Chapter 8: Full System Simulation and Validation of Drone Design

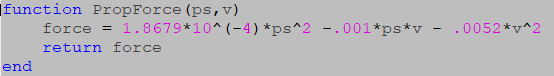
# Chapter 8 Outline

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 8: Full System Simulation and Validation of Drone Design

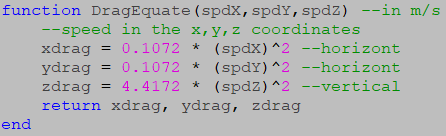
Section 1: Necessary Forces for a Practical Simulation Environment

* Covers how the buoyant force is applied and its implications
  + Buoyancy is much more complicated than a simple simulation can handle due to the nature of fluid dynamics that affect buoyancy:
    - Density of a fluid
    - Volume of liquid displaced
    - Local acceleration of uneven surfaces
  + Due to fluctuations of real fluids and inaccuracy in our ability to simulate these properties, we rely on the most simple, symbolic form of the buoyant force, a single upward vector
  + To apply this buoyant force, a lift bag object is mounted within the larger envelope to imitate the physical design. Not only are the masses of lift bag and envelope separate, providing their own local areas of gravitational forces, but the buoyant force is also shifted higher than the center of mass, essentially providing a buoyant moment
* Individual gravity forces on individual parts due to the working CAD model.
  + Individual gravity is applied directly to the center of that object, assuming an evenly distributed part. Parts that have a mass assigned to them include:
    - Envelope
    - Lift bag
    - Gondola including all parts inside of it
    - Ultrasonic
    - Corner attachment with servo weight
    - Servo shaft
    - Propeller with motor
  + These parts additionally are provided with surrounding respondable objects that react to collision according to the momentum they are provided.
* Covers the implementation of propeller forces, the equations used, and tests done.
  + Propeller forces are non-linear and adjusted entirely by the propeller speeds they are provided.
  + We use this equation to calculate the forces required, simplifying all coefficients and adjusting them to scale alongside the max thrust force:



‘ps’ represents the propeller speeds and ‘v’ the wind velocity

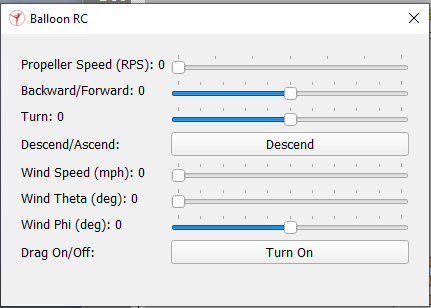
* + The forces and speeds are applied to the respondable propeller objects seperately, since once again buoyancy is not inherently within the sim so these forces must be manually inputted.
  + To test that these forces were working, we turned the propeller speeds to max on individual propellers so we would know that they provided an accurate torque to the overall drone.
* How we implemented drag forces and analysis for tests for each force.
  + Drag forces were completed in two distinct areas, drag forces due to a wind’s vector force and drag forces due to the air displaced by the movement of the drone
  + In order to change the wind’s direction to point in every direction we chose, the coordinates were aligned on a spherical axis with wind speed changing along the radius, theta along the x-y plane, and phi along the z-axis.
  + Wind is then converted to rectangular coordinates and applied to the drag equation.
  + The equations are simplified to a single coefficient and written as follows:



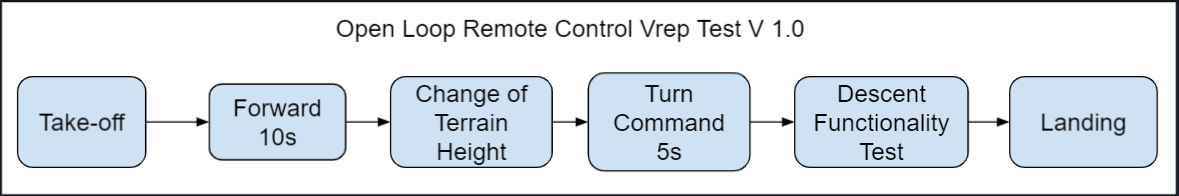
* + Drag force due to air displaced in the drone’s direction uses the same equation to calculate drag force; however the speed is replaced using the velocities of the drone and the force applied opposite to the direction the drone is moving.
  + By applying both of these methods, a drag force can be calculated and applied to all directions of the drone.
* By operating both drag forces and propeller forces together, the drone demonstrates capability of moving forward within a specified quantity of headwind.

Section 2: Open Loop Remote Control Implementation

* Technical Requirements
  + 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
  + The software shall be fast enough to respond quickly to all user commands and error handling.
* Input layout in GUI included below:



* + As mentioned in the previous section, the propeller speed and wind controls, illustrated above, control the propeller force and wind vector, respectively.
  + However, the propeller forces are not fully completed without adjusting the propeller force directions.
* Open loop remote control functionality, testing, and analysis of results
  + As previously mentioned in the controls system chapter, open loop control was designed to change the direction of the propellers using factors for thrust and turn.



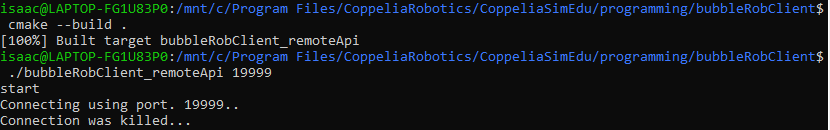
* + Before conducting the test, drag should be turned on.
  + Starting at the left side of the simulation platform, take off is initiated and propellers speeds are set to max throttle until a height of 5m is reached.
  + Using the forward slider, the propellers are tilted forward at around 60 percent for optimal lift as the drone moves forward.
  + To overcome obstacles in the path of a drone’s flight, either propeller speeds or propeller direction is adjusted.
  + A turn command is then issued by increasing or decreasing the turn factor slider, which causes the balloon to rotate either clockwise or counterclockwise.
  + Descent functionality flips the propeller controls across the x-y plane, which could be used if the drone encounters heavy updrafts of wind.
  + At the end of a test, landing the drone requires utmost precision when adjusting propeller speeds in order to reduce vertical velocity as close to zero as possible before landing.
* Analysis of tests
  + Under no wind conditions, drone flight was seen to be wobbly due to an uneven distribution of mass between the gondola and envelope. The drone was capable of making wide turns with slight difficulty, which was made possible with multiple turn commands.
  + With maximum wind conditions, steering the drone using entirely open loop controls was very difficult. Since the drone relies on its ability to move in the direction of either forward or backward, slight misdirections due to wind can cause the drone to veer away from its intended direction. With a large central mass, rotational momentum can be difficult to correct. Therefore, once the drone begins spinning, propellers will need to correct by applying a torque in the opposite direction of the drone.
  + Closed loop remote control is needed to be implemented in order for the drone to be able to withstand faster wind conditions and make wide turns.

Section 3: Sensor Array Implementation

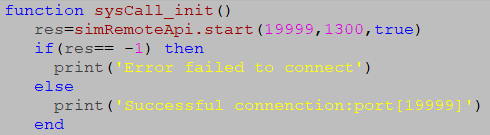
* Technical Requirements
  + The sensors shall be able to monitor the area in front of the drone inorder to maintain a constant height of 1 m. Physical location and sensor sensitivity affects effectiveness
* Noisy sensors and the accuracy of state estimation on board the drone.
* Sensor data recorded for the drone included data from the barometer, gps , ultrasonic, and IMU.
* In order to account for the noise during actual flight and test our RC response to the varying tolerances of the sensors, we decided to add noise to these sensors by applying a Gaussian distribution centered around ideal values.
* Provided a variance and mean, our function outputs a randomly generated number, which is located along the gaussian curve.
* Variance was determined by data sheets with the following values:
  + Barometer : ± 0.4 kPa
  + GPS : ± 50m (CEP)
  + Ultrasonic : ± 3mm
  + IMU: ± 1.5%
* Two standard deviations are recorded for each value meaning pseudo noise provides values 95% of time within the given values of noise.
* Once a sensor array is complete with the appropriate considerations for noise, sensor data could be collected and employed within a closed loop remote control system to maintain further stability

Section 4: Closed Loop Remote Control Implementation

* Technical Requirements
  + 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
  + The software shall be fast enough to respond quickly to all user commands and error handling.
* Although we were not able to complete the closed loop remote control, we have listed out the tasks we were able to perform and possible future testing that could be done on a completed RC system.
* Setting up a remote application programming interface(API) in V-rep and Virtual Studio Code(VS Code)
  + In order to fully test a control system, the system must be applied to a functioning drone design. Since physical testing was incomplete, V-rep seemed to be the next best option.
  + Interfacing V-rep with Matlab code, converted to C++ for its capability to compile on a remote controller, proved to be a much more daunting task than we expected.
  + Compiling C++ on a Windows OS first of all requires an Ubuntu terminal to run gcc commands. However, Ubuntu does not run on Windows OS; therefore, the Linux OS is required to be installed prior to running any terminal commands.
  + In order to connect the server side, V-rep, to our client program, Visual Studio Code, we needed to include a number of required V-rep files into VSC. Due to vague or unspecified instructions on outdated V-rep user manuals[5[2](#xogr7kxu7qa1)], when the required files were found, further files from further directories within V-rep needed to be included as well. To solve the problem of missing file inclusions, we end up editing an example file, which implements a remote API, written directly in the V-rep source folder, which is why the file retains its former name, “bubbleRobClient”.
  + Finally, by using Cmake to build a makefile, our program is compiled directly by calling the file and specifying the port number, to which V-rep has connected to, while it is running.
* An example of a successful remote API connection:



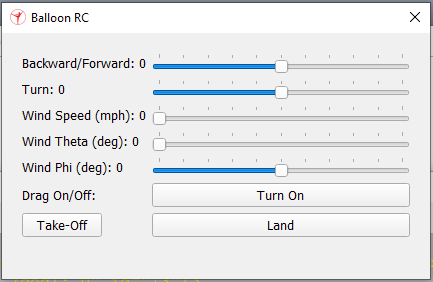
Code snippet taken from an Ubuntu terminal showing compilation of code written in VS Code and a connection to the port, 19999.



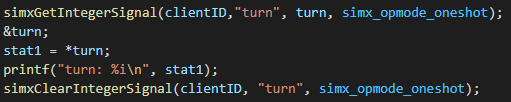


Code snippet and terminal output shown in Lua, V-rep’s embedded scripting language, managing the connection to port 19999.

* Implementation of the closed loop remote control function
  + For the purposes of V-rep implementation, a simplified understanding of closed loop remote control is necessary to be established.
  + The closed loop function, described in further detail in Chapter 6, takes in the four inputs: forward; turn; maximum force of propellers; and an array of six variables listed as pitch, roll, height and their corresponding derivatives, alternatively known as “PRH states”.
  + And the output of the RC function provides the necessary angles and throttles of each individual propeller for the drone to operate with.

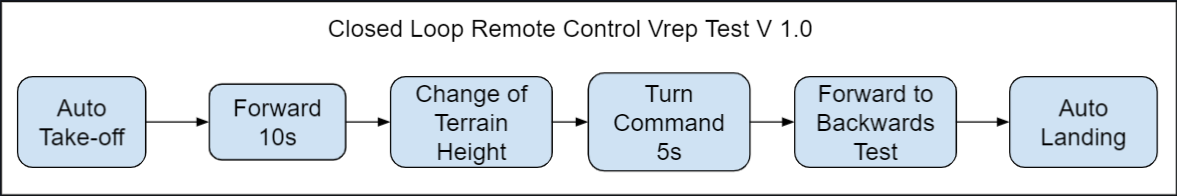


* + A completed closed loop remote control removes the need for a propeller speed variable, which is illustrