Chapter 6: System Behavioral Design

# Mini Abstract

1-2 paragraph chapter description. Should generally go over contents, expectations, and results. Abstracts are usually the last part of something to be written out since it is a summary of the article, but we can use them hear to help flesh out our ideas a bit for how to structure. Final abstract should be overhauled at the end of the chapter though, the chapter dictates the abstract, not the other way around.

Chapter 6 applies the physical system developed over the previous chapters to develop drone behavior from Startup Procedures to Remote and Autonomous Control to Landing Procedures. To begin the chapter, the system state machine is introduced covering the high level layout of the system, and the detailed operations within those high level function blocks are developed. Closed-Loop control and a few auxiliary functions were tested and verified in MATLAB, but were not verified in VREP. Also, autonomous control was not completed.

# 

# Chapter Outline

Develop the chapter outline here. Should become very detailed and broken down to paragraph level. Remember, if we invest time and effort into making a detailed outline, the actual writing will be far easier since we understand the flow and structure before we lay out the details. Before even writing a subsection, take the time to outline that subsection in the chapter outline. A lot of writing is in the layout. Remember to update this chapter in the Master Outline file so we can all keep track of the full outline of the report, its large so breaking it up this way should help everyone keep track of each other's ideas and work.

Chapter 6: System Behavioral Design

* Section 1: Discussion of Drone Operating Environments
  + Identify and elaborate on the conditions the drone will operate in in order to develop a model for how the system should be able to respond to multiple situations
    - 15 mph headwind
    - 5 mph minimum drone speed
    - Assuming standard temperature pressure for design, will need to identify limitations to this assumption by analyzing the change of buoyancy, but this should only matter if loss of buoyancy is beyond motor capabilities.
* Section 2: Day in the Life of The Barone
  + Provide a very high level overview of the drone throughout a normal day of operation to provide context for the state machine. We developed this flow chart during the winter quarter. We should update it and include it here
* Section 3: Detailed State Machine Design
  + Should be a detailed version of day in the life, but only include the system itself, not outside the environment. Include conditions for changing states and reasons for these decisions. This is the first design section of the chapter.
* Section 4: Startup Procedures
  + Software development to check functionality before takeoff
* Section 5: Physical Limitations
* Section 6: Open Loop Remote Control Design
  + Brief Review of Applicable Physics
    - No subsection, should be 1 page at most since it is all in Chapter 2
  + Design of Remote Control Response
  + Testing of Open Loop Remote Control Design in VREP
  + System Errors and Need for Autonomous Assistance
* Section 7: Linearization of Equations of Motion
  + Rectangular Coordinates
  + Spherical Coordinates
  + Assumptions and Limitations of Linearized Equations
* Section 8: Auxiliary Functions
  + Large Angle Error Response - 2 DOF
  + Hover with Stable Angle
  + Autonomous Take-Off and Landing
* Section 9: Closed Loop Remote Control Design
  + Design Overview
    - Essential Functionality
      * Stable Pitch and Roll
      * Stable Height
      * RC Input interpretation
      * Levels of Priority, stability before user response
    - Identification of Essential System Components
      * PRH Regulator
      * RC Command Interpreter
      * Saturator to Scale motor Commands
      * RC interpretation is only done with the available resources after the regulator is complete since regulator is used for system stability
  + Design of the Closed Loop System Response
    - Adjustments to PRH system regulator
    - Saturator Design
    - Open loop control response design
    - Component Integration
    - Assumptions and Limitations
  + Closed Loop RC Validation
    - Testing responses in Matlab
    - Testing Robustness in Matlab
    - Testing drone Response in VREP
* Section 10: Autonomous Design
  + Fill in when I get there. Not there
* Section 11: Error Identification and Response

# Chapter 6 Draft

Chapter 6 applies the physical system developed over the previous chapters to develop drone behavior from Startup Procedures to Remote and Autonomous Control to Landing Procedures. To begin the chapter, the system state machine is introduced covering the high level layout of the system, and the detailed operations within those high level function blocks are developed. Closed-Loop control and a few auxiliary functions were tested and verified in MATLAB, but were not verified in VREP. Also, autonomous control was not completed.

## 6.1 Discussion of Drone Operating Environments

The drone must be able to perform in the operating environments defined by the client, Primarily STR 2.0.0, Minimal Drone Speed, the Drone shall be able to fly at least 5mph in winds up to 15mph. Assuming standard temperature pressure for design, will need to identify limitations to this assumption by analyzing the change of buoyancy, but this should only matter if loss of buoyancy is beyond motor capabilities. Disturbances can be caused by outside forces such as drag that can destabilize critical system states, and these must remain stable for the drone to be controllable.

## 6.2 Day in the Life of The Barone

In order to satisfy STRs 3.0.0, Remote Control and 4.0.0, Autonomous Control, a system state machine had to be developed that would be capable of performing multiple tests before and during flight, switching between different modes of drone control, and providing user feedback. To do this, a system flowchart was created that would show the drones’ actions from a very high-level perspective:

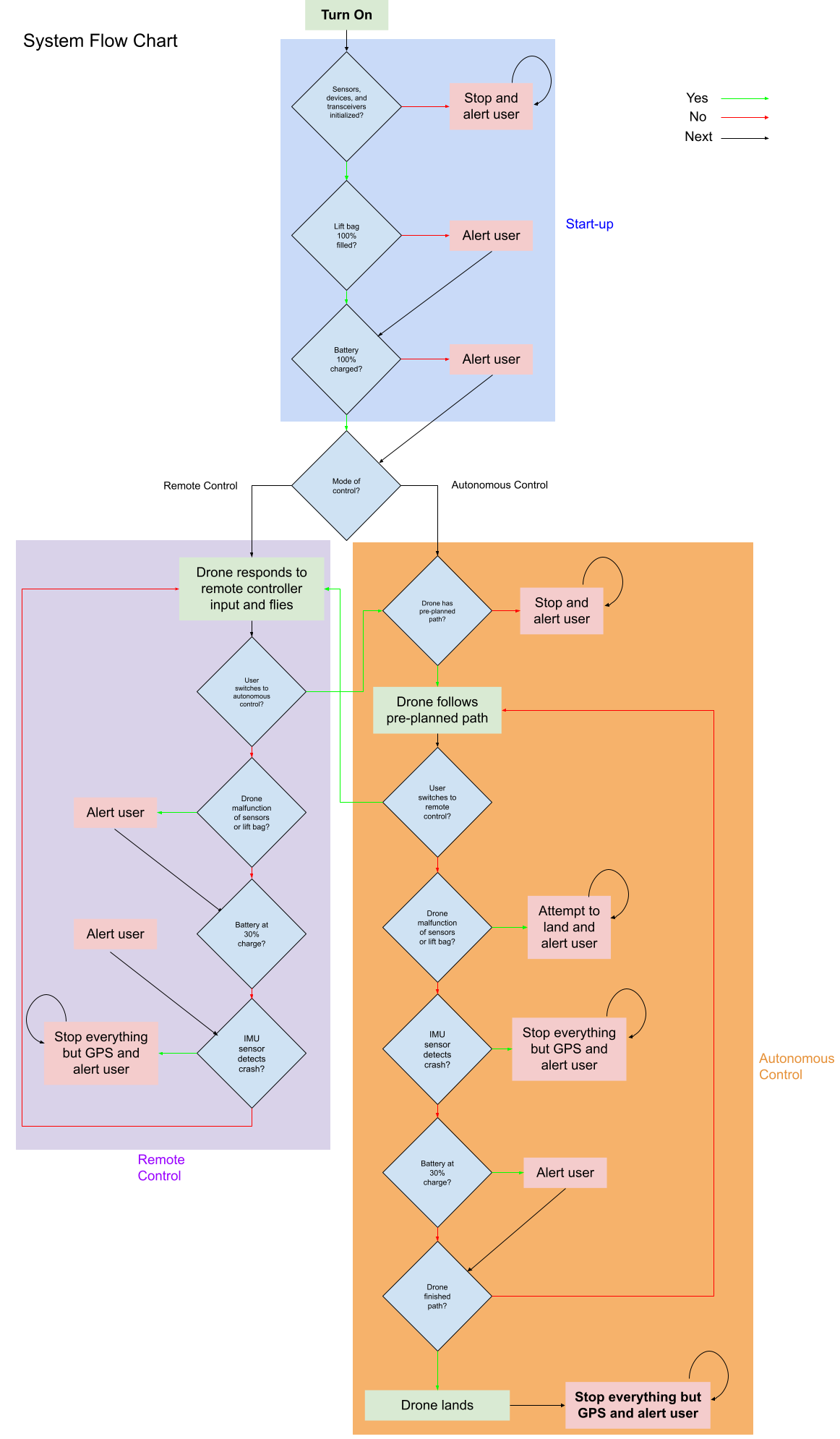


Fig. 6.1. System Flow Chart Part 1-preflight Functions

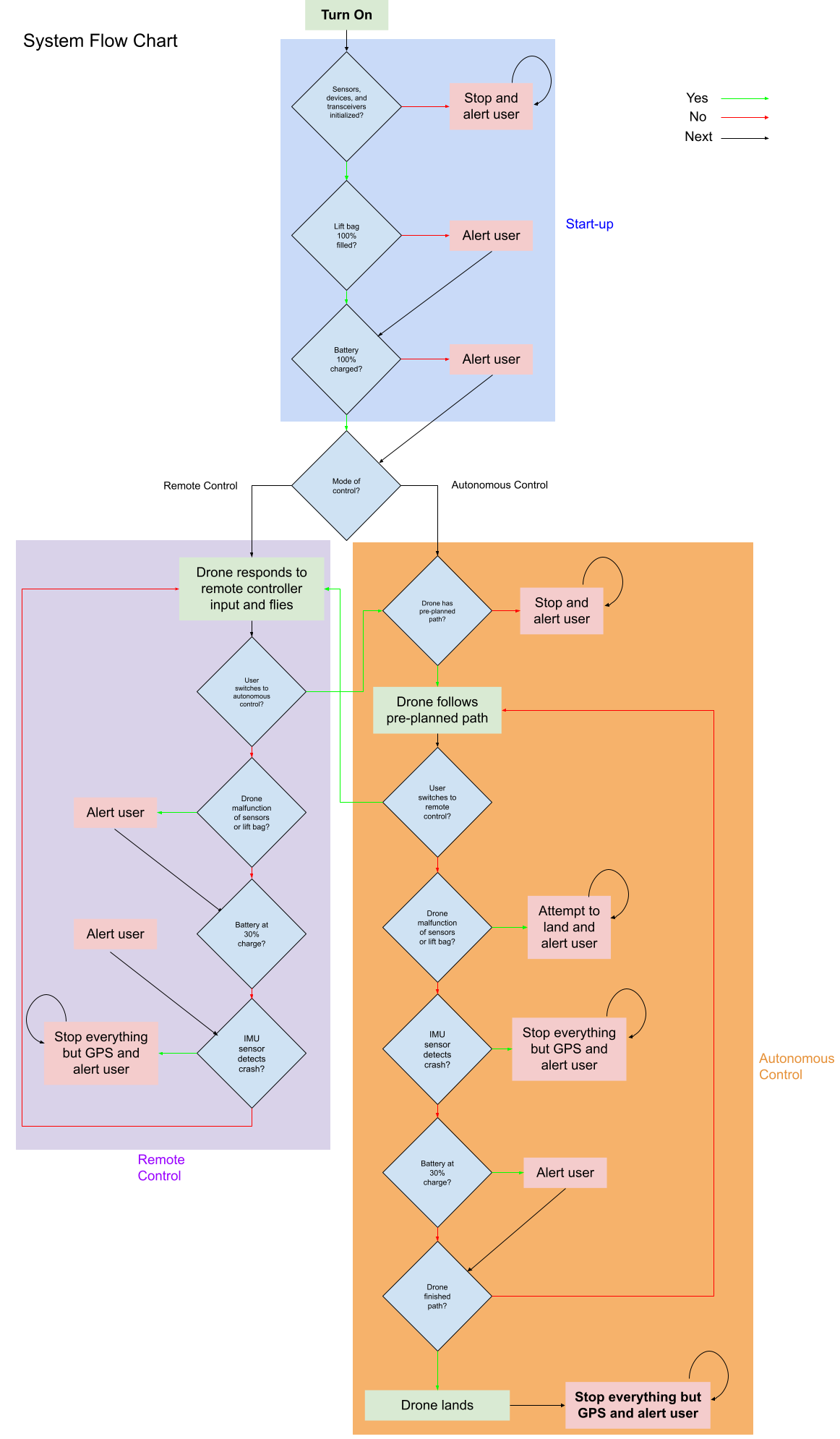


Fig. 6.2. System Flow Chart Part 2-Flight Functions

Starting from Fig. 6.1, when the drone is first turned on, it enters the start-up procedure, where it first checks if the sensors, devices, and transceivers were initialized properly. If any of them weren’t, the drone stops what it is doing and continuously alerts the user that something is wrong. If they were initialized properly, the lift bag is then checked for 100% capacity, and the drone alerts the user if it isn’t but keeps going to check that the battery is 100% checked, where it does the same thing and advances towards checking which mode of control it is in. If the mode of control is remote control, then the drone enters the remote control procedure where it responds to remote controller input and flies. Looking at Fig. 6.2, the drone then checks if the user switched to autonomous control, and if they did, the drone moves to the autonomous control procedure, but if they didn’t, the drone then checks if any sensors malfunctioned or if the lift bag was punctured. If it was, the drone alerts the user and keeps going towards checking if the battery is at 30% charge, and does the same thing if it is. Finally, the drone checks if the IMU sensor detected a crash, and if it did, the drone stops what it’s doing and continuously alerts the user; otherwise it returns back to responding to the remote controller and flying.

Meanwhile, if the mode of control was chosen to be autonomous control, the drone first checks if it has a pre-planned path as seen in Fig. 6.2. If it doesn’t, it stops everything and continuously alerts the user, but if it does, the drone follows the pre-planned path. The drone then checks if the user switched to remote control, and if they did, the drone switches to the remote control procedure, otherwise it goes on to check if any sensors malfunctioned or if the lift bag was punctured. If it was, the drone attempts to land by itself, and continuously alerts the user, but if it doesn't, it checks if the IMU sensor detected a crash, to which it will stop everything but GPS and continuously alert the user if that was the case. If not, the drone checks if the battery is at 30% charge as seen in Fig. 6.2, and alerts the user if it is, and then checks if the drone completed the pre-planned path. If it doesn't, the drone continues on its pre-planned path, and if it does, the drone lands and stops everything but GPS and continuously alerts the user.

## 6.3 Detailed State Machine Design

Once the system flowchart was designed, it was easier to create a lower-level state machine that would essentially perform the same actions as the flowchart but in greater detail:

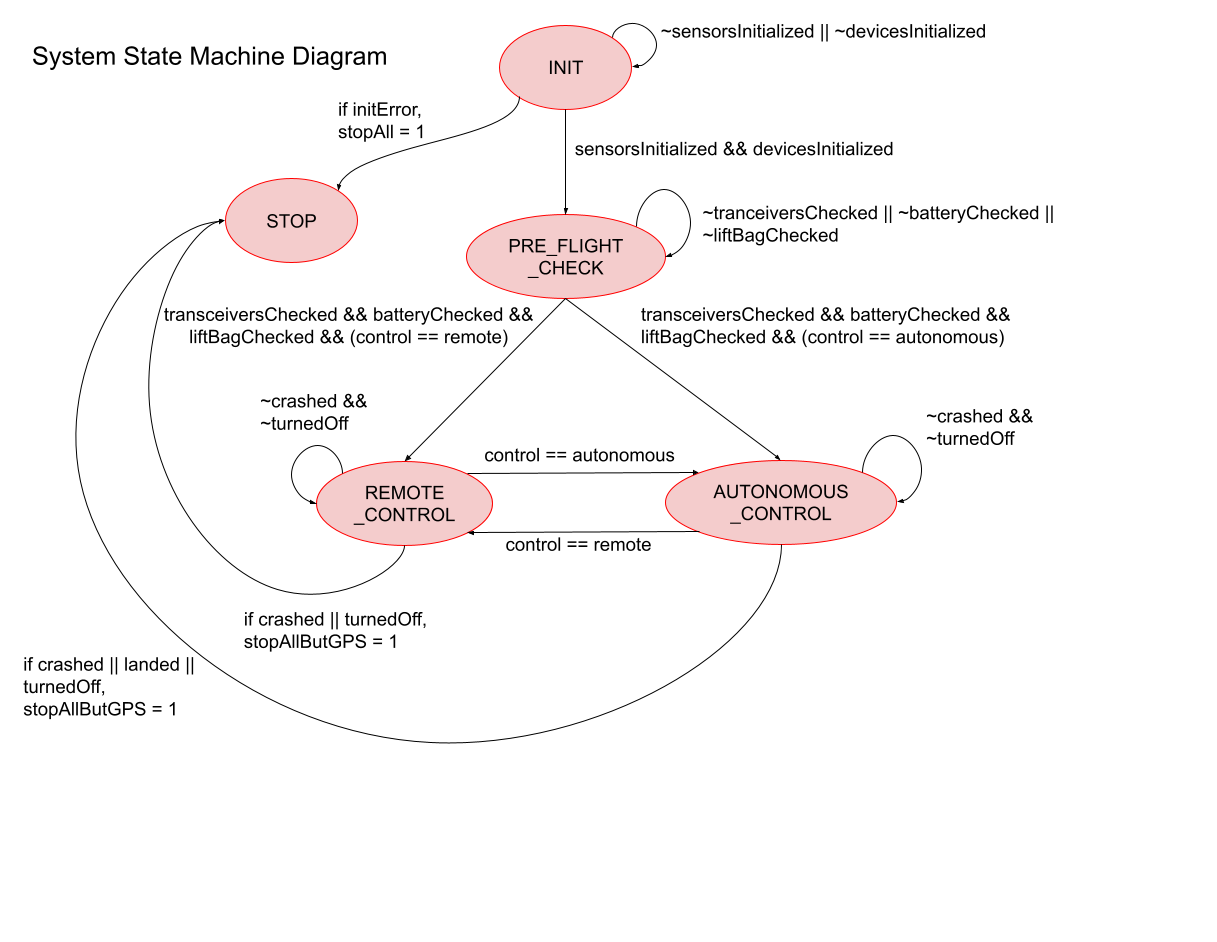


Fig. 6.3. System State Machine

On start-up, the drone enters the INIT state, and remains there until all sensors and devices are initialized. If there is an error in initialization, the drone enters the STOP state and remains there and alerts the user, but if there isn’t the drone enters the PRE\_FLIGHT\_CHECK state where it remains until all transceivers, the battery, and the lift bag are checked. Once this is true, if the mode of control was remote control, then the drone switches to the REMOTE\_CONTROL state, where it remains until it crashes, is turned off, or switched to autonomous control. If the drone crashes or is turned off, it stops all but GPS and enters the STOP state and alerts the user, and if the drone is switched to autonomous control, it enters the AUTONOMOUS\_CONTROL state.

Meanwhile, if the mode of control was autonomous control, then the drone switches to the AUTONOMOUS\_CONTROL state, where it remains until it crashes, lands, is turned off, or switched to remote control. If the drone crashes, lands, or is turned off, it enters the STOP state and stops all but GPS and alerts the user, and if the drone is switched to remote control, it enters the REMOTE\_CONTROL state.

This state machine is only theoretical and has never been written in code and tested, so the system state machine has not contributed anything to meeting STR 3.0.0, RC Control, and 4.0.0, Autonomous Control. If the state machine were to be implemented and tested on the drone during flight, it would have made some progress towards meeting the requirements.

## 6.4 Startup Procedures

On start-up, the drone performs several tests to ensure it can properly fly, and either prevents the drone from flying or warns them so that they may make their own decision to progress. The first thing the drone checks is if the sensors, devices, and transceivers are initialized correctly. This includes making sure that the ultrasonic, IMU, altimeter, GPS, and air pressure sensors are detected and enabled by the PIC32, the Raspberry Pi and PIC32 establish a connection, and the remote controller receiver and data telemetry transmitter are ready to be ready from/transmit data; if any of the sensors or devices fail to initialize properly, the start-up procedure is stopped by the drone. Once this is completed successfully, the level of helium in the lift bag is checked by the air pressure sensor, and alerts the user to make their own decision whether the flight should continue. Finally, the battery is checked by means of the battery sensor attached to the battery, and lets the user know if it isn’t 100% charged.

## 6.5 Linearization of Equations of Motion

A control method is needed to develop the predictable and stable response of the drone behavior. The creation of a nonlinear control system is complicated and beyond the scope of the team’s current abilities. Nonlinear control systems are also more computationally complex, and the delayed control system response can affect the stability of the drone. For design and computational simplicity, a linear control system design is chosen. This section will reference section 2.whatever from this report, and several figures and equations will be repeated here for easy reference, but section 2.whatever will cover their original derivations.

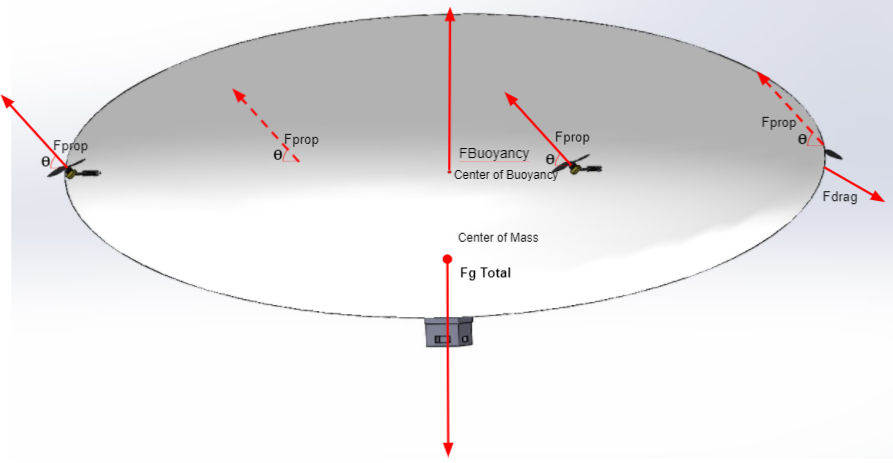


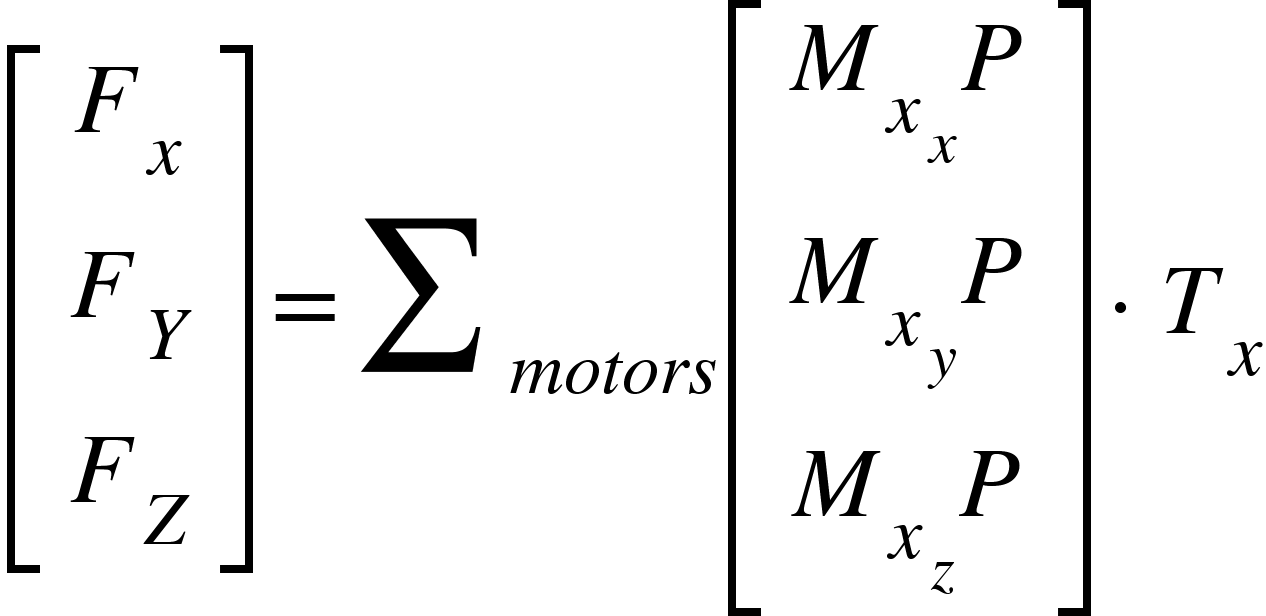
Fig. 2.1. Force Diagram

6.5.1 Translational Movement

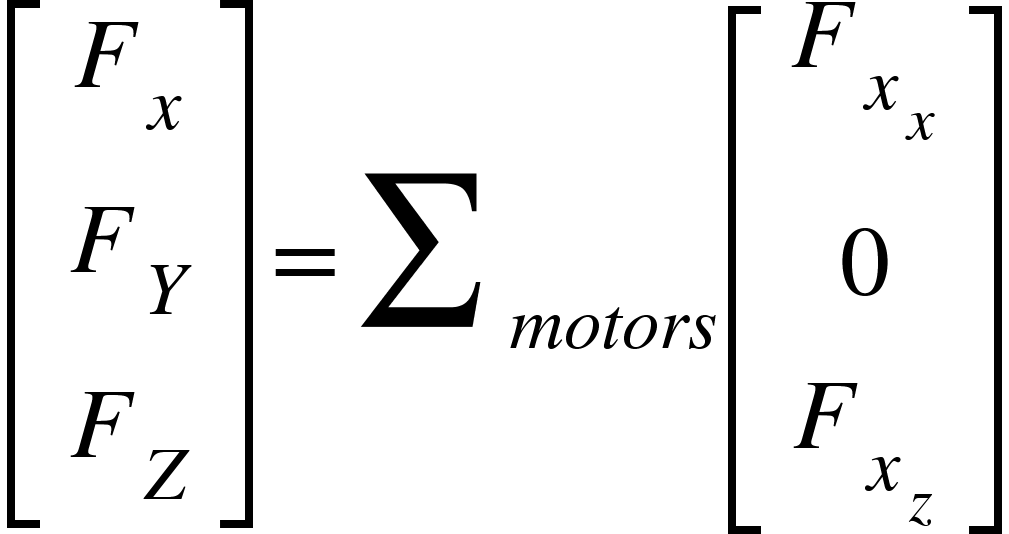
The vectorized net force of the drone’s control mechanisms, is given by equation equation 2.7

(2.7)

Where is the vector describing the direction of the propeller force (affected by servo position), and is the thrust provided. To expand the equation slightly, the vector components are written in equation 6.1.

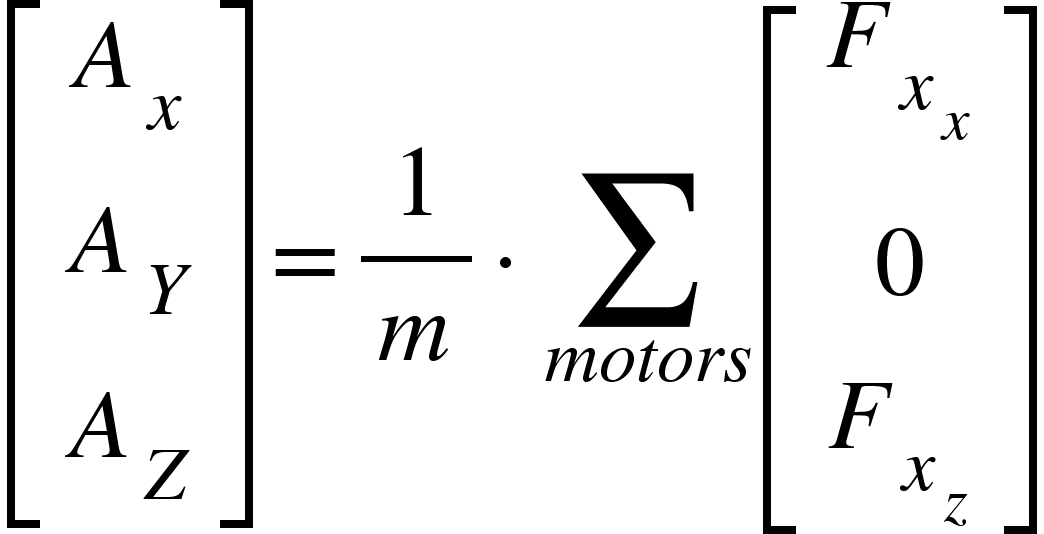
 (6.1)

Where the second subscript denotes the force direction, but can be simplified to equation 6.2.

 (6.2)

Where the new force components are the result of . and since as designed in the propulsion system. This equation is linear and simple to implement in a linear control system directly calculating the force responses needed. Limitations of this method are discussed in subsection 6.5.3.

Applying Newton's second law, the acceleration of the system from internal forces is calculated in equation 6.3.

 (6.3)

6.5.2 Angular Movement

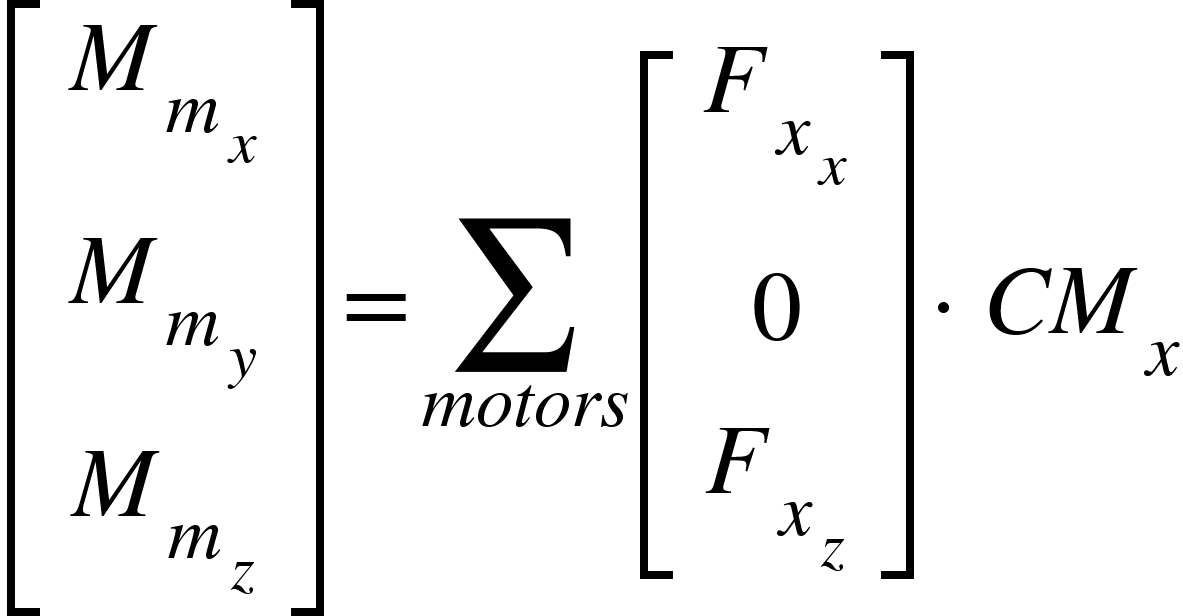
Angular approximations are more complicated to set up since positions of the center of mass, center of buoyancy, and propellers need to be known in addition to their forces. Equation 2.12 summarizes the behavior of the moments affected by the propulsion system.

(2.12)

To begin, the term is a vector that represents the propeller position relative to the motor to determine the accurate propeller position, since the propeller itself is the source of the force. However, due to the drone’s large size, is small compared to since the center of mass to the motor position is approximately 55 inches whereas the length d is only a couple of inches, and 2.12 can be approximated as equation 6.4.

(6.4)

Once again, expanding the equations out to their vector components will help identify and isolate the individual components of the system, as shown in equation 6.5.

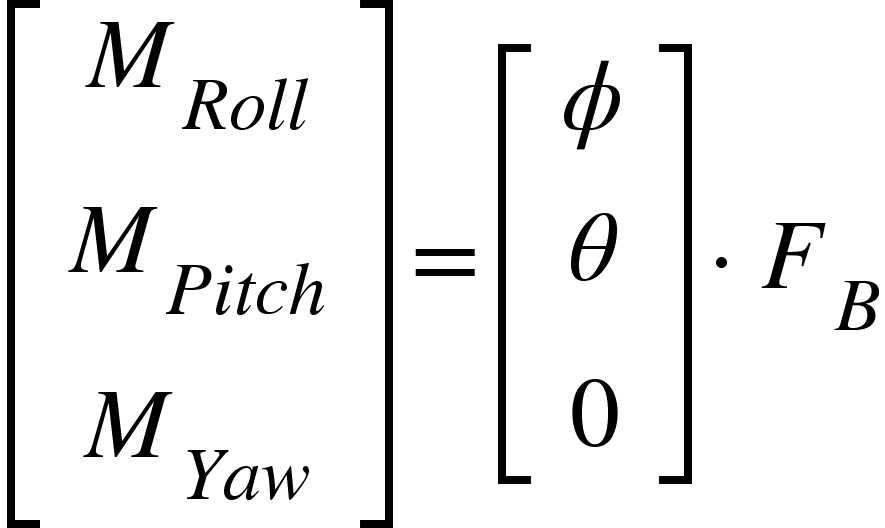
 (6.5)

The moment caused by buoyancy is given by the equation 2.13

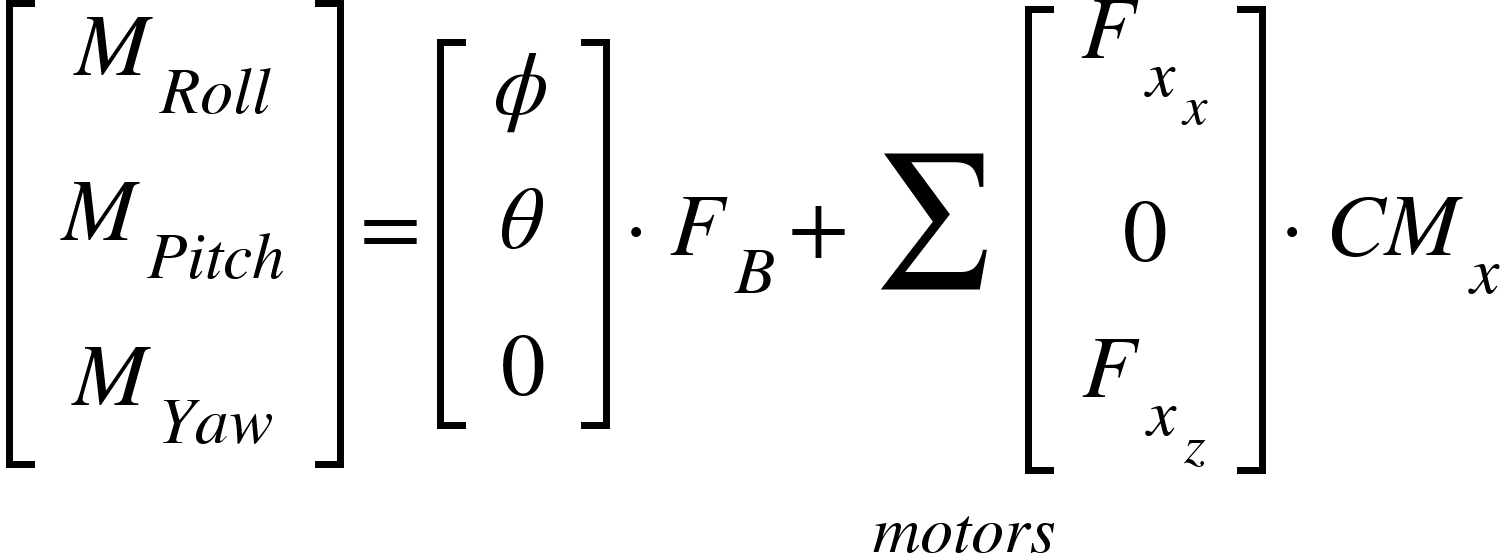
(2.13)

Since we are defining the center of the system as the center of mass, the center of mass contributes no moment and simplifies the math a bit. The R term is the rotation matrix and dependent on the angular position of the drone, and has very nonlinear dependencies involving sines, cosines, multiple terms products of each other, and simply is too nonlinear to incorporate.

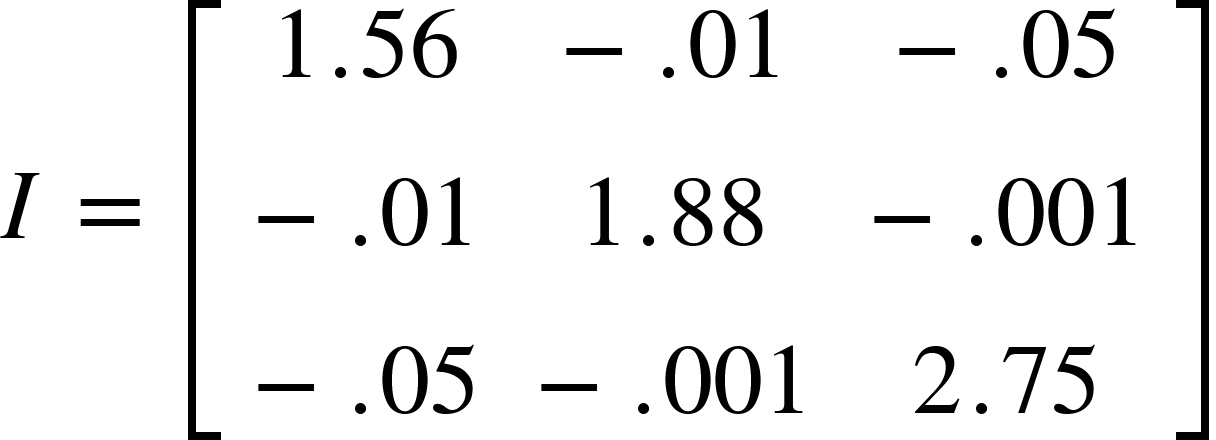
This is a complicated relationship so several approximations will be made to ensure the system is linear. The primary simplification that is used is the small angle approximation, where the sin of an angle in radians is approximately the angle and the cos of the angle is 1, and the approximation can be applied since the drone design does not require any tilt to maneuver. The buoyant moment can be simplified and expressed with equation 6.6.

 (6.6)

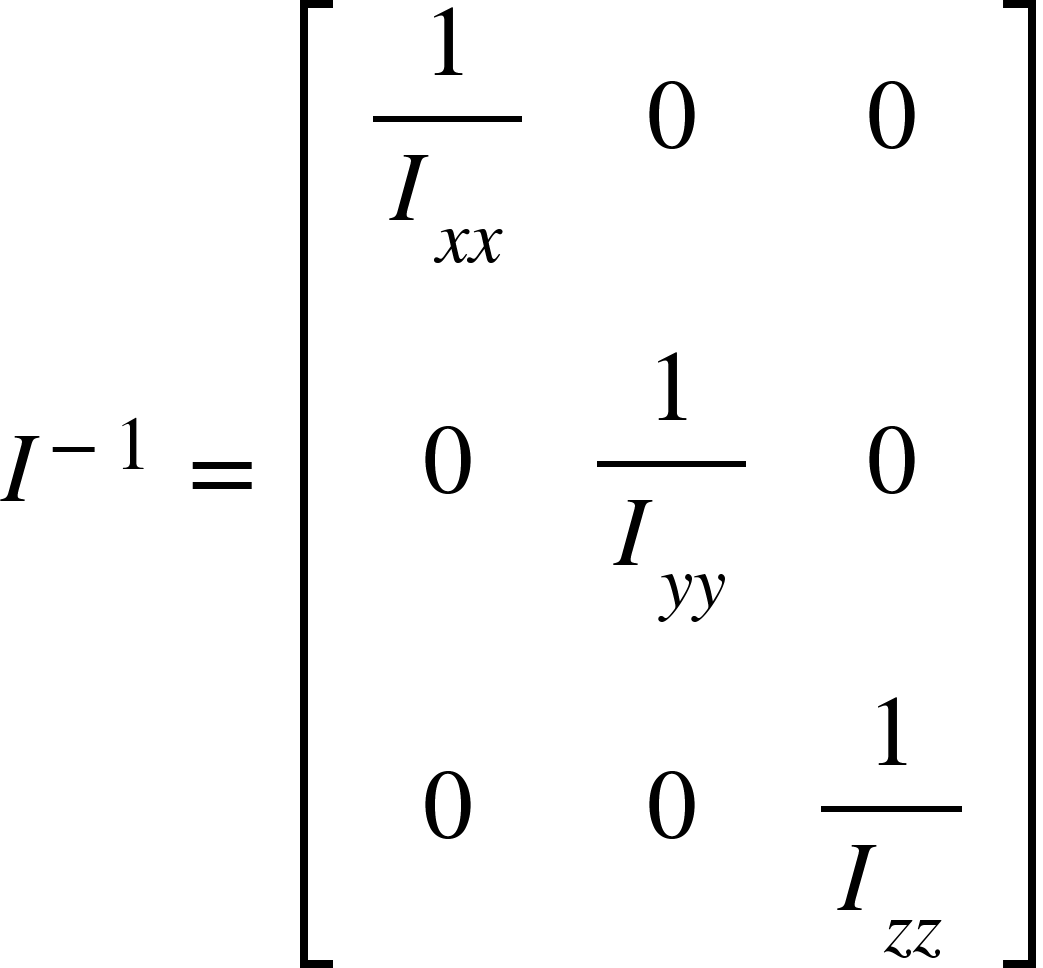
The moments caused by buoyancy are now given directly by the angular positions of the drone and are linear and can be combined with the moments caused by the motors in equation 6.7.

 (6.7)

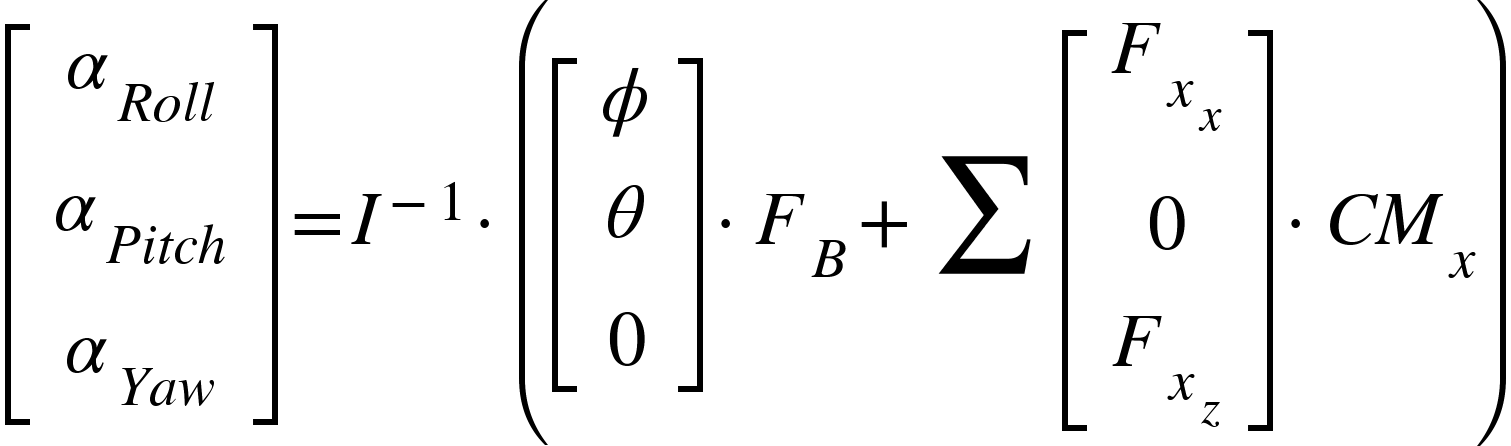
To find the angular acceleration the drone moments must be multiplied by the inertia matrix for the drone about the center of mass, given in equation 6.8.

 (6.8)

Since the diagonal values are much larger than the other values, *I* can be approximated as a diagonal matrix, which will also simplify the inverse of the matrix to be 6.9.

 (6.9)

The angular accelerations are now given in a linearized form in equation 6.10.

 (6.10)

The equations of motions are all linearized and ready to be incorporated into a linear control system, however, the assumptions do face limitations that will be discussed in the next section.

6.5.3 Assumptions and Limitations

To linearize the equations of motion to be used in a linear control system, a few assumptions were made.

First, the system uses the small angle approximation. The approximation is only accurate within 0.2 radians of zero. Since our propulsion system and buoyant moment mean that we do not need to tilt to maneuver, the limitation is manageable. To use the approximation, the drone must be kept as close to zero as possible, which is also required to have minimal drag on the drone anyways. The target for tilt of the drone will be within 0.1 radians to provide extra room for the system to operate in case something does go wrong. Additionally, a 2-DOF control system should be implemented to restabilize the pitch and roll angles of the drone if the drone starts to approach the limitations of the approximation, and, since the control system estimations lose accuracy and the drag will have a greater impact on the drone leading to system instability, all other functions should be abandoned to restabilize the pitch and roll angles of the drone.

Another limitation is that we treat Fx and Fz components of the motor forces separate during calculations. On the drone, the Fx and Fz components are linked since there is a single force and the direction is controlled by the servo angle, however, this is nonlinear. The Fx and Fz components can be decoupled, as was done in the system linearization, however, the limitations of the forces need to be passed through a function in order to convert the values back to throttle and angle values for the motors and servos.

Approximations were made to help ensure the system remained manageable and linear. This will result in inaccuracy of the system response, so an integral path needs to be added to the control system to reject the error. The integral path also needs to be present to reject the disturbance of wind and other outside factors, but that's expected. To work around the errors associated with the approximations, the system needs to be tested in detailed simulation, VREP, at ideal conditions, i.e. no wind. The integral values can be recorded and the integral paths can be initialized with the recorded values to prevent error at the start of the flight path and prevent overshoot, since the system will not need time to build up the integral values. Then the adjustments of the integral path can respond directly to wind, with the steady state bias already incorporated.

## 6.6 Open Loop Remote Control Design

6.6.1 Design of Remote Control Response

STR 3.0.0, Remote Control, requires a remote control implementation of the drone that is capable of responding to user input while maintaining stable pitch and rolls, <-0.1 radians from zero. The drone differs from other drones since it does not need to tilt, so the Remote Commands need to be interpreted. To prevent the need to design a custom remote controller, a standard controller is used, but needs to have inputs modified to fall in line with the needs of the drone.

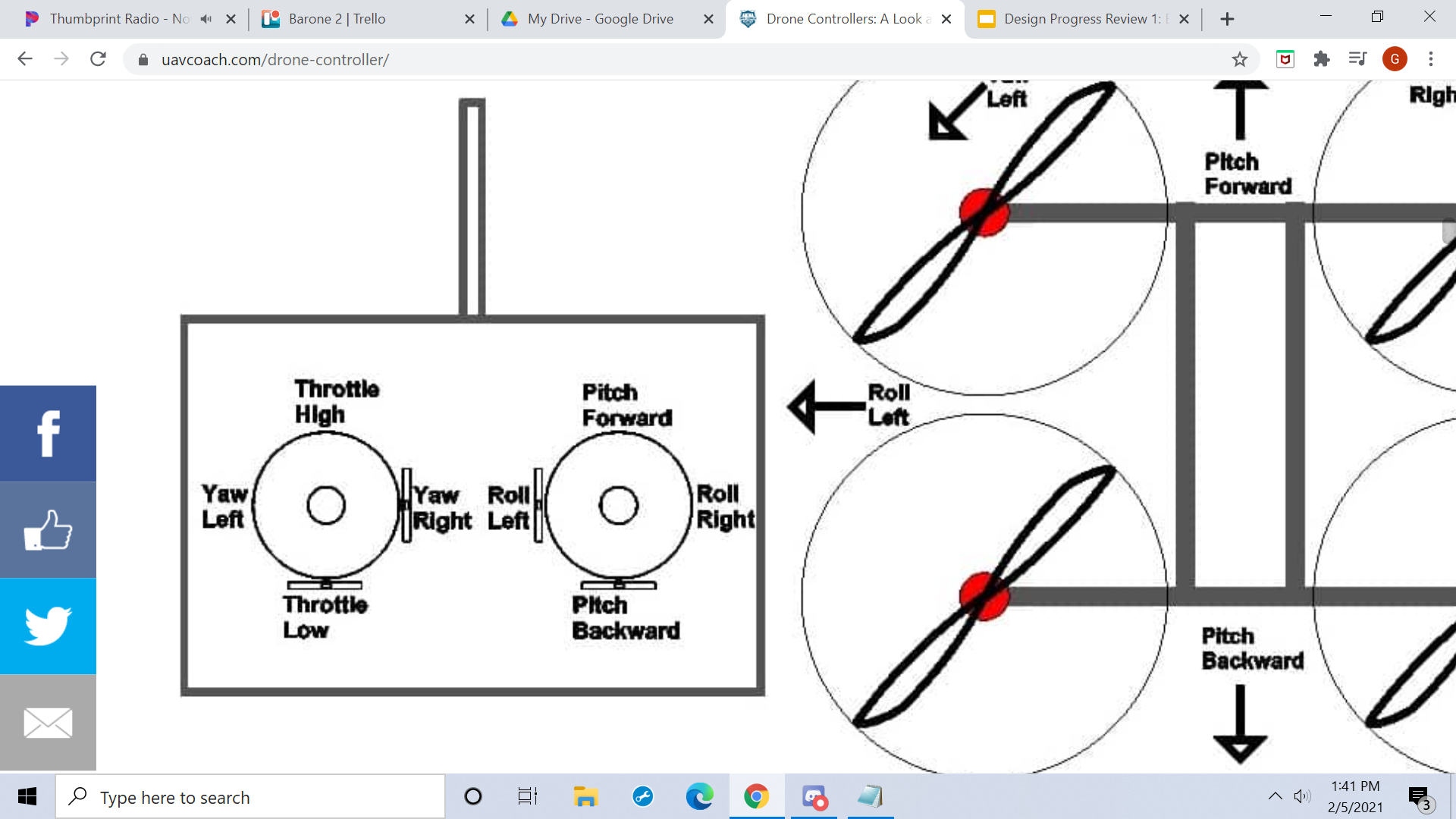


Fig. 6.4. Typical RC Controller Layout

Fig. 6.4 provides the basic layout of most controllers. Pitch forward and backward translate to forward and backward movement. Yaw left and right translate to turn left and right movement. Roll Left and right translate to moving left and right. The throttle translates to power given to the propellers. Since standard drones move by tilting, the throttle just needs to scale all motor commands, and adjusting the pitch and roll angles can translationally move. Also, throttle is how the drone will ascend, since high throttle will move the drone up, and low throttle lets gravity pull the drone down.

The Barone operates differently than other drones so several inputs need to be renamed. Instead of pitch, the command is changed to forward, since we do not change pitch. Roll inputs are not used, since the Barone is not capable of moving sideways with the propulsion setup and the need to not change roll. Yaw is just changed to turn, the drone can change yaw but since pitch is changed to forward turn is just chosen to prevent jumping between greek letters and command names. Throttle high and low is a slightly more complicated setup. Gravity can not be relied on to pull down the Barone. Switching between a negative one value and positive one value on the throttle control to control upwards and downwards movement is confusing when it will always be treated as positive for turn and command inputs, so it is not a great solution. Instead, the throttle will control the throttle, and a button needs to be added to change between ascend and descend. The RC commands become turn, forward, throttle, and a binary input for ascending and descending.

Next the values need to be scaled to servo angles to control forward, turn, and height control to the drone. The drone needs to be able to go full forward, and full turn, but should also have angles scaled if both inputs are maxed to ensure both commands and desired magnitudes are implemented. The behavior is set up in equation 6.11.

(6.11)

The forward and turn commands are combined, positive turn on the right side and negative on the left, to get a combination of inputs. Next, to scale the values to always be less than one, the denominator considers the product of the two terms and adds them to 1 in order to scale the full turn. If either forward or turn are zero, the denominator is one and the servo angles will give max commands. If both are maxed, the denominator becomes 2, as well as the numerator, resulting in a net value of 1 on one side, within the max. The equation also allows for values in between and scales the servo angles accordingly. This also allows for the servos to still allow propellers in the up and down directions so the drone can move up and down. The pi/2 term is just for scaling of the system.

In the max command example given previously, one side is given a full forward command, but the opposite side has a full up command, resulting in an unintentional moment as shown in Fig. 6.5.

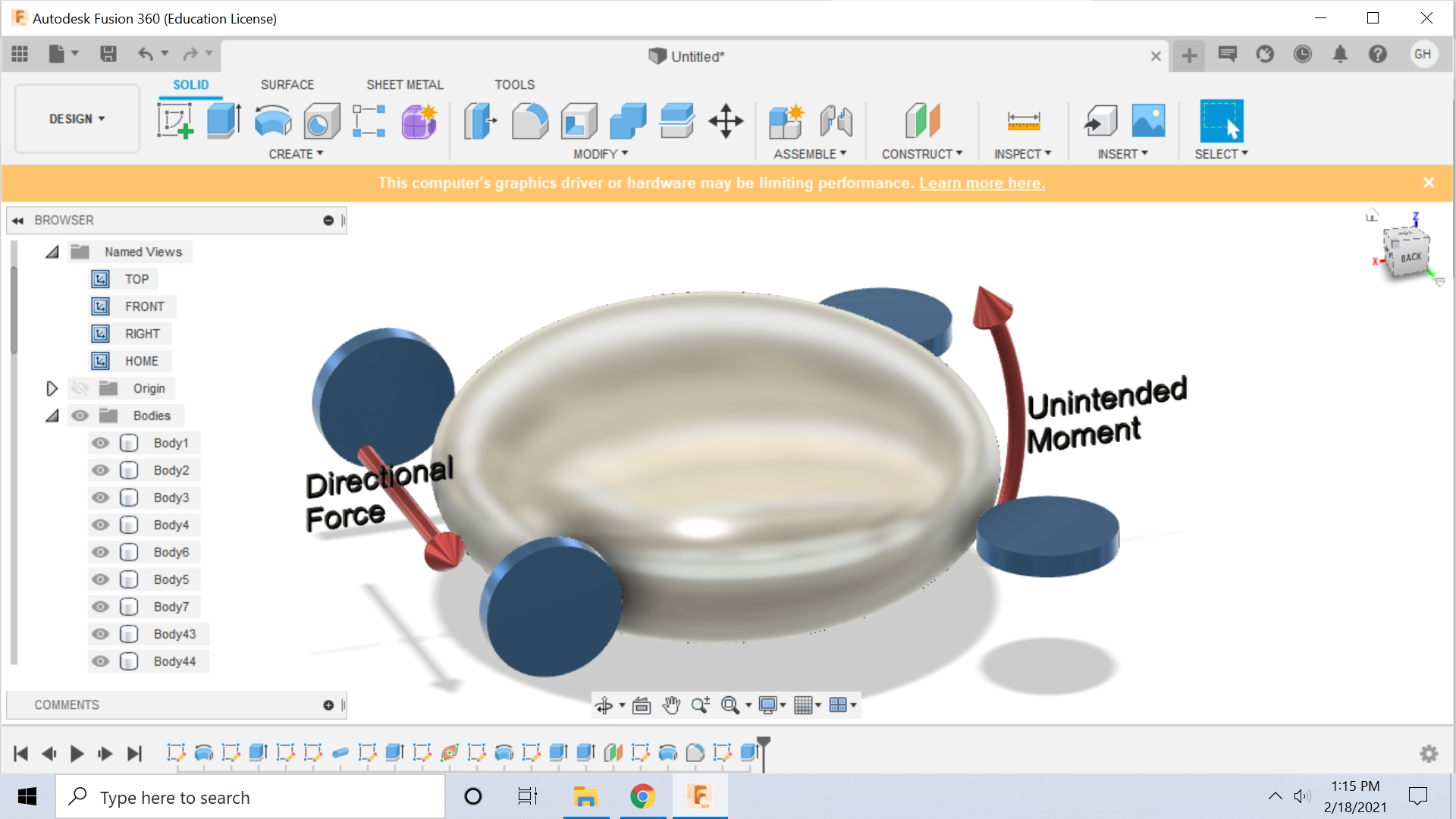


Fig. 6.5. Unintentional Moment With RC Commands

The unintentional moment affects system stability, so the throttle needs to be adjusted. The up and down forces need to remain equal on both sides, so the z components of the thrust values need to equal. The z components of the angles are isolated using the cosine function and the inner value of the turn, the side that would provide the excess force in the z-direction, and that is multiplied by the ratio of the z-component to scale, and is modeled in equation 6.12.

(6.12)

6.6.2 Testing of Open-Loop Control Design and Instability

The open-loop control response was simulated in VREP due to the complexity of the system response. Due to the setup, the full simulation is given in Chapter 9, but in short the drone was unstable during flight.

The drone’s propulsion system is mounted above the center of mass, so when the propellers apply a force, there is a pitch moment created as illustrated in Fig 6.6.

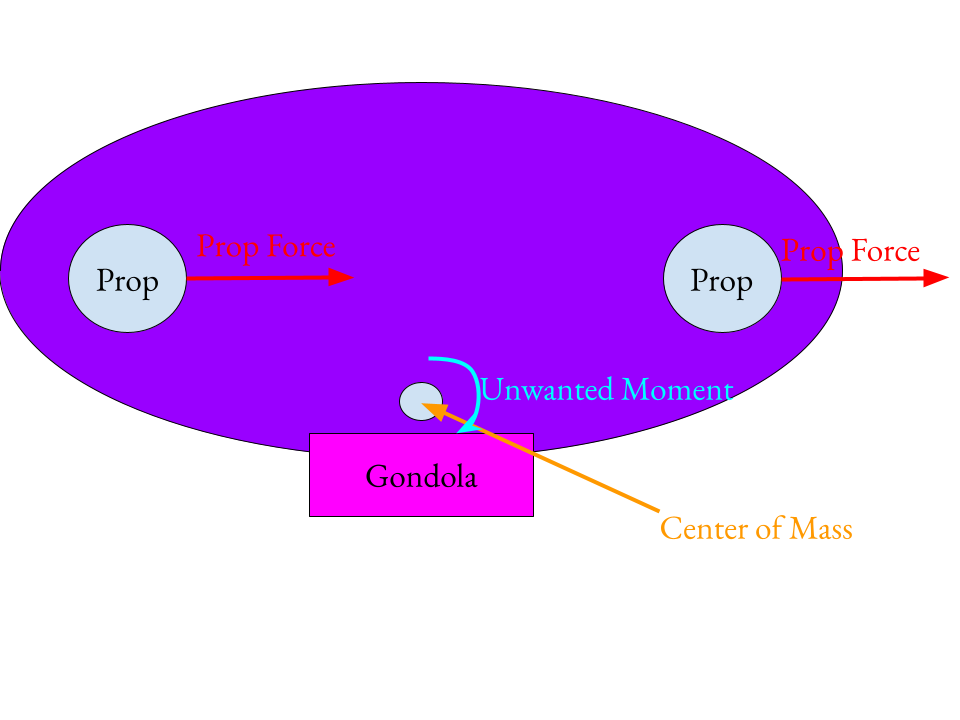


Fig. 6.6. Pitch Moment Caused by Propeller Force

In addition to the effect of the unwanted pitch moment, any other changes on the pitch and roll affect the system, and since the open loop controls don’t take the angles into account, the system stability will remain and potentially amplify. The system cannot operate on an open-loop RC system alone, and a closed-loop RC system is needed in order to maintain the system stability.

## 6.7 Auxiliary Functions

STR 3.1.3, Autonomous Functions, requires multiple functions to aid with Remote Control, including large tilt angle handling, hovering, auto landing and auto take-off. The Auxiliary Functions will be covered before the Closed- Loop RC for a few reasons. Although the auxiliary functions themselves are less important to be designed than the closed loop RC, they provide the basis for the Closed-Loop RC, and the simpler designs of the auxiliary functions enable the system response to be developed in complexity over time, with testing in between functions. To be more specific, the Large Angle Error response only maintains the pitch and roll angles of the drone to be zero and is the simplest closed loop control system on the drone. The Large Angle Error can then have height added to create a hovering system with stable angles. After height and angles are controlled, the auto-takeoff and landing can be implemented by feeding a ramp input to the hovering function, and tests the system against input response. The approach enables all the components of the closed-loop RC to be built and tested incrementally before variable user input is tested, which will be the hardest test the controls will encounter due to the uncertainty of user inputted commands as well as external forces such as drag.

6.7.1 Large Angle Error Response

The large angle error is the simplest and most important control system for the Barone. Pitch and roll stability is pivotal to drone stability, and every control system built for the drone will prioritize pitch and roll above other states. One of the reasons for the importance of maintaining pitch and roll is given in Fig 6.7.

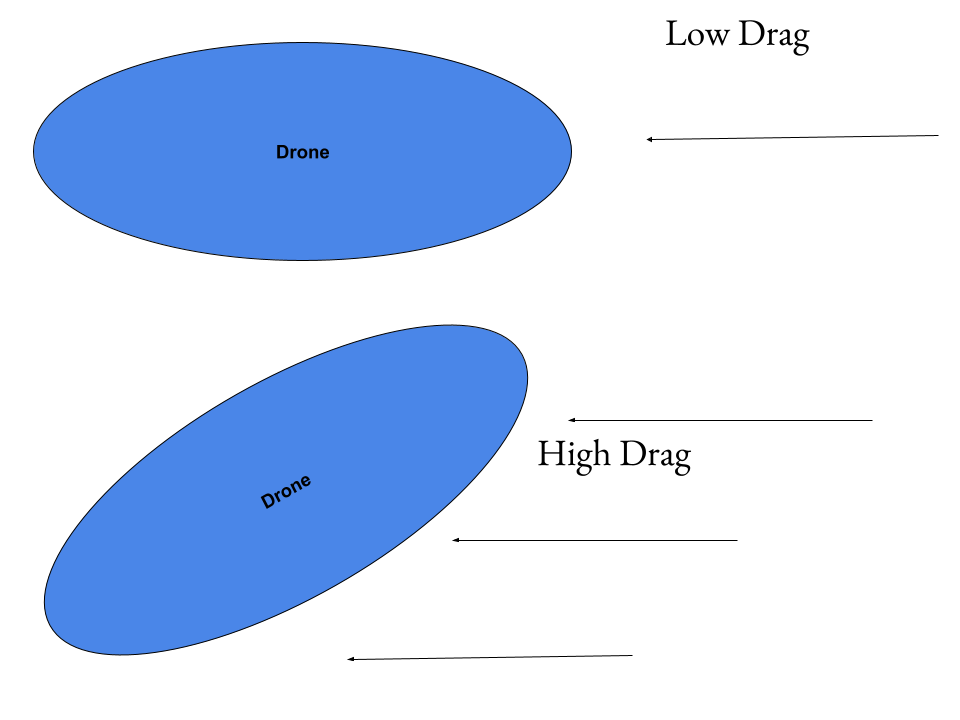
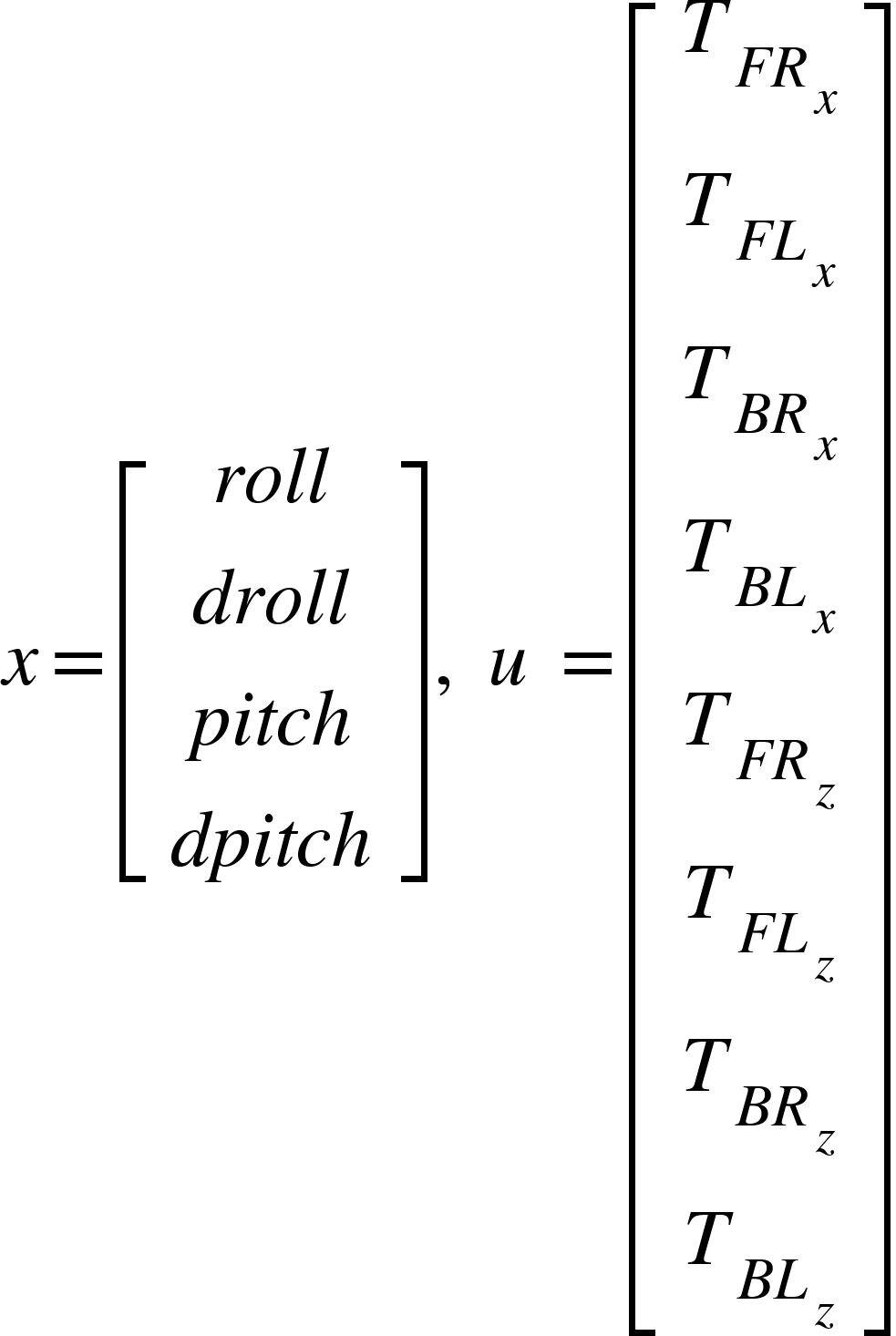


Fig. 6.7. Effects on Drag on Large Tilt Angles

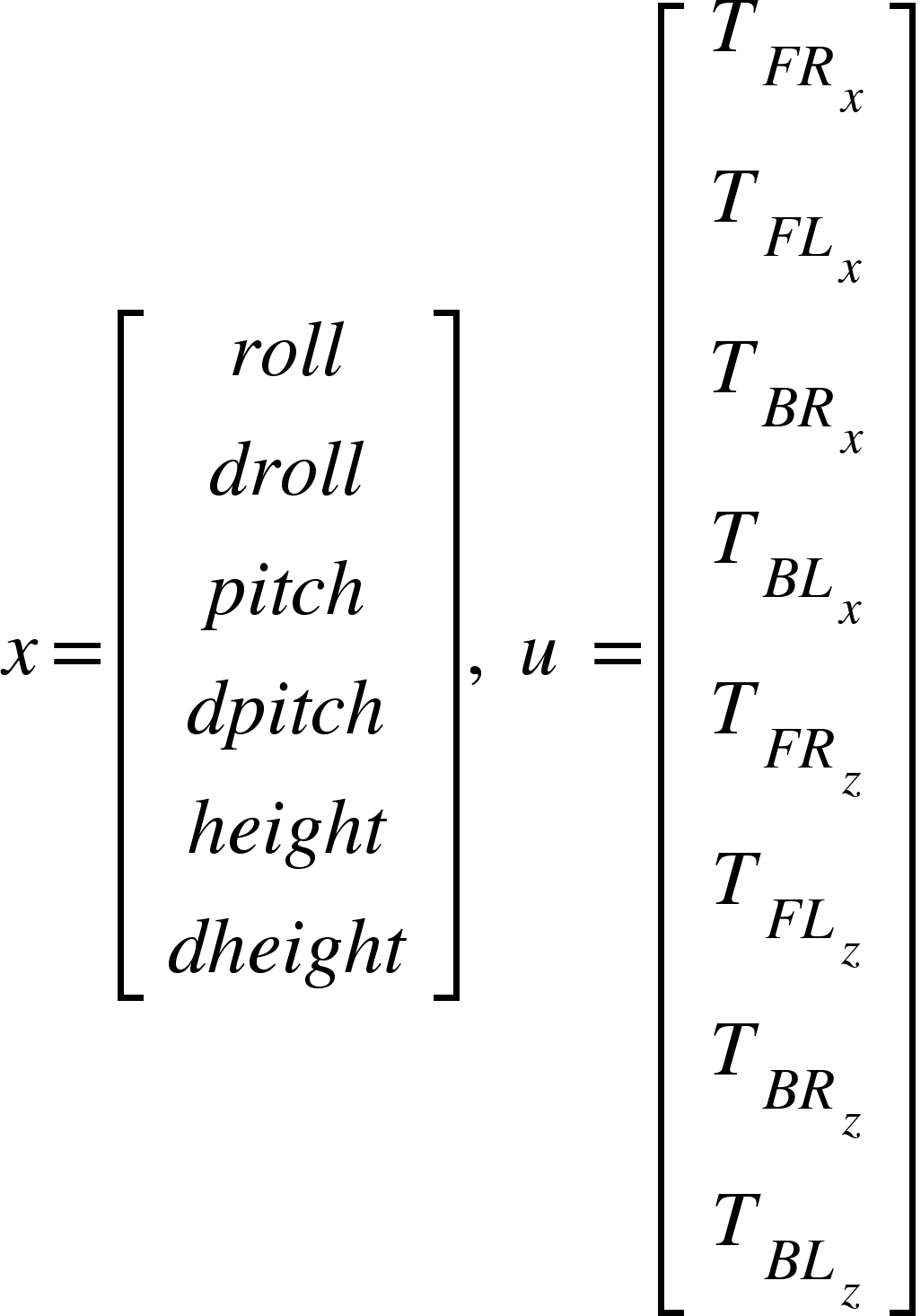
The Barone has an ellipsoid shape, and when the drone’s tilt is near zero, has very low drag. As the tilt angle increases, the cross sectional area of the lift bag in relation to the air velocity increases, drastically increasing the drag. In addition to the small angle approximation error that arises at the tilt angle increases, the drag will increase dramatically, and can result in the loss of controllability of the drone and crash--this is probably the worst case scenario for the control system, and it is why the pitch and roll angles are prioritized. As mentioned in the system linearization, the force components of the propellers are decoupled, so the input and state vectors are defined as

 (6.13)

Where x is the state vector, u is the input vector, T is thrust of the propeller, and FR stands for front right, FL stands for front left, BR stands for back right, and BL stands for back left, and the x and z stand for the x and z components of the force they represent. Due to the number of inputs and outputs, as well as the dependencies that exist between each other, a full state feedback system is implemented. The A and B matrices are determined using equation 6.10. The C matrix is given by [1 0 1 0] since the drone only needs stable pitch and roll and their derivatives can vary.

6.7.2 Pitch Roll Height Regulator

To add height to the regulator developed in section 6.7.1, the states only needed to be expanded to equation 6.14. STR 3.0.0 Autonomous Control had a height control added to state that the height must be within 0.15 meters of 1 meter off the ground.

 (6.14)

And the A and B matrices are expanded to incorporate changes in height using equation 6.3. An integral path is needed to help reject disturbance, so it is added to the system, as shown in Fig. 6.8.

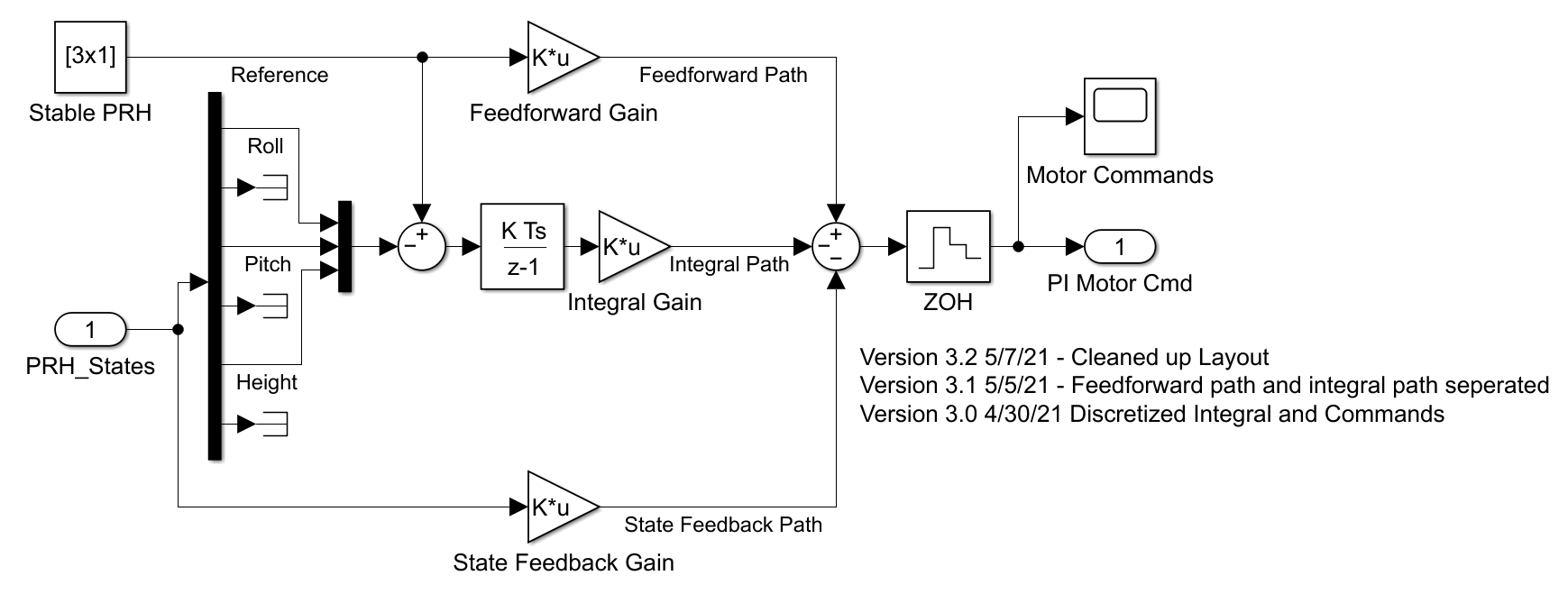


Fig. 6.8. PRH\_PI Regulator

Since the regulator controls pitch roll and height, and has proportional and integral gains, it is referred to as the PRH\_PI regulator. As shown in Fig 6.8, full state feedback, feedforward gain, and integral gain are all used to control the system response. Pole placement was used to place the poles originally, and somewhat arbitrarily, with the pitch and roll states having the poles farthest from zero since their response speed is more important than conserving energy. The feedforward gains were determined by inverting the DC gains of the state feedback path to scale the system response back to a DC gain of 1. The integral path was adjusted manually to be 1/10th of the feedback gain, and this can be further adjusted with simulation in VREP. The responses of the system at undesirable starting states are given in Fig. 6.9.

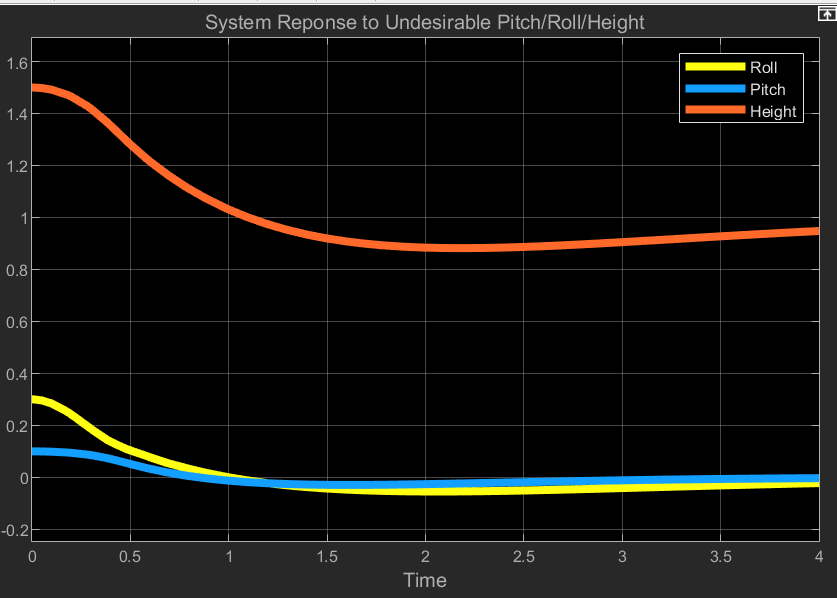
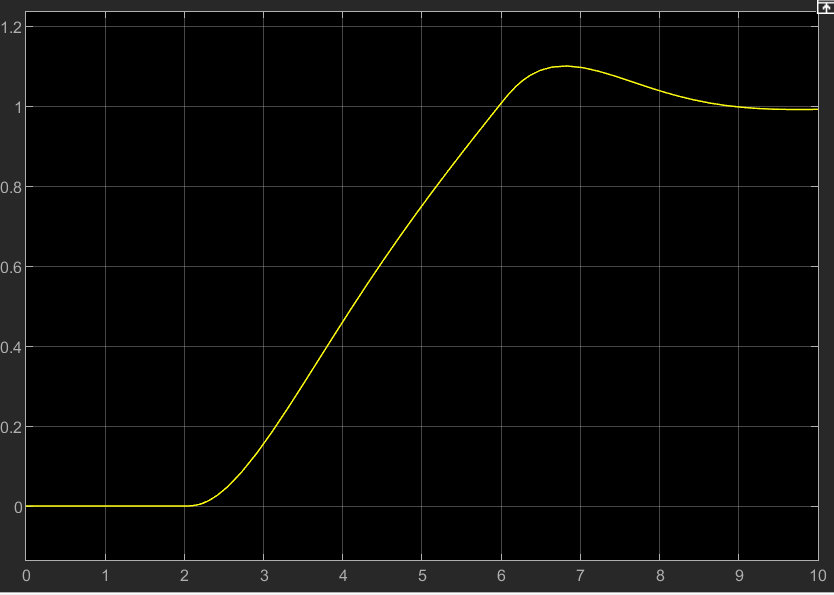


Fig. 6.9. PRH\_PI System Response

The system starts 0.5 meters of the desired height, 0.3 radians off the desired roll, and 0.1 radians off the desired pitch. These are all very large errors, since height should be kept within 0.15 m of 1 m and pitch and roll should be within 0.1 radians of zero, which is why they were chosen to test the system response. The height dips to 0.84 in the tests, outside of the desired range of the system, but this is with a large starting height error and due to resources being given priority to the correcting the large pitch and roll errors. The pitch and roll of the system respond quickly and return to zero in less than a second, and stay well within the 0.1 radian error required. The system can be further improved, but tuning should be used with the help of the VREP responses to the increased accuracy of the system responses.

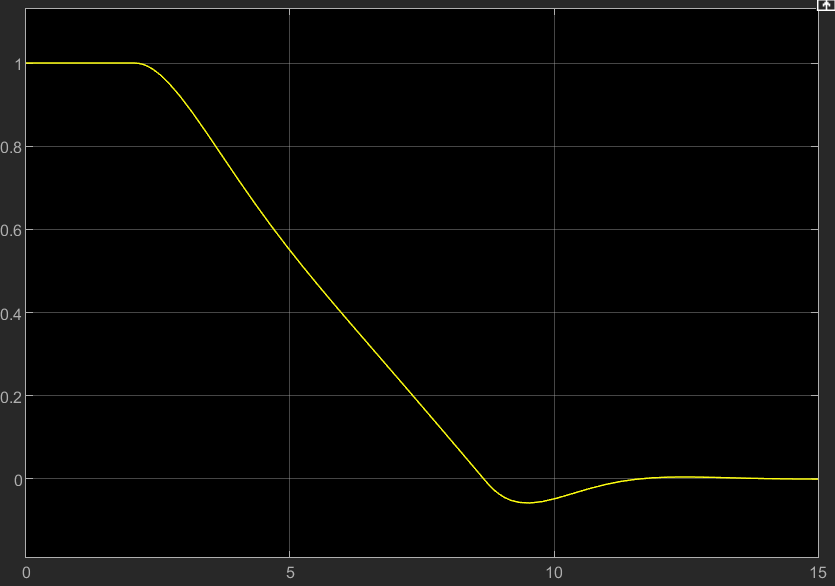
6.7.3 Auto Take-Off and Landing

The auto take-off function is helpful in getting the drone to s safe height before flying, and the auto-landing function can help with a smooth landing. For these functions, the drone only needs to keep the tilt stable and adjust height, so the PRH\_PI regulator can be used as a base, and the height value command varies. Since there is a change of a meter in the height, integrator windup is a problem, so instead of a step input, a ramp input can be fed for a smoother transition and prevent overshoot. Since it only manipulates one value, the rest of the setup remains the same. The ramp function was given a slope of 0.25m/s and maxed at 1 m, and the height response is given in Fig 6.10.

Fig 6.10 Height Response in Auto Take-Off Function

The take off function was delayed slightly for testing purposes, but clearly shows the drone is on the ground, 0 meters, and height increases to 1 meter steadily and with an overshoot of 0.1 meters, within the 0.15 meter requirement. The system response is stable and ready for testing in Vrep.

Similarly, auto landing just uses a ramp function, but goes from 1 meter to zero instead. Also, since the drone can hit something going down, the ramp is given a slighter slope to minimize impact, since overshooting is not possible. The height response is given in Fig. 6.11.

Fig. 6.11 Height Response to Auto-Landing

The drone height command is steady with minimal overshoot. In reality there would not be overshoot but impact instead. Since the velocity before impact is 0.1 m/s, with a 4.5kg drone, the kinetic energy is only 0.05 joules, and not a cause for concern. The system may be tuned more but should be done based on results from a more detailed VREP simulation.

## 6.8 Closed Loop Remote Control Design

6.8.1 Design Overview

STR 3.0.0, Remote Control, states The drone should have RC control implementation to allow for direct control of the drone. The drone can start in this state, or be switched to from autonomous control. STR 3.1.3, Autonomous Functions, states autonomous functions should be accessible in the remote control state, such as terrain tracking, and tilt control was added after the open loop design proved unstable.

To maintain system stability during remote control flight, a few factors need to be considered. First, the tilt angle, the pitch and roll angles, must be kept near zero for the system linearization to remain accurate, as well as ensure the drag will not increase significantly and cause the drone to lose controllability. Terrain Tracking was also a function that is required of the RC Functionality, so height control is needed as well. The user inputs also need to be interpreted in a way such that the motor can respond predictably, but should also not come at a cost to system stability. The order of priority for the system is

1. Tilt Stability
2. Height Control
3. User Response

To achieve the desired results, there's several functions that need to be incorporated into the remote control design to be effective. First, the PRH regulator developed in the auxiliary functions section can be used to maintain the stability of the identified critical states. An RC command interpreter is also required to translate user inputs. A saturator is needed to limit the commands to realizable values. Lastly, a converter is needed to turn the force values into throttle and servo angle commands. The setup is provided below in Fig. 6.12.

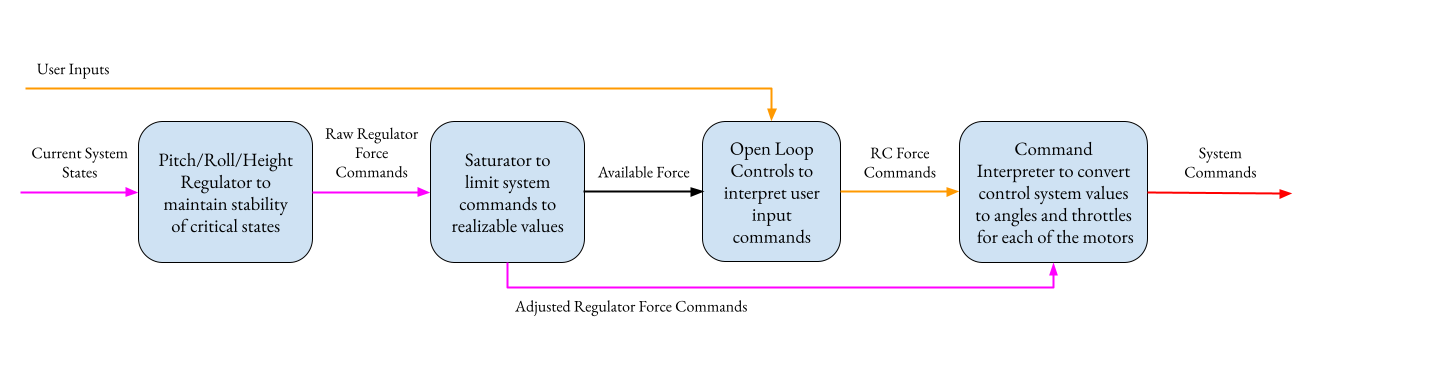


Fig. 6.12. Closed-Loop RC Layout

An output is added to the saturator called available force, which tells the open loop controls block to scale the values of the force commands, ensuring that the RC commands only use “leftovers” of the PRH regulator, and the system can respond to user commands, but never at the cost of system stability. The open loop and regulator force commands are then simply added together before being converted to motor and servo commands.

6.8.2 Adjustments to the PRH Regulator

The PRH\_PI regulator actually already handles the critical states of the drone on its own. The only adjustment that may be needed is to in testing the response of the system to RC commands needs to be tested since that is now an internal disturbance to the drone, something the regulator was not tested against. Also, the open loop controls will exert a moment as shown in the open loop section, and the integral path needs to ensure that it can compensate against the steady state and rapid changes in user inputs.

6.8.3 Saturator Design

A Saturator is needed to ensure values are realizable and can be scaled for proper motor commands. Since the force values are decoupled into their X and Z components, they need to be recombined and tested against the max force value. The max force value can be given as a constant of the system if there is a max value that can not be exceeded, or if a voltage detector is incorporated on the battery, can be given as an input that varies over time. If any values exceed the max value, the max force command set (X and Z component recombined) is used to scale all the force inputs. All the values are scaled together to prevent an unbalanced input from destabilizing the system. If no pair exceeds the max value, the force commands are passed on without modification. Lastly, the saturator outputs the available force after the saturator commands to help scale the open loop system. The full setup is given in Fig. 6.13.

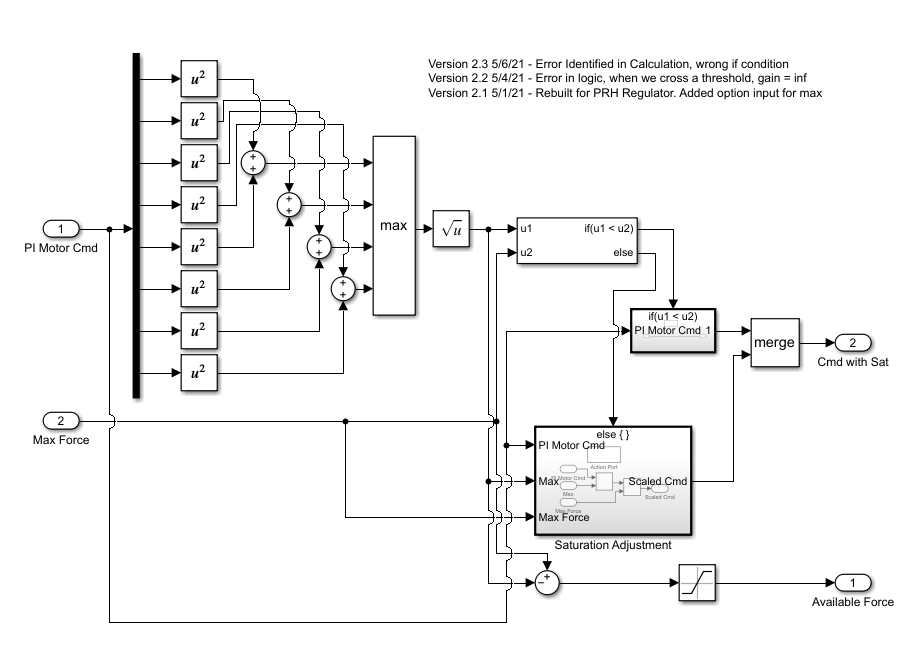


Fig. 6.13. Saturator Design

6.8.4 Open Loop Control Design for Closed Loop Control

In the original open loop design, ascend and descend commands were needed, however, they are not needed since the PRH Regulator will control height, so the function is adjusted.

First the turn and forward commands are set to be between -1 and 1, a common standard for throttle commands. The Forward vector has all X-direction vectors pointed forward to move the drone forward, and the turn vector has the right X-direction vectors as forward and the left X-direction vectors as backwards to turn the drone. The Z-Components are all kept at zero for three reasons: first, the PRH Regulator will correct for errors in tilt caused by the moment generated by applying force in the X-direction, and second, at steady state, the drag and propulsion systems will apply equal force at the center of the lift bag due to their mounting, creating near zero net moment, and third, it is easier to scale the value to max throttles since the distance formula won't be needed like in the saturator.

Next, the commands should be combined to create a single command. To combine, first the throttles can be added, and the combination is scaled by the inputs so they’re balanced. The issue now is the value can be greater than 1, not a standard throttle command. The motor values then have their max values checked, and if any value is greater than 1, all the values will be scaled so the largest value is equal to 1. All the values will be scaled the same to preserve the forward and turn ration as provided by the user.

Finally, since the max value is one, the throttle can be directly multiplied by the available force input to scale the force commands of the open loop system to the max force that can be used, and the value can be passed to the rest of the system.

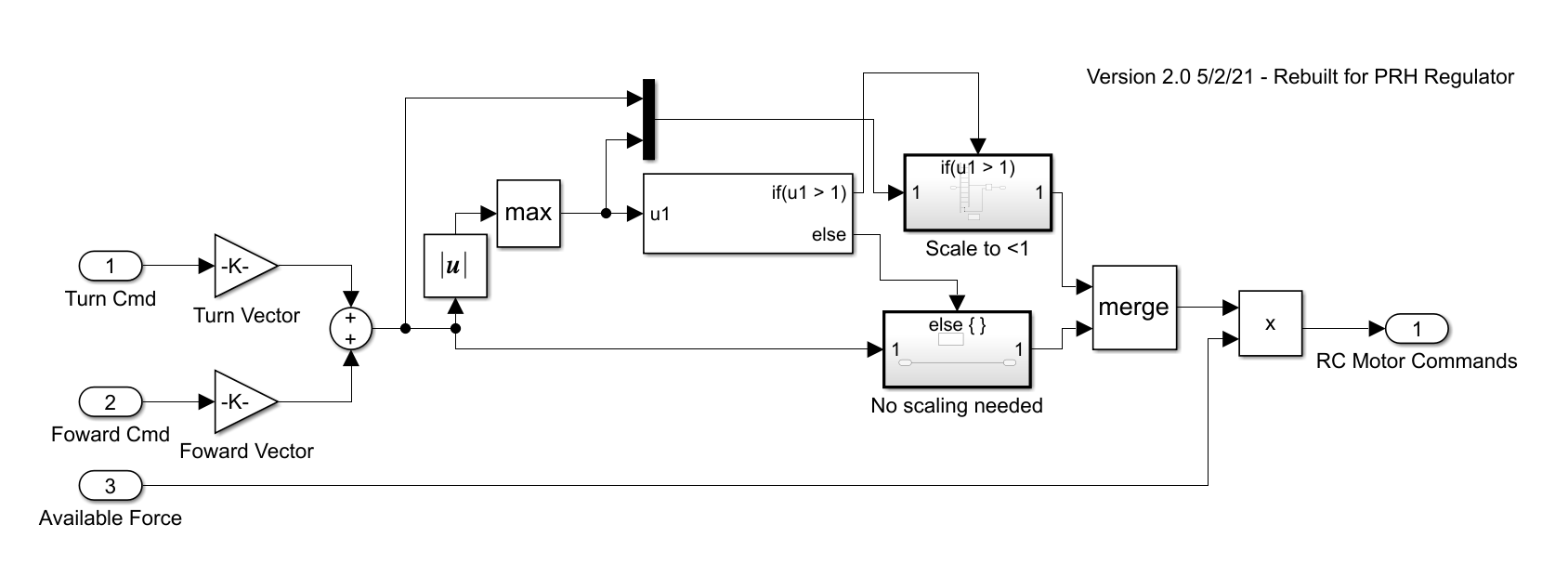


Fig. 6.14. Open-Loop RC Design

6.8.5 Force to Throttle/Angle Converter

To help linearization of the system, the components were decoupled and the servo angle was ignored. To output commands to the system, the force commands need to be converted to throttle and servo angle commands.

Since the saturator and scaled open loop commands are already within the force capabilities of the system, the vectors can be used to calculate the servo angles using the arctangent function with atan(Fx/Fy), and the throttle value determined by comparing the ratio of the max force of the system and the combined force values of the X and Z components. The only limit is the atan function has a range between -pi/2 and pi/2, so the X-component is checked to check the angle quadrant, and a value of pi can be added to shift the servo angle to the proper quadrant. Due to the size of the block, and since the math itself is simple, it is included in the appendix for reference, but does not offer much more information.

6.8.6 Closed-Loop RC Integration

All the components are finally combined to generate the full Closed-Loop RC system as shown in Fig. 6.15.

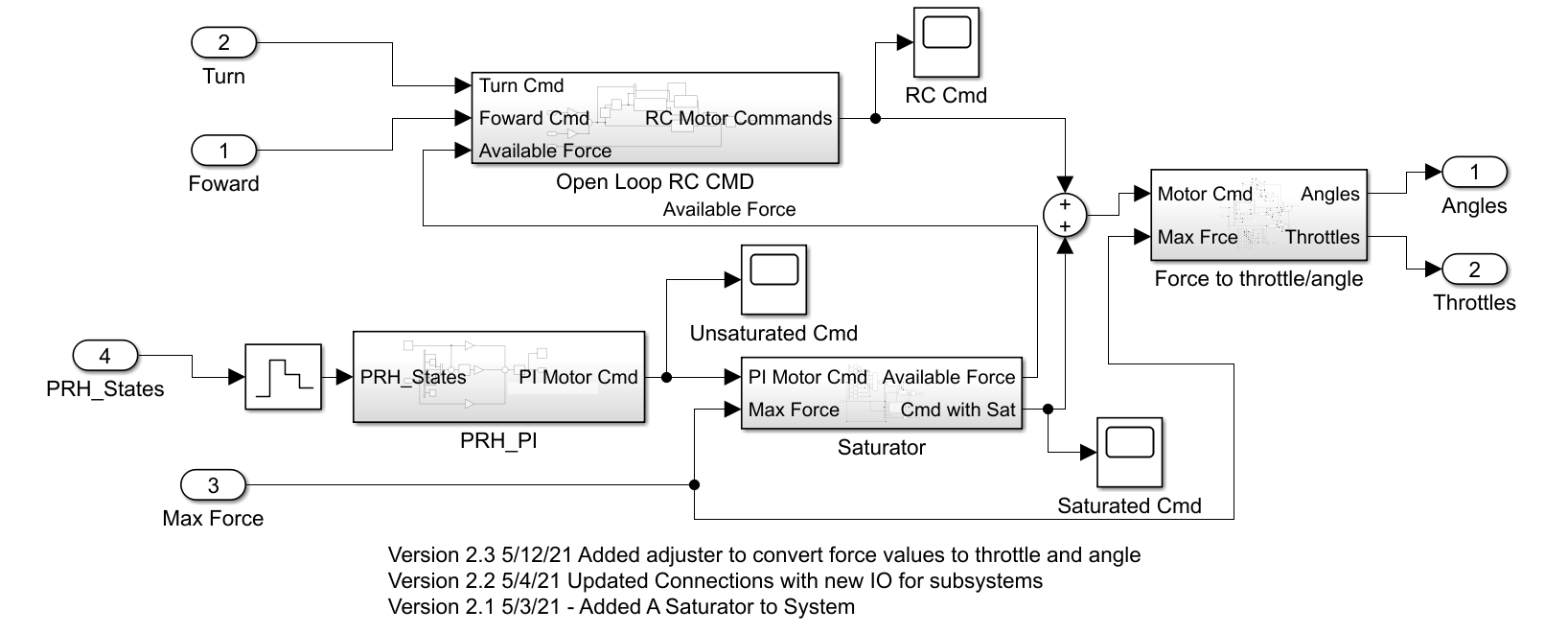


Fig. 6.15. Closed-Loop RC, Full System

The Full system follows the general setup as laid out in Fig. 6.12. The states are fed into the regulator to get the first set of force commands. The values are passed into the saturator block along with the max force value to scale the commands to realizable forces, and returns the available force to the Open-Loop RC block. The Open-Loop RC then interprets turn and forward commands into a throttle to be scaled with the available force input. Then, the regulator and open-loop RC commands are combined and pass through the Force to Throttle/Angles block to be converted into commands for the system.

6.8.7 Closed-Loop RC Verification in MATLAB

Since the open loop design failed to meet design specifications, STR 3.4.0, Closed-Loop RC was added, and requires a system that maintains pitch of roll error of <0.1 radians and a height error of less than 0.15 radians. The system was verified through matlab simulation, however, it is not fully detailed like the Vrep simulation environment. To be tested, the force to throttle/angle converter was removed and the force commands were tested with the State Space Matrices defined in section 6.7.2, Pitch/Roll/Height Regulator.

The PRH regulator and its stability was already tested in the auxiliary functions section, so the system stability response to the RC commands was what needed testing. The worst input response from users would be full stop to full forward, since the pitch moment would be greatly affected from the propulsion systems positioning onboard the drone. To test, the drone is given a forward throttle command of 1, and after 1 second, the command is immediately changed to -1, and the effects on the pitch roll and height are given in Fig. 6.16. At one second, the pitch value jerks as expected, and the pitch never exceeds 0.06 radians, well within the defined 0.1 limit. The height seems like a small fluctuation but almost imperceptible, especially when compared to the 0.15 m error allowed.

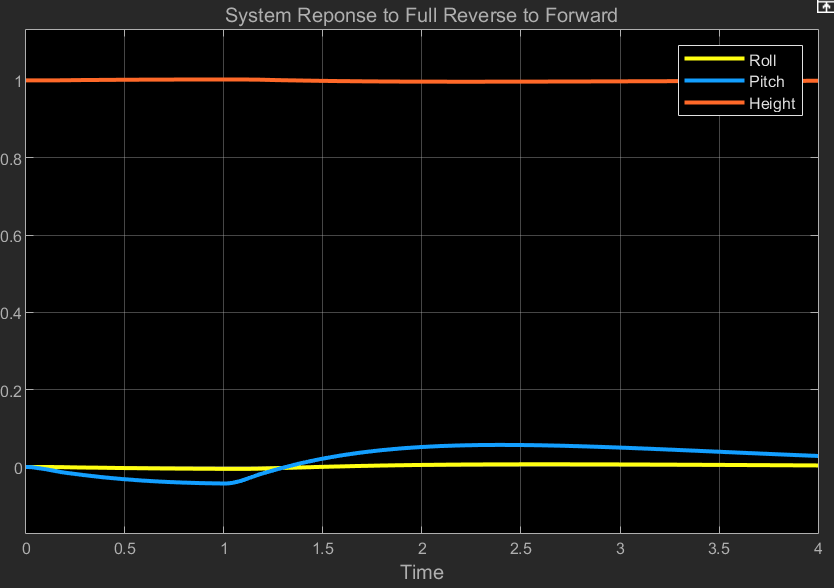


Fig. 6.16. Critical States Response to RC Input

The system maintains full stability even with the worst case scenario tested in MATLAB. The values can be tuned to have a better response with some trial and error, but will be fine tuned with using the system responses in VREP as more accurate guides to the system response.

The system should be tested and verified in VREP, especially since the physical model is more accurate than what is used in MATLAB, but the simulation communication is incomplete, so it is not verified in the environment. The simulation layout and testing plan is laid out in section 9.whatever

## 6.9 Autonomous Design

Autonomous design was required as stated in STR 4.0.0, Autonomous Control. The system plants were defined by adding yaw and north east components using equations 6.3 and 6.10, and the small angle approximation was used to simplify the rotation matrix, but only for pitch and roll since the yaw angle could not use small angle. The yaw angle included using gain scheduling to allow for the full 360 degree use, but did not seem like a good approach. Some work was started on instead continually updating the next waypoint to be defined in the drone frame coordinates, which would require constant updates, but gain scheduling would be vastly simplified to only a few sections instead of a full 360 degrees, but no significant progress was made. The requirement was not met.

## 6.10 Error Identification and Response

6.10.1 Crash Detection

The drone has an 9-DOF IMU sensor to detect the orientation of the drone, which is required by STR 4.3.1, Error Handling By the IMU Sensor. The IMU is constantly sending back information about the drone’s orientation, and can detect if a sudden change in orientation occurs, meaning that the drone has begun tumbling. The IMU sensor has been implemented and verified as seen in Chapter 5, so it meets STR 4.3.1, Error Handling By the IMU Sensor.

6.10.2 Deflated Bag

Onboard the drone is also a pressure sensor to determine if the lift bag deflates. STR 4.3.3, Popped Balloon Error Case requires the drone to make an emergency landing if the lift bag rapidly loses volume. Currently, the pressure sensor is not implemented in software and has not been tested, so STR 4.3.3, Popped balloon was never verified. Ideally, the drone would be able to keep track of the pressure of the lift bag, and be able to detect a drastic change in pressure. However, with the current physical design of the drone, the propulsion modules are held in place by the lift bag inflation, and if the bag deflates, there is no way to controllable land since there is no way to know the propeller positions. The loss of rigidity of the propulsion modules is shown in the flight test section, and the drone is too uncontrollable, so the drone fails STR 4.3.3, Popped balloon.

## 6.11 Conclusion

The controls design was verified in matlab for STR 3.4.0, Closed-Loop RC, and STR 3.1.3, Autonomous Functions, but not functions were fully verified in VREP. The autonomous functions were not completed or tested. The error detection and automatic responses were also not completed. Based on the MATLAB verification, the drone is controllable and a full state feedback PI controller is possible to incorporate on the drone, but controls as a whole failed to meet the system requirements.

# Chapter Bibliography

We do have a full bibliography that should absolutely be updated with all content here. The point of the chapter bibliography is to help keep track of citations in the chapter since the numbering may change in the full bibliography with changes and additions. This way will isolate the sources in this section so you can cite here without having to worry about it, and can use a simple find and replace on your citations to update the new numbering when we combine everything in the final report.