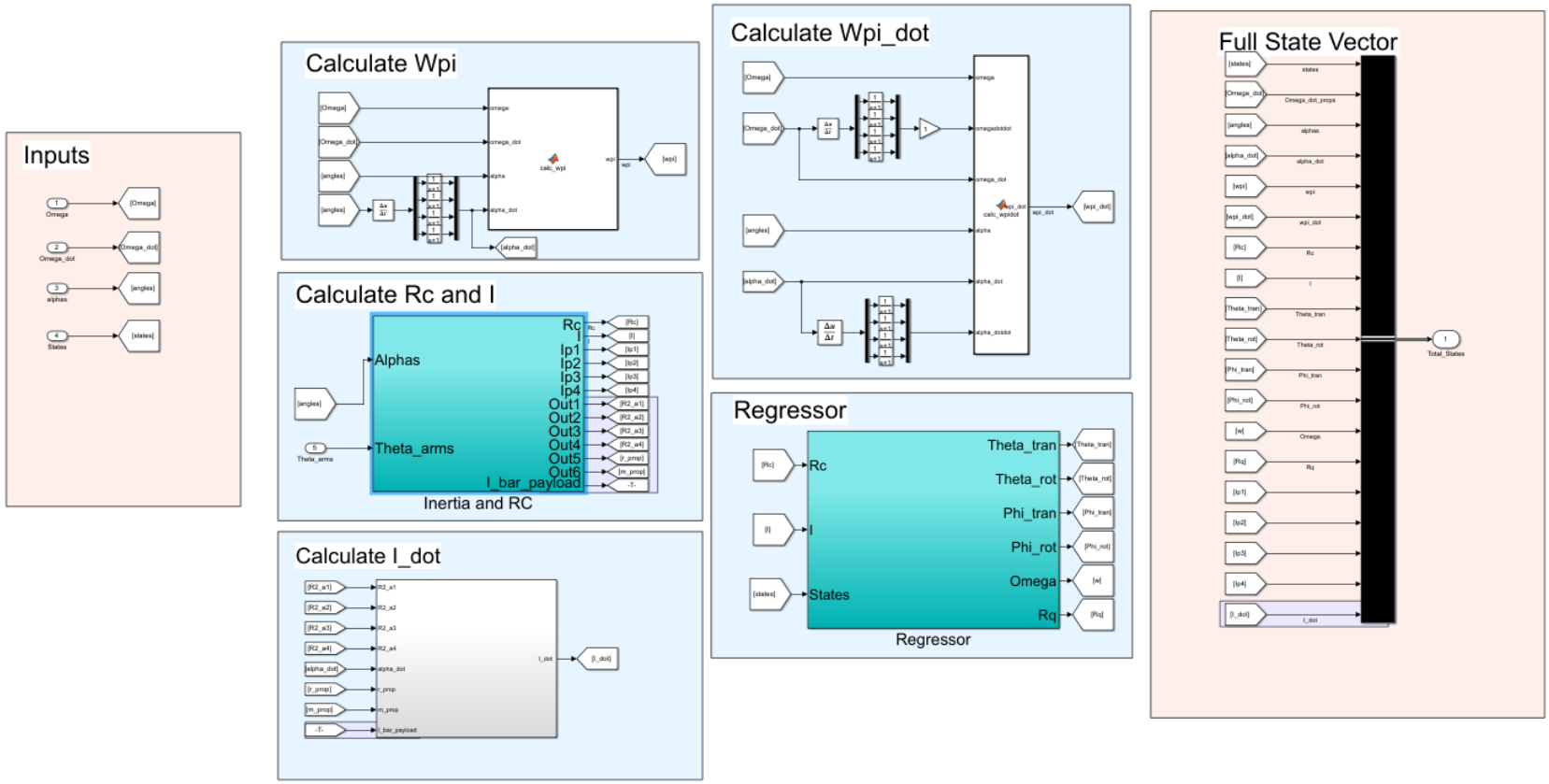

Figure 5.43: Mass, inertia, and regressor vector Calculations

The first subsystem of note in Figure 5.43 is the `Calculate Rc and I` block. In this block, the mass, inertial, and dimensional properties of the components of the aerial manipulator are explicitly defined, and will require updating if the vehicle or manipulator in the simulation change.

Following this, the center of mass is calculated using equation (2.30) of [79], see Figure 5.44. Next, the rotation matrices for each propeller and link of the manipulator are defined, as in Chapter 2 of [71], see 5.45.

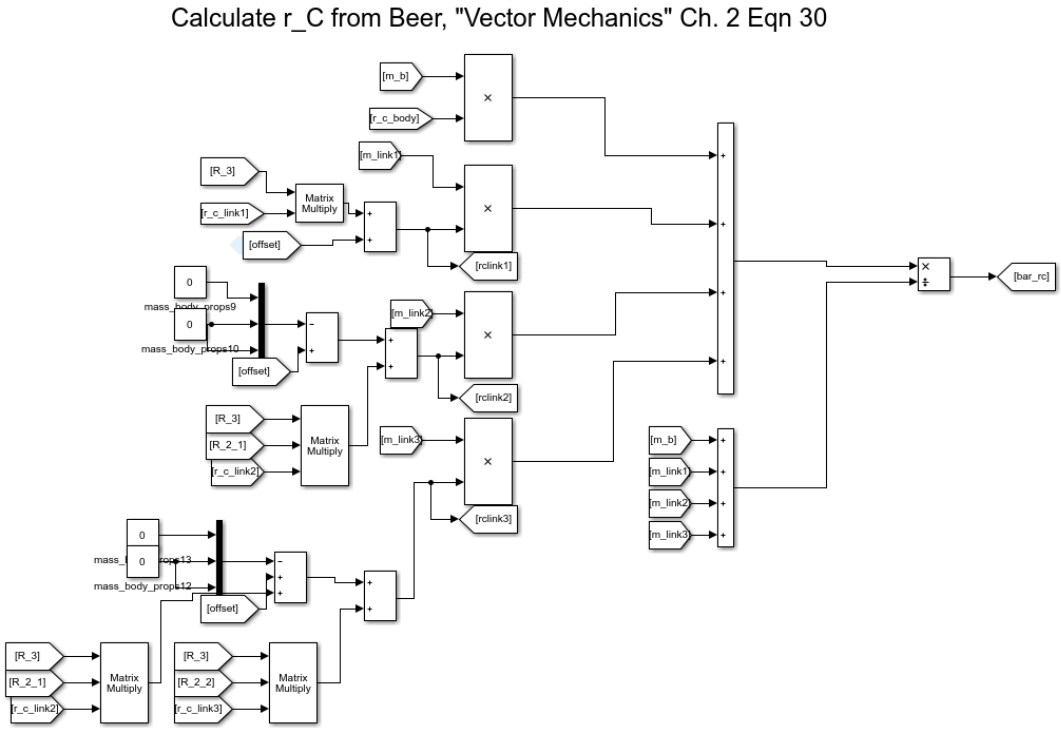

Figure 5.44: Calculation of the system center of mass

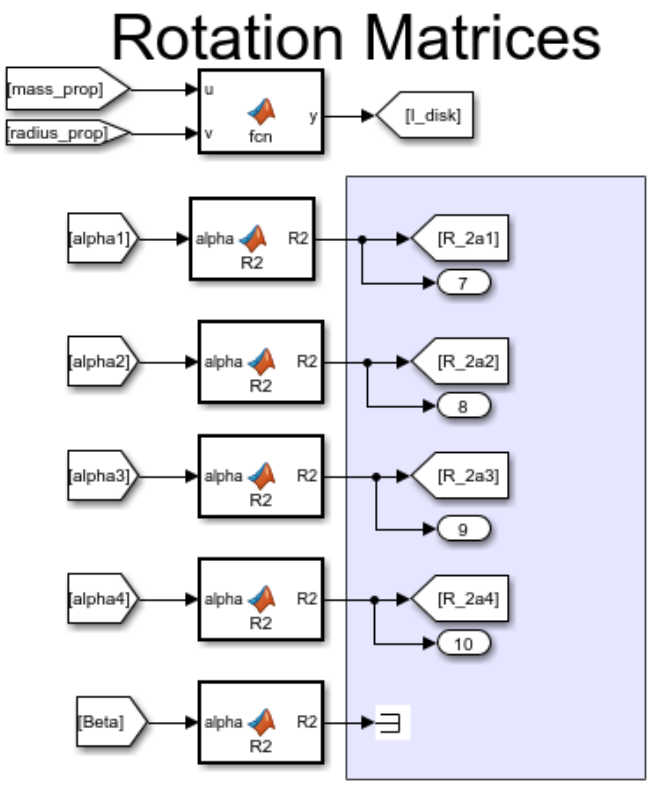

Figure 5.45: Formulation of the rotation matrices

Then, the center of mass properties and rotation matrices are used to find the total inertia for the payload and the overall system, using the same methodology as in [80, p. 167], see Figure 5.46.

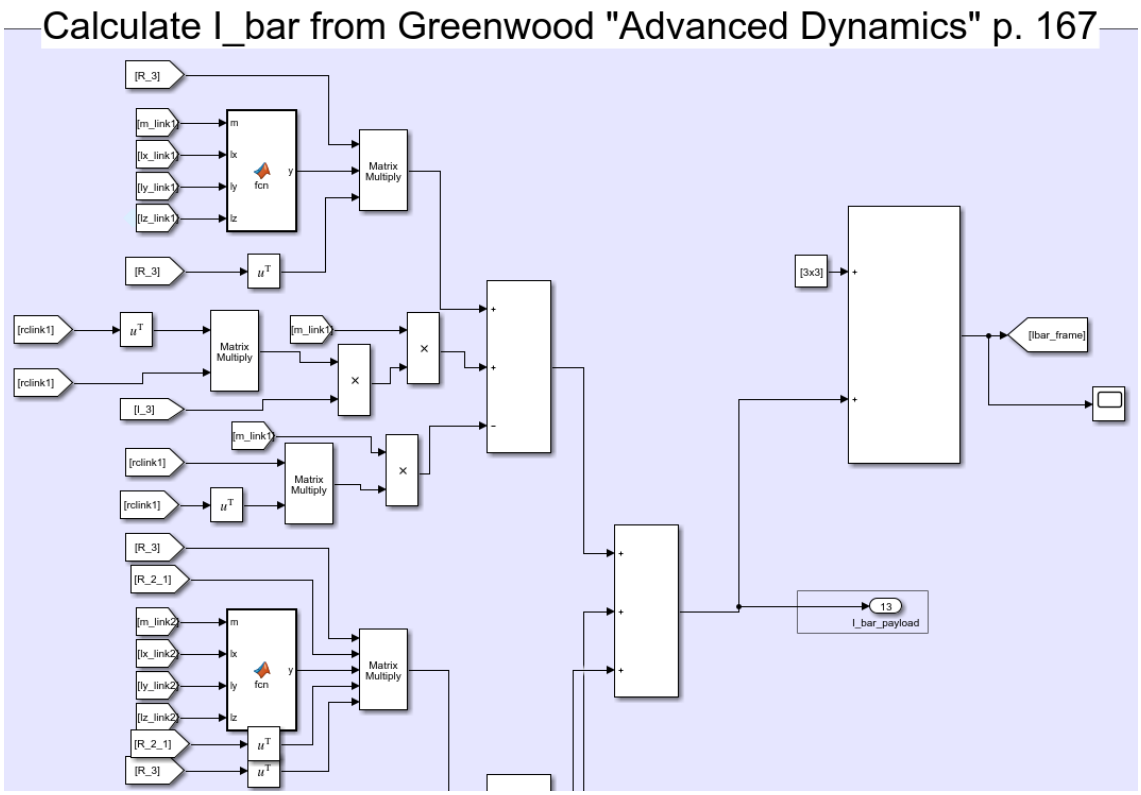

Figure 5.46: Calculation of the body and manipulator inertia

Following from this, the individual propeller inertias are calculated as in [80, p. 167], see Figure 5.47.

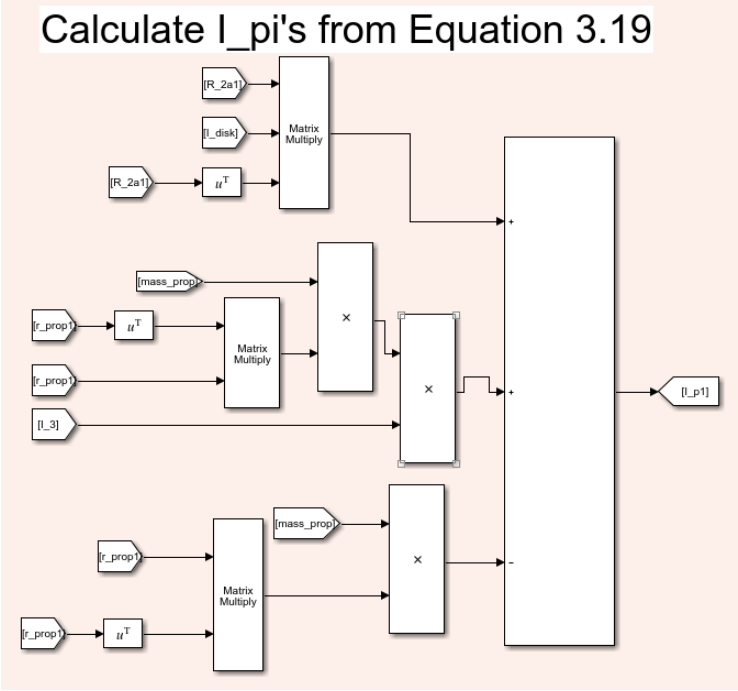

Figure 5.47: Calculation of propeller inertia

Finally, the overall inertia matrix is calculated using the block shown in Figure 5.48

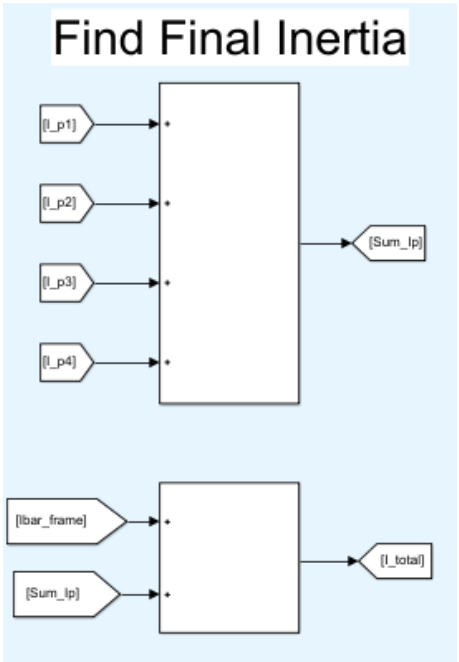

Figure 5.48: Calculation of the total system inertia

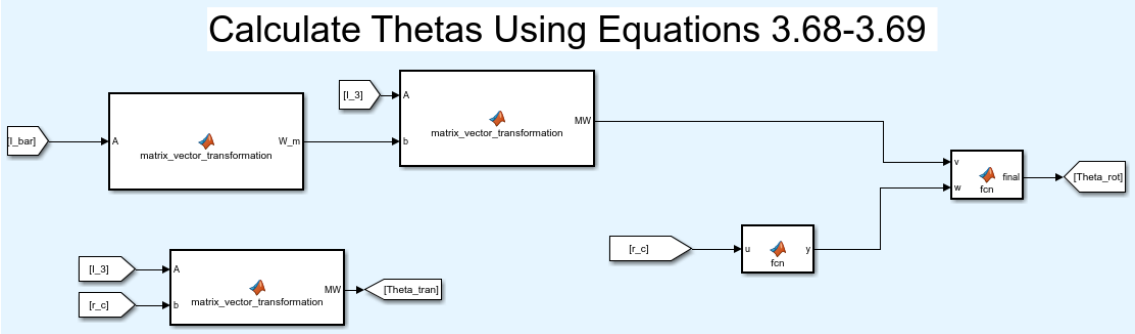

Figure 5.49: Calculation of Θ

and passed to the `Regressor Vector` subsystem. This subsystem takes as inputs the r_C, I, and current state values from the simulator, and uses them to calculate Θ according to (3.68)-(3.69) and Φ according to (3.70)-(3.71). Note that Θ is not used explicitly, but is needed to perform the simulation. These values are all output back into the `Calculate Rc, I, Wp, Wpi_dot` block, where `Calculate I_dot` uses the Simulink derivative function to take the derivative of the calculated I values, `Calculate Wpi` uses the tilt angle and angular velocity of each propeller to calculate

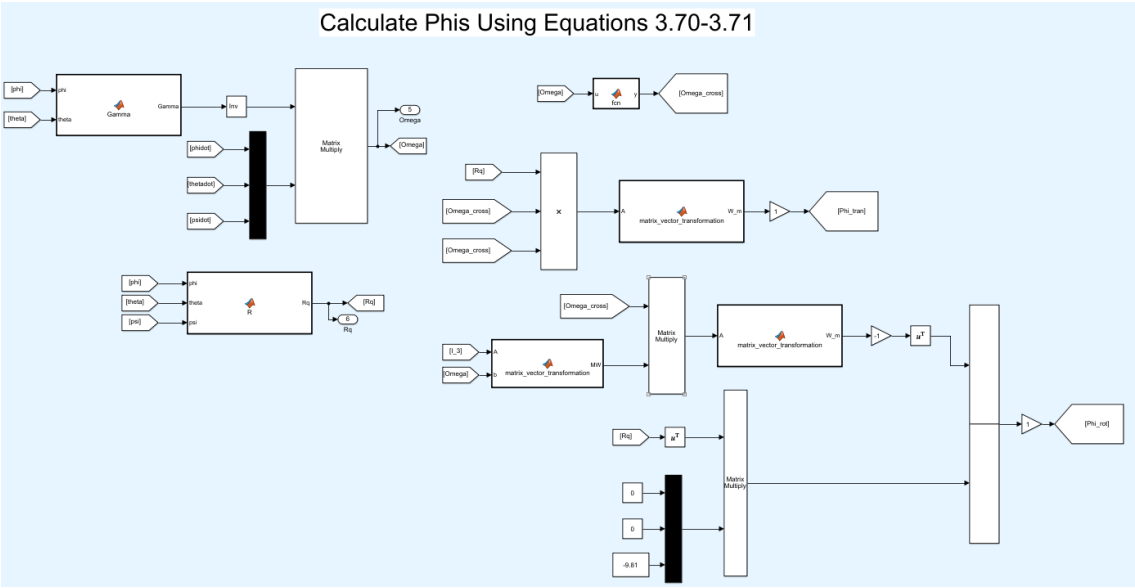

Figure 5.50: Calculation of Φ

ω_{P_i}, and `Calculate Wpi_dot` takes the derivative of ω_{P_i}. This is then combined into the new state vector and sent back through the simulation to calculate the next timestep.

5.7. Results

While the simulation is running, all state data captured is passed into the `Log Data` subsystem, where it is stored in the Matlab workspace as well as output to several scopes so that the data results can be tracked in real time. At the top of the main simulator screen, there are eleven black function blocks arranged in three columns. double-clicking any of these blocks will output the associated plot with its label, allowing the user to examine the results of the simulation. For the aerial manipulation system presented in this thesis, the desired trajectory was one full circle with a radius of 2 m, a hovering state during the manipulator motion, and another full circle with the manipulator in its new position. The reference trajectory and the UAV's actual trajectory are shown in Figure 5.51.

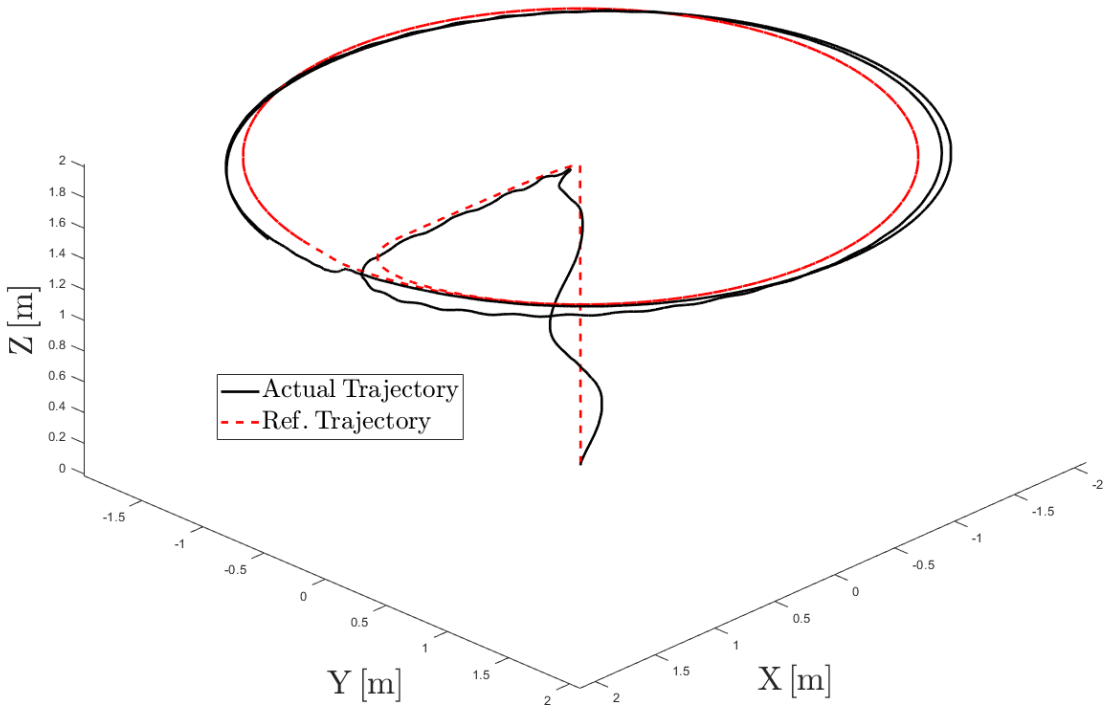

Figure 5.51: 3D Plot of aerial manipulator trajectory

It is apparent that the actual trajectory clearly follows the reference trajectory despite the poorly modeled rotation of the robotic manipulator. The effect of the manipulator as a disturbance can be seen during takeoff, where the adaptive control law is still attempting to account for the arm's presence.

The normalized control inputs are shown in Figure 5.32, where it can be seen how the pitch control input was especially active to compensate for the position of the arm as it was moving, and that the yaw control input had to compensate for the aerodynamic forces. Finally, it can be seen that the error function $h(\cdot, \cdot)$ stayed within the user-defined range, as shown in Figure 5.53.

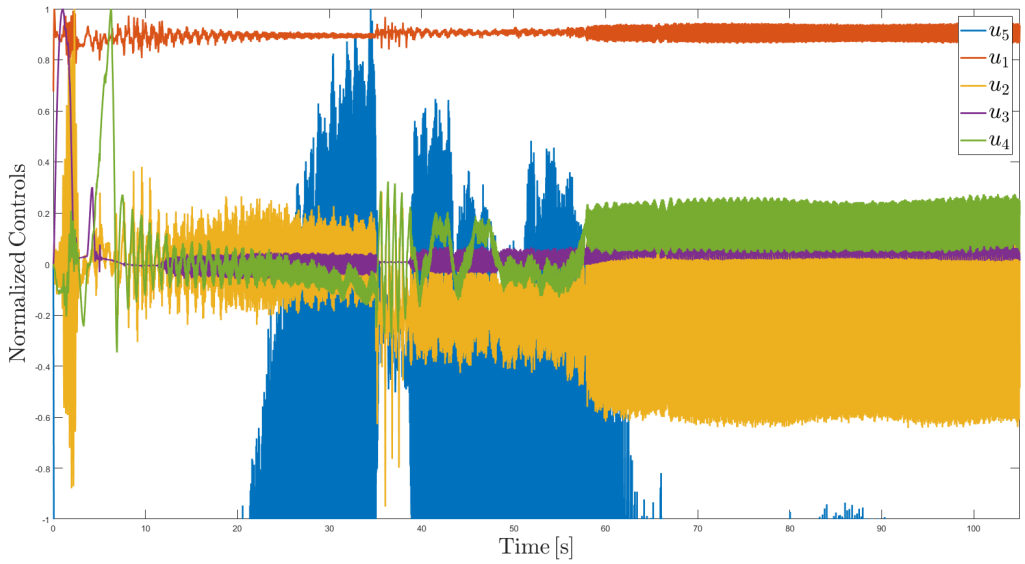

Figure 5.52: Normalized vehicle control inputs

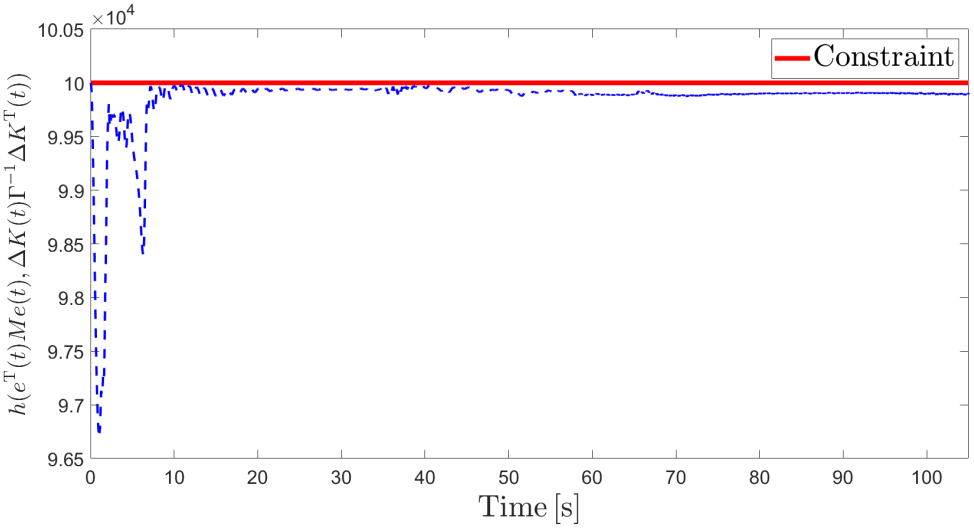

Figure 5.53: Constriant Plot

6. Conclusion

This thesis provided a literature review on the state of aerial manipulation, a new robust MRAC algorithm which has been applied to control a tiltrotor quadcopter equipped with a robotic manipulator, a brief discussion on the two-link manipulator used, and a walkthrough of the numerical simulator used to validate the theoretical results. To enhance the robustness of the MRAC law, the tiltrotor's equations of motion were derived and analyzed. The key result of the new MRAC algorithm was that the control law keeps the trajectory tracking and adaptive gain errors within some user-defined constraint. This result was demonstrated through a numerical simulation with several added disturbances to highlight the efficacy of the proposed robust MRAC law.

In this thesis, the robotic manipulator installed on the tiltrotor was considered as a poorly modeled disturbance. In order to deduce the effect of the robotic manipulator on the aerial platform, the manipulator's inverse kinematics were derived. Next, the manipulator's trajectory as outlined by using quintic polynomials. Finally, a PID control law was implemented to actuate the robotic manipulator.

The last result of this thesis was the combination of the tiltrotor and robotic manipulator into one aerial manipulation system. The combined CAD model was constructed, then imported into the Simulink environment. The MRAC control algorithm previously developed was implemented to control the tiltrotor, and the PID control used to control the manipulator. Then, a step by step walkthrough

of the simulation was presented, highlighting the implementation of the previously developed MRAC algorithm, and providing a user manual for future work on the simulation. Finally, the results of the simulation are presented, to demonstrate how the control algorithm responds to the greater disturbance of the manipulator.

It is this simulation that offers the best potential for future research. The steps to modify and run this simulation are all present in this thesis, allowing for future users to implement new manipulators, trajectories, mission tasks, and so on. This thesis highlights a general case of use for this simulation, but its real potential lies in providing an flight test platform for aerial manipulators, allowing for quickly repeatable flights with simple to change tuning parameters, with the additional benefit of crashing the vehicle in the simulation does not damage the vehicle or manipulator. This allows for wide variety of experimentation within the simulation, in order to gain valuable knowledge about a system's performance before performing physical flight tests. As for future work, the implementation of the control algorithm on a tiltrotor carrying a robotic manipulator would be the next logical step in continuing the results presented here.

A. Appendix 1

This write-up serves as an introductory guide into the importing of a Solidworks model into Simulink, as well as an explanation of how to manipulate the model once it is in Simulink. The methods discussed are valid with Solidworks 16-17, 17-18, and 18-19, as well as all Matlab versions from Matlab 2016a-2018a. This write up serves as a general guide which will work for all software versions listed. It is assumed that the user has Simulink, Simmechanics, added to their Matlab License.

Once the user has verified the prerequisite software is in place, one more program will need to be downloaded. This tool is the Simscape Multibody Link, for which can be downloaded from `https://www.mathworks.com/products/simmechanics/download_smlink.html`, and for which the user will need to select the correct package for their Solidworks version. Once the provided zip folder and `install_addon.m` are downloaded from the website, they need to be placed in the currently active Matlab folder or added to the current Matlab path, then from the Matlab command line, run the command `install_addon('zip_file_name.zip')`. If this process shows the user an error, ensure that the downloaded files are on the Matlab path by navigating to the folder they are placed in on the left side of the Matlab screen, right clicking

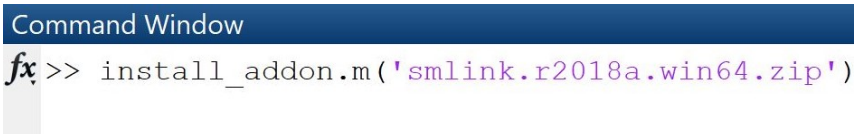

Figure A.1: Installing the Simmechanics Link from the Command Line

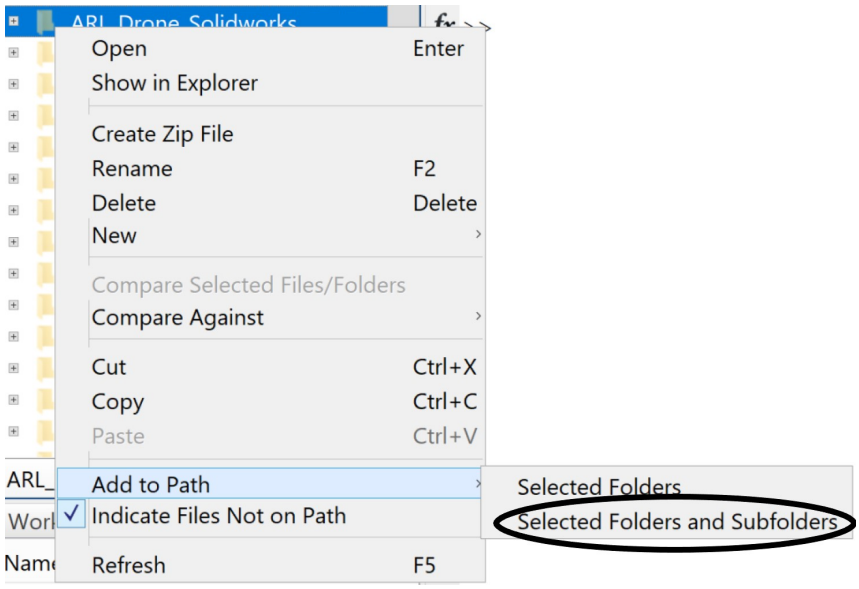

Figure A.2: Adding a Folder to the Current Matlab Path

on the folder, and selecting `Add to Path > Selected Folders and Sub-folders` as in Figure A.2, and re-attempt to install the .zip file.

Once this process is completed, restart Matlab in administrator mode, and run the command `regmatlabserver` from the command line, as shown in Figure A.3., which will set up a registered Matlab server on the user's computer in order to communicate between Solidworks and Matlab.

The last set-up step to be done performed in Matlab is to link Solidworks and Simulink, done by executing the command `smlink_linksw` from the Matlab command line, as in Figure A.4. Once this process is complete, the user should close

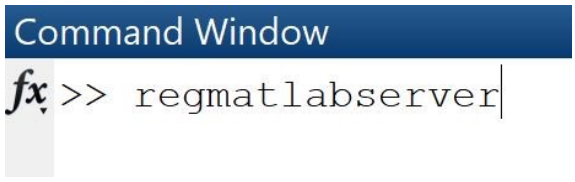

Figure A.3: Registering the Matlab Server from the Command Line

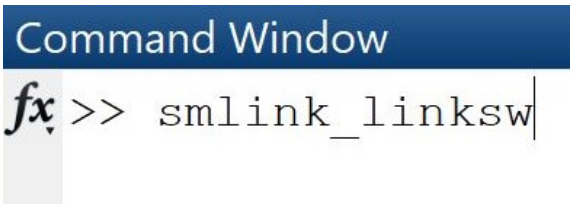

Figure A.4: Linking Solidworks and Simulink from the Matlab Command Line

Matlab and proceed to open up their Solidworks installation.

In Solidworks, open the CAD assembly to export, then navigate to `Tools` > `Add-ins` in the top menu, as demonstrated in Figure A.5, and once this is selected, a new window should pop up with the list of user add-ins to Solidworks, as shown in Figure A.6.

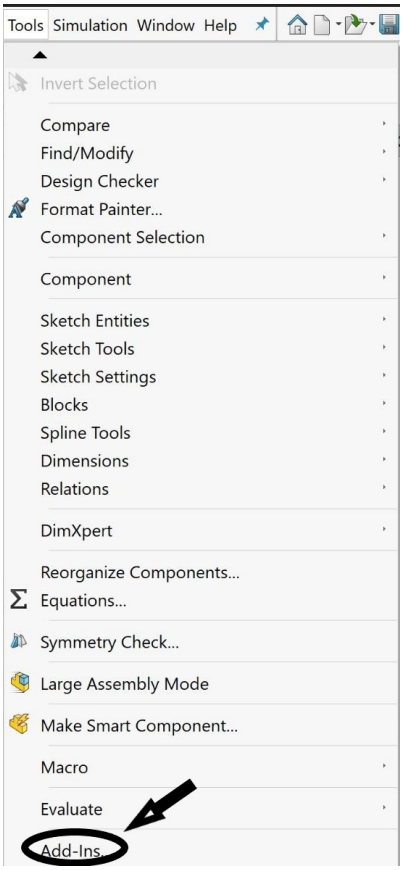

Figure A.5: Finding the Add-ins Menu in Solidworks

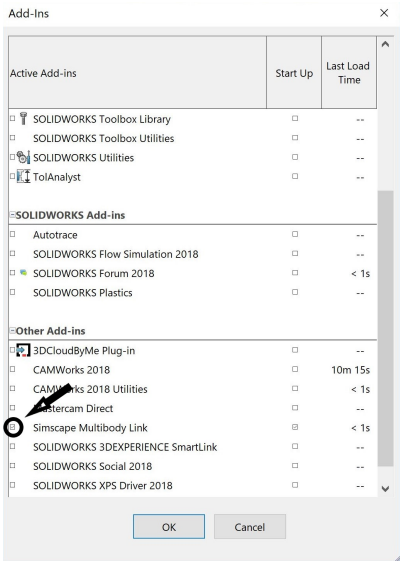

Figure A.6: Enabling Simscape Multibody Link in Solidworks

The user should ensure that the box to the left of Simscape Multibody Link is checked, as in Figure A.6. This enables the Simscape Multibody Link, and once this is complete, navigate once again to the toolbar, select `Tools` > `Simscape Multibody Link` > `Export` > `Simscape Multibody`, as illustrated in Figure A.7. Once this is complete, the model should export to an .xml file, and the individual Solidworks part files (.sldprt) should be copied as individual .STEP files. Make sure that all of these .Step files are kept in the same folder as the .xml file, in order to

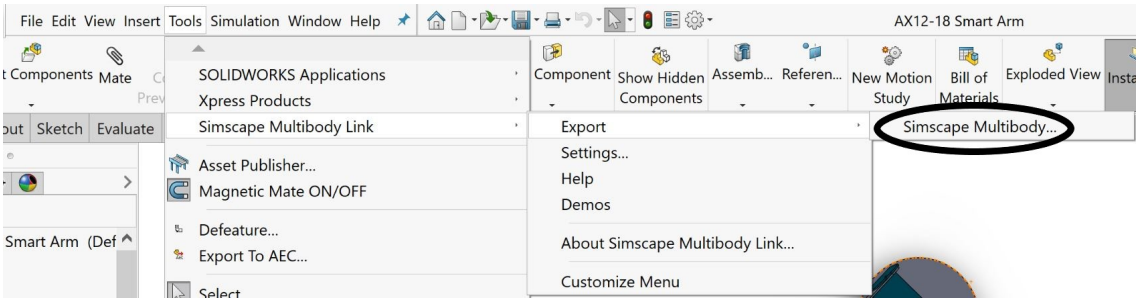

Figure A.7: Linking Solidworks and Simulink from the Command Line

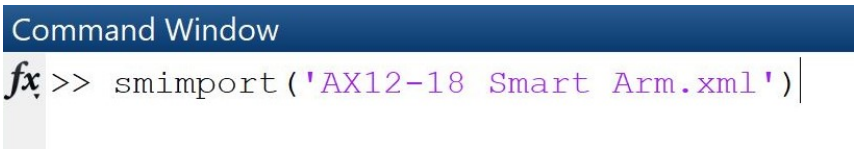

Figure A.8: Importing the .xml file to Simulink

allow the model visuals to render in Simulink. After this process is complete, close down Solidworks, and re-open Matlab.

Once Matlab is open, ensure all the .xml and .step files are added to the Matlab path, then run `smimport('your_file_name.xml')` from the command line, as demonstrated in Figure A.8, which should automatically create a .mdl or .slx Simulink model of the assembly, as well as a .m data file containing all of the physical properties of each part of the system. Look over the newly created Simulink model to ensure all blocks have at least one connection, and that there are no unnconnected lines in the model. Then, run the model, and once it compiles a visual should be created in the main Matlab window, as in Figure A.9. From this screen, visually inspect that all parts appear to be present in the model. If there is a missing

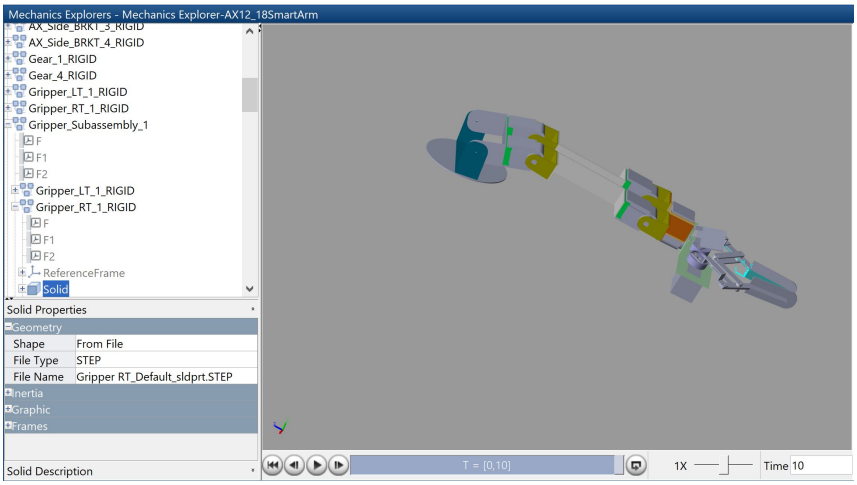

Figure A.9: Observing the Simulink Model

visual, ensure that the corresponding .step file is contained in the simulation folder. If the visual is still not present, the best course of action is to repeat the entire export from Solidworks and import to Simulink steps.

Finally, a couple of notes on how to modify parameters or values, especially useful when combining separately imported assemblies. First, each Simulink model can only reference one .m data files, so the user will need to combine data files if combining Simulink models. However, care should be taken to ensure that the reference numbers to all the parts in one of the .m files is changed, to avoid repeat parts. To find the reference number, navigate to the .m data file on the left side

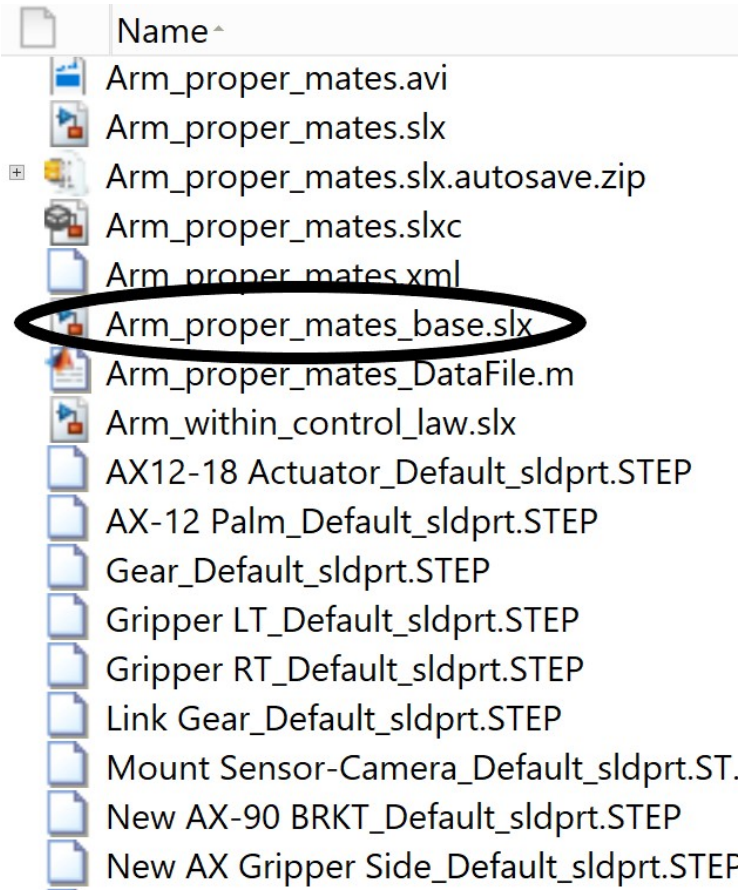

Figure A.10: Opening the Simulation Data File

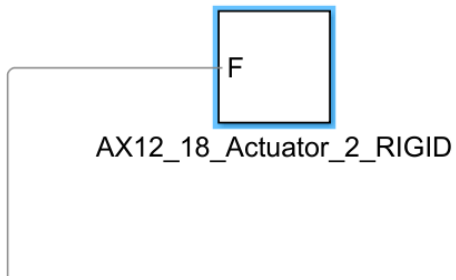

Figure A.11: Simulink Compoment Block

of the main Matlab view, then double-click on the appropriate data file where the user can then navigate the list of parts, with the reference number being displayed as `smiData.Solid(n)` with `n` being the part reference number. To find a specific part's reference number, open the Simulink model from the sidebar, circled in Figure A.10, then double-click on the desired component block within the model, shown in Figure A.11, and once inside select the solid block from the system, circled in Figure A.12, which results in the display where the user can find the corresponding part reference number, shown in Figure A.13.

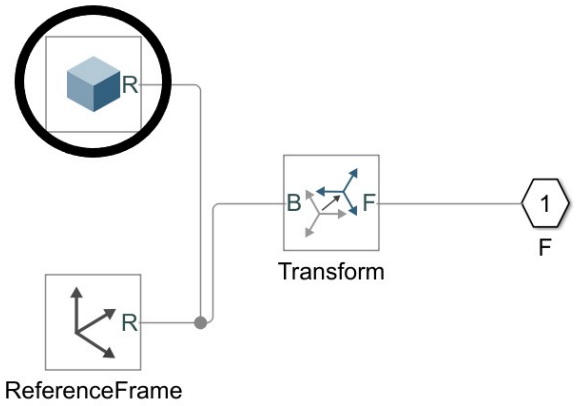

Figure A.12: Selecting the Simulink Solid Block

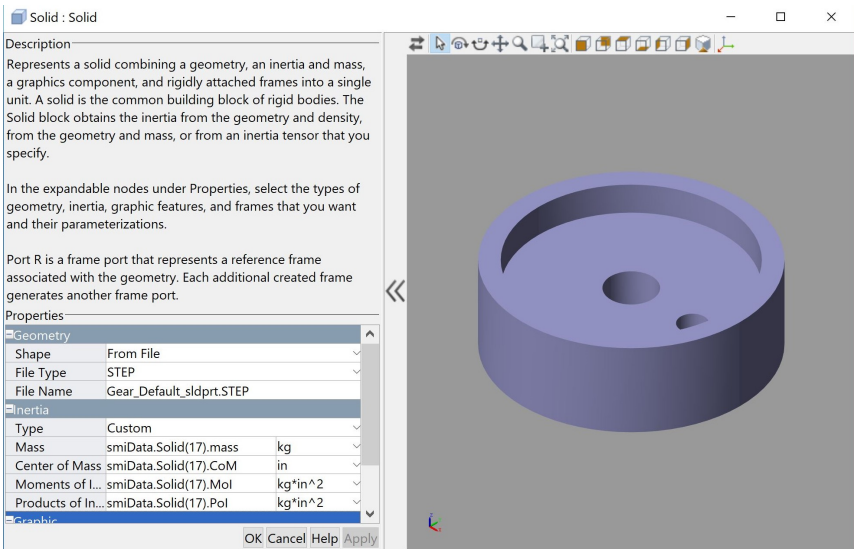

Figure A.13: Simulink Solid Mass, Inertia, and Visual Properties

Next, going back to the data file in Matlab, the user can scroll down to the desired part, where the properties will be displayed as in Figure A.14.

If any of the mass or inertia properties need to be changed, this can be done by editing the corresponding values and re-initializing the model workspace. To reinitialize the model workspace, go into the Simulink model, then select `View > Model Explorer > Model Workspace` as shown in Figure A.15. then select the `Browse` option on the right side of the screen then from the file list that pops up, select the Matlab file containing the model data. Once this is selected, select `Reinitialize from Source` and then close the `Model Explorer` window. These steps are illus-

```
%Inertia Type - Custom
%Visual Properties - Simple
smiData.Solid(17).mass = 0.0043601193663677209;  % kg
smiData.Solid(17).CoM = [-0.003347385316446406 0 0.14857262446152003];  % in
smiData.Solid(17).MoI = [0.00042131655636395597 0.00041667027642013196 0.00076315466696509 93];  % kg*in^2
smiData.Solid(17).PoI = [0 -3.4404244328873878e-07 0];  % kg*in^2
smiData.Solid(17).color = [0.75294117647058822 0.75294117647058822 1];
smiData.Solid(17).opacity = 1;
smiData.Solid(17).ID = 'Gear*:*Default';
```

Figure A.14: Solid Block Properties Menu

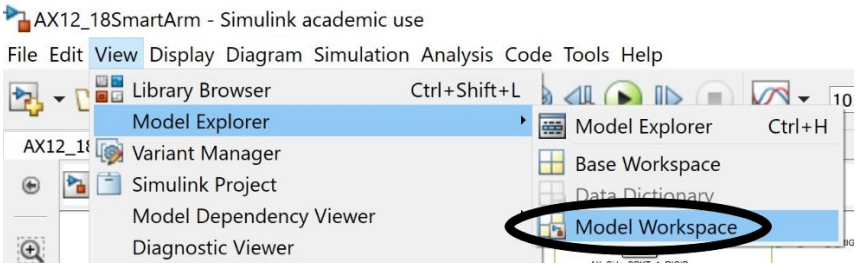

Figure A.15: Opening the Model Workspace

trated in Figures A.15-A.17. Running the simulation will apply the user's changes. In addition, re-initializing the model workspace will fix any "Model Workspace not Found" errors that might arise.

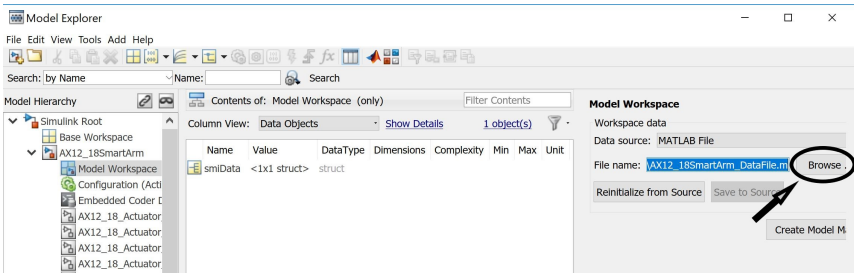

Figure A.16: Finding the Correct Data File for the Model Workspace

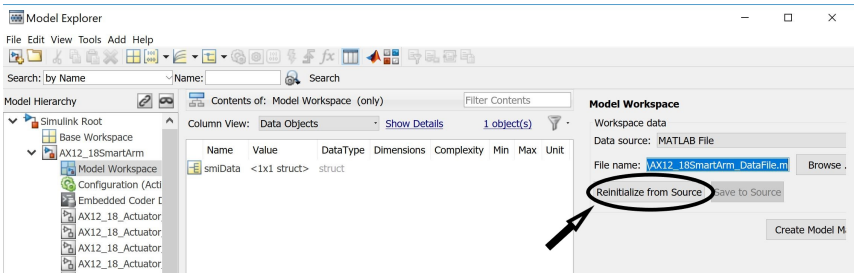

Figure A.17: Re-initializing the Model Workspace

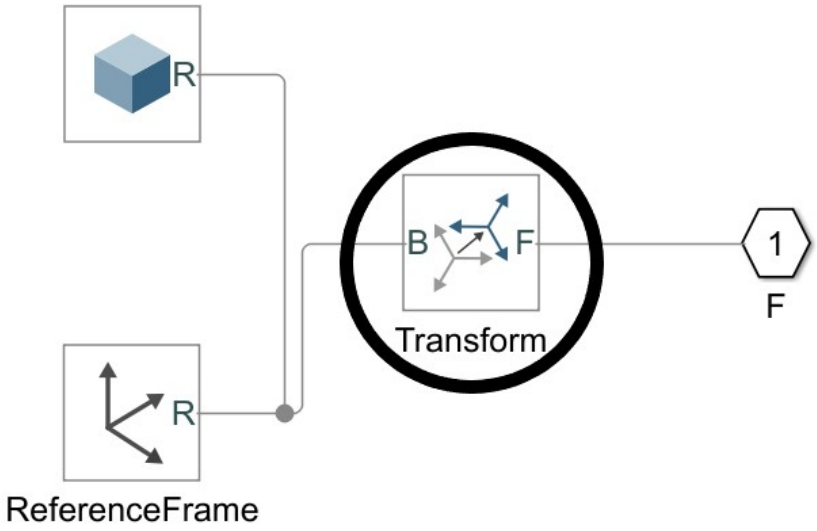

Figure A.18: Selecting the Transform Block

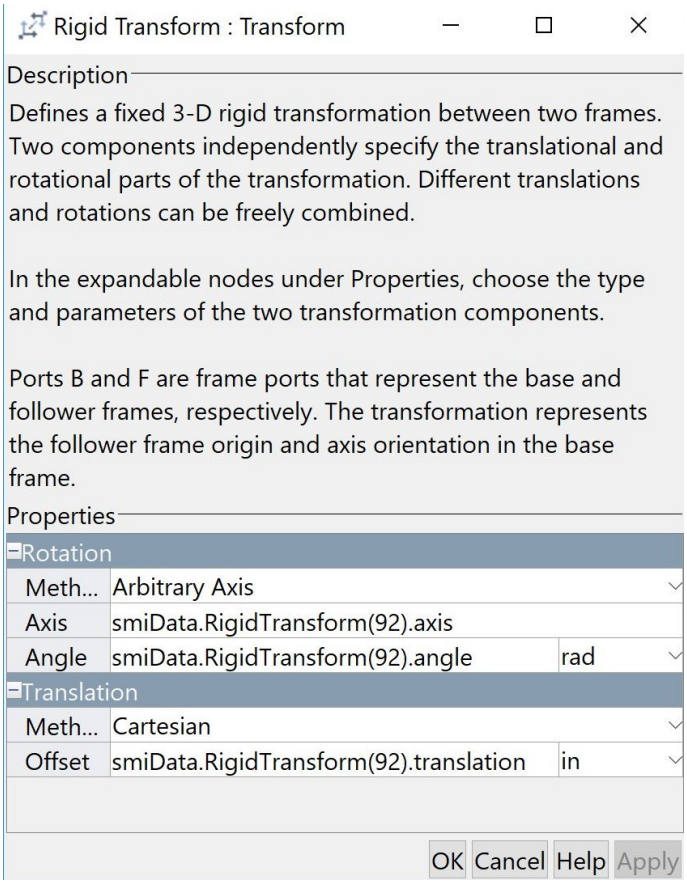

Figure A.19: Finding the Transform Reference Number

Another potential error when combining models is misaligned parts. To fix this, go to the transform block in Simulink immediately preceding the misaligned part, circled in Figure A.18, where the transform reference number can be found by double-clicking it to open the pop-up shown in figure A.19. To find the properties of this transformation from the data file, scroll down to its listing in the file and edit the parameters to their desired value, where the properties are shown in Figure A.20. The editable properties of the transformation block are the `Rigid Transform translation`, which designates the offset of the current part's reference frame from the previous part's reference frame through a translation, the `Rigid Transform angle`, which defines the rotation of the reference frames, and the `Rigid Transform axis` defines the axes around which the rotation occurs. If a part is misaligned, these values can be adjusted to fix the misalignment, however, it is recommended that the user save a copy of the data file before undertaking the task. Furthermore, this process can be fairly complex, and for large discrepancies between the actual model state and the desired model state, it will be much faster for the user to modify the Solidworks CAD file to reflect the desired changes, then repeat the steps above to import it into Simulink. By running the simulation and then selecting the desired transformation from the parts list on the left of the screen, the user can see the effect of the transformation on the reference frames, circled in Figure A.21.

```
%Translation Method - Cartesian
%Rotation Method - Arbitrary Axis
smiData.RigidTransform(92).translation = [-0.73425000000000029 1.0599999999999947 -0.12499999999999978];  % in
smiData.RigidTransform(92).angle = 2.0943951023931953;  % rad
smiData.RigidTransform(92).axis = [0.57735026918962595 0.57735026918962584 0.57735026918962562];
smiData.RigidTransform(92).ID = 'F[AX-12 Palm-1:-:Mount Sensor-Camera-1]';
```

Figure A.20: Transform Properties in Data File

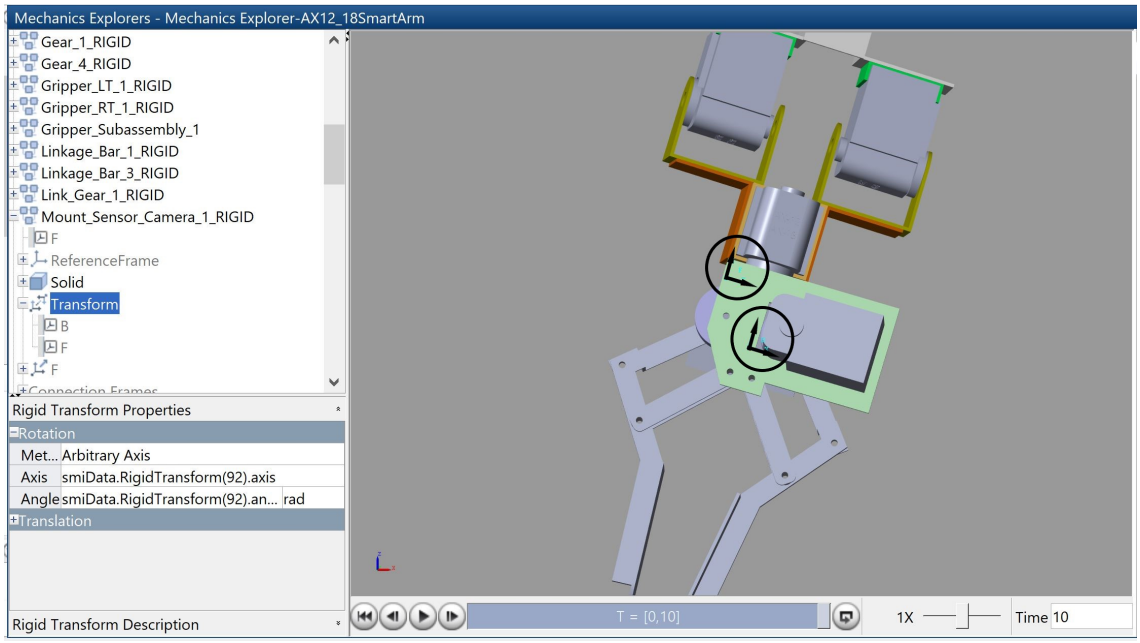

Figure A.21: Visualization of the Transformation Operation

This should cover all the basic steps required to create and adjust new and existing Simulink visual models. These techniques will cover almost all situations that arise during the import of CAD models to Simulink Models, and any future user who finds new issues is encouraged to document the problem and solution to pass along to future users.

Bibliography

[1] A. L'Afflitto and K. Mohammadi, "Equations of motion of rotary-wing unmanned aerial systems with time-varying inertial properties," *AIAA Journal of Guidance, Control, and Dynamics*, 2017.

[2] S. Bouabdallah, P. Murrieri, and R. Siegwart, "Design and control of an indoor micro quadrotor," in *IEEE International Conference on Robotics and Automation. Proceedings.*, vol. 5, 2004, pp. 4393–4398.

[3] K. Li, K. Wang, K. Zhang, and B. M. Chen, "Aggressive maneuvers of a quadrotor mav based on composite nonlinear feedback control," in *IEEE International Conference on Advanced Intelligent Mechatronics*, 2016, pp. 513–518.

[4] T. Hamel, R. Mahoney, R. Lozano, and J. Ostrowski, "Dynamic modelling and configuration stabilization for an X4-flyer," in *International Federation of Automatic Control World Congress*, 2002, pp. 1–6.

[5] D. Gheorghiţă, I. Vîntu, L. Mirea, and C. Brăescu, "Quadcopter control system," in *19th International Conference on System Theory, Control and Computing*, 2015, pp. 421–426.

[6] I. Sa and P. Corke, "System identification, estimation and control for a cost effective open-source quadcopter," in *IEEE International Conference on Robotics and Automation*, 2012, pp. 2202–2209.

[7] B. T. M. Leong, S. M. Low, and M. P.-L. Ooi, "Low-cost microcontroller-based hover control design of a quadcopter," *Procedia Engineering*, vol. 41, pp. 458 – 464, 2012, international Symposium on Robotics and Intelligent Sensors.

[8] L. M. Argentim, W. C. Rezende, P. E. Santos, and R. A. Aguiar, "PID, LQR and LQR-PID on a quadcopter platform," in *International Conference on Informatics, Electronics and Vision*, 2013, pp. 1–6.

[9] K. Patel and J. Barve, "Modeling, simulation and control study for the quadcopter UAV," in *9th International Conference on Industrial and Information Systems*, 2014, pp. 1–6.

[10] P. Wang, Z. Man, Z. Cao, J. Zheng, and Y. Zhao, "Dynamics modelling and linear control of quadcopter," in *International Conference on Advanced Mechatronic Systems*, 2016, pp. 498–503.

[11] Y. M. Al-Younes, M. A. Al-Jarrah, and A. A. Jhemi, "Linear vs. nonlinear control techniques for a quadrotor vehicle," in *International Symposium on Mechatronics and its Applications*, 2010, pp. 1–10.

[12] S. Bouabdallah and R. Siegwart, "Backstepping and sliding-mode techniques applied to an indoor micro quadrotor," in *IEEE International Conference on Robotics and Automation*, 2005, pp. 2247–2252.

[13] T. Madani and A. Benallegue, "Sliding mode observer and backstepping control for a quadrotor unmanned aerial vehicles," in *American Control Conference*, 2007, pp. 5887–5892.

[14] B. Sumantri, N. Uchiyama, S. Sano, and Y. Kawabata, "Robust tracking control of a quad-rotor helicopter utilizing sliding mode control with a nonlinear sliding surface," *Journal of System Design and Dynamics*, vol. 7, no. 2, pp. 226–241, 2013.

[15] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2002.

[16] B. Sumantri, N. Uchiyama, and S. Sano, "Second order sliding mode control for a quad-rotor helicopter with a nonlinear sliding surface," in *IEEE Conference on Control Applications*, 2014, pp. 742–746.

[17] S. Islam, P. Liu, and A. El Saddik, "Adaptive sliding mode control of unmanned four rotor flying vehicle," *International Journal of Robotics and Automation*, vol. 30, 2015.

[18] A. L'Afflitto, R. B. Anderson, and K. Mohammadi, "An introduction to nonlinear robust control for unmanned quadrotor aircraft: How to design control algorithms for quadrotors using sliding mode control and adaptive control techniques [focus on education]," *IEEE Control Systems Magazine*, vol. 38, no. 3, pp. 102–121, 2018.

[19] B. Zhao, B. Xian, Y. Zhang, and X. Zhang, "Nonlinear robust adaptive tracking control of a quadrotor UAV via immersion and invariance methodology," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 5, pp. 2891–2902, 2015.

[20] A. Nemati, N. Soni, M. Sarim, and M. Kumar, "Design, fabrication and control

of a tilt rotor quadcopter," in *ASME Dynamic Systems and Control Conference*, 2016.

[21] S. Yoon, H. Lee, and T. Pulliam, "Computational analysis of multi-rotor flows," in *AIAA SciTech Forum*, 2016.

[22] D. Invernizzi and M. Lovera, "Geometric tracking control of a quadcopter tiltrotor UAV," in *International Federation of Automatic Control*, 2016.

[23] A. Nemati and M. Kumar, "Modeling and control of a single axis tilting quadcopter," in *American Control Conference*, 2014, pp. 3077–3082.

[24] S. Sridhar, R. Kumar, M. Radmanesh, and M. Kumar, "Non-linear sliding mode control of a tilting-rotor quadcopter," in *ASME Dynamic Systems and Control Conference*, 2017.

[25] A. Alkamachi and E. Erçelebi, "A proportional derivative sliding mode control for an overactuated quadcopter," *Journal of Aerospace Engineering*, 2018.

[26] R. Anderson, J. Burke, J. Marshal, and A. L'Afflitto, "A robust adaptive control for constrained tilt-rotor quadcopters of unknown inertial properties," Accepted for publication at the IEEE American Control Conference, 2019.

[27] ——, "A robust adaptive control for constrained tilt-rotor quadcopters of unknown inertial properties," *IEEE Transactions on Control Systems Technology*, To appear, 2019.

[28] M. W. Spong, M. Vidyasagar, and S. Hutchinson, *Robot dynamics and control*. Hoboken, NJ: Wiley, 2004.

[29] S. Oh and K. Kong, "Two-degree-of-freedom control of a two-link manipulator in the rotating coordinate system," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 9, pp. 5598–5607, 2015.

[30] S. Kim, S. Choi, and H. J. Kim, "Aerial manipulation using a quadrotor with a two DOF robotic arm," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 4990–4995.

[31] L. Everett, M. Driels, and B. Mooring, "Kinematic modelling for robot calibration," in *IEEE International Conference on Robotics and Automation*, vol. 4, 1987, pp. 183–189.

[32] A. De Luca and B. Siciliano, "Closed-form dynamic model of planar multilink lightweight robots," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 4, pp. 826–839, 1991.

[33] S. Dubowsky and D. DesForges, "The application of model-referenced adaptive control to robotic manipulators," *Journal of Dynamical Systems, Measurement, and Control*, vol. 101, no. 3, pp. 193–200, 1979.

[34] J.-J. E. Slotine and W. Li, "On the adaptive control of robot manipulators," *The International Journal of Robotics Research*, vol. 6, no. 3, pp. 49–59, 1987.

[35] P. K. Khosla and T. Kanade, "Parameter identification of robot dynamics," in *IEEE Conference on Decision and Control*, 1985, pp. 1754–1760.

[36] Y. Huang and C. S. G. Lee, "Generalization of newton-euler formulation of dynamic equations to nonrigid manipulators," in *American Control Conference*, 1987, pp. 72–77.

[37] F. Boyer and P. Coiffet, "Generalization of newton-euler model for flexible manipulators," *Journal of Robotic Systems*, vol. 13, no. 1, pp. 11–24, 1996.

[38] S. Singh and M. Leu, "Optimal trajectory generation for robotic manipulators using dynamic programming," *Journal of Dynamic Systems, Measurement, and Control*, vol. 10, no. 2, pp. 88–96, 1987.

[39] K. Shin and N. McKay, "Minimum-time control of robotic manipulators with geometric path constraints," *IEEE Transactions on Automatic Control*, vol. 30, no. 6, pp. 531–541, 1985.

[40] D. P. Garg and M. Kumar, "Optimization techniques applied to multiple manipulators for path planning and torque minimization," *Engineering Applications of Artificial Intelligence*, vol. 15, no. 3, pp. 241 – 252, 2002.

[41] S. Macfarlane and E. A. Croft, "Jerk-bounded manipulator trajectory planning: design for real-time applications," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 1, pp. 42–52, 2003.

[42] D. Constantinescu and E. A. Croft, "Smooth and time-optimal trajectory planning for industrial manipulators along specified paths," *Journal of Robotic Systems*, vol. 17, no. 5, pp. 233–249, 2000.

[43] R. L. Andersson, "Aggressive trajectory generator for a robot ping-pong player," *IEEE Control Systems Magazine*, vol. 9, no. 2, pp. 15–21, 1989.

[44] E. M. Jafarov, M. N. A. Parlakci, and Y. Istefanopulos, "A new variable structure PID-controller design for robot manipulators," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 1, pp. 122–130, 2005.

[45] J. Baek, M. Jin, and S. Han, "A new adaptive sliding-mode control scheme for application to robot manipulators," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 6, pp. 3628–3637, 2016.

[46] J. Alvarez-Ramirez, R. Kelly, and I. Cervantes, "Semiglobal stability of saturated linear PID control for robot manipulators," *Automatica*, vol. 39, no. 6, pp. 989 – 995, 2003.

[47] R. Volpe and P. Khosla, "A theoretical and experimental investigation of explicit force control strategies for manipulators," *IEEE Transactions on Automatic Control*, vol. 38, no. 11, pp. 1634–1650, 1993.

[48] A. Ferrara and L. Magnani, "Motion control of rigid robot manipulators via first and second order sliding modes," *Journal of Intelligent and Robotic Systems*, vol. 48, no. 1, pp. 23–36, 2007.

[49] C. Su, T. Leung, and Y. Stepanenko, "Real-time implementation of regressor-based sliding mode control algorithm for robotic manipulators," *IEEE Transactions on Industrial Electronics*, vol. 40, no. 1, pp. 71–79, 1993.

[50] S. S. Ge, T. H. Lee, and C. J. Harris, *Adaptive Neural Network Control of Robotic Manipulators*. River Edge, New Jersey: World Scientific, 1998, vol. 19.

[51] P. G. Keleher and R. J. Stonier, "Adaptive terminal sliding mode control of a rigid robotic manipulator with uncertain dynamics incorporating constraint inequalities," *Australia and New Zealand Industrial and Applied Mathematics Journal*, vol. 43, no. 0, 2002.

[52] B. Yao and M. Tomizuka, "Smooth robust adaptive sliding mode control of manipulators with guaranteed transient performance," in *American Control Conference*, vol. 1, 1994, pp. 1176–1180.

[53] M. Zeinali and L. Notash, "Adaptive sliding mode control with uncertainty estimator for robot manipulators," *Mechanism and Machine Theory*, vol. 45, no. 1, pp. 80 – 90, 2010.

[54] W. He, Y. Chen, and Z. Yin, "Adaptive neural network control of an uncertain robot with full-state constraints," *IEEE Transactions on Cybernetics*, vol. 46, no. 3, pp. 620–629, 2016.

[55] C. Sun, W. He, and J. Hong, "Neural network control of a flexible robotic manipulator using the lumped spring-mass model," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 8, pp. 1863–1874, 2017.

[56] K. Lochan and B. K. Roy, "Control of two-link 2-DOF robot manipulator using fuzzy logic techniques," in *International Conference on Soft Computing for*

Problem Solving, K. N. Das, K. Deep, M. Pant, J. C. Bansal, and A. Nagar, Eds. New Delhi: Springer India, 2015, pp. 499–511.

[57] R. Sharma, P. Gaur, and A. Mittal, "Design of two-layered fractional order fuzzy logic controllers applied to robotic manipulator with variable payload," *Applied Soft Computing*, vol. 47, pp. 565 – 576, 2016.

[58] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive Processing*, vol. 12, no. 4, pp. 319–340, 2011.

[59] C. Wang, Y. Zhao, Y. Chen, and M. Tomizuka, "Nonparametric statistical learning control of robot manipulators for trajectory or contour tracking," *Robotics and Computer-Integrated Manufacturing*, vol. 35, pp. 96 – 103, 2015.

[60] S. Kim, S. Choi, and H. J. Kim, "Aerial manipulation using a quadrotor with a two dof robotic arm," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 4990–4995.

[61] M. Orsag, C. Korpela, S. Bogdan, and P. Oh, "Hybrid adaptive control for aerial manipulation," *Journal of Intelligent & Robotic Systems*, vol. 73, no. 4, pp. 693–707, 2014.

[62] V. Lippiello and F. Ruggiero, "Cartesian impedance control of a UAV with a robotic arm," in *International IFAC Symposium on Robot Control*, 2012.

[63] R. Rossi, "Control of an unmanned aerial vehicle equipped with a robotic arm," Ph.D. dissertation, Politecnico di Milano, Milan, Italy, 2016.

[64] F. Huber, K. Kondak, K. Krieger, D. Sommer, M. Schwarzbach, M. Laiacker, I. Kossyk, S. Parusel, S. Haddadin, and A. Albu-Schäffer, "First analysis and experiments in aerial manipulation using fully actuated redundant robot arm," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 3452–3457.

[65] F. Ruggiero, M. A. Trujillo, R. Cano, H. Ascorbe, A. Viguria, C. Peréz, V. Lippiello, A. Ollero, and B. Siciliano, "A multilayer control for multirotor UAVs equipped with a servo robot arm," in *IEEE International Conference on Robotics and Automation*, 2015, pp. 4014–4020.

[66] H. Lee, S. Kim, and H. J. Kim, "Control of an aerial manipulator using on-line parameter estimator for an unknown payload," in *IEEE International Conference on Automation Science and Engineering*, 2015, pp. 316–321.

[67] G. Heredia, A. E. Jimenez-Cano, I. Sanchez, D. Llorente, V. Vega, J. Braga, J. A. Acosta, and A. Ollero, "Control of a multirotor outdoor aerial manipulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 3417–3422.

[68] F. Forte, R. Naldi, A. Macchelli, and L. Marconi, "Impedance control of an aerial manipulator," in *American Control Conference*, 2012, pp. 3839–3844.

[69] H. Nguyen and D. Lee, "Hybrid force/motion control and internal dynamics of quadrotors for tool operation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 3458–3464.

[70] G. Gioioso, M. Ryll, D. Prattichizzo, H. H. Bülthoff, and A. Franchi, "Turning a near-hovering controlled quadrotor into a 3D force effector," in *IEEE International Conference on Robotics and Automation*, 2014, pp. 6278–6284.

[71] A. L'Afflitto, *A Mathematical Perspective on Flight Dynamics and Control*, 2017.

[72] W. Haddad and V. Chellabonia, *Nonlinear Dynamical Systems and Control: A Lyapunov-Based Approach*. Princeton, NJ: Princeton University Press, 2008.

[73] A. L'Afflitto, "Barrier lyapunov functions and constrained model reference adaptive control," *IEEE Control Systems Letters*, vol. 2, no. 3, pp. 441–446, 2018.

[74] B. Bernstein, *Matrix Mathematics*, 2nd ed. Princeton, NJ: Princeton University Press, 2009.

[75] S. Bouabdallah, "Design and control of quadrotors with applications to autonomous flying," Ph.D. dissertation, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2007.

[76] O. Hassanein, S. Anavatti, and T. Ray, "Robust position control for two-link manipulator," vol. 7, pp. 347–359, 2011.

[77] B. Horn, *Kinematics, Statics, and Dynamics of Two-D Manipulators*. Cambridge, MA: MIT Artificial Intelligence Laboratory, 1975.

[78] K. S. Narendra and A. M. Annaswamy, "A new adaptive law for robust adaptation without persistent excitation," in *American Control Conference*, 1986, pp. 1067–1072.

[79] F. P. Beer, *Vector Mechanics for Engineers: Statics*. New York City, NY: McGraw-Hill Education, 2012.

[80] D. Greenwood, *Advanced Dynamics*. New York City, NY: Cambridge University Press, 2006.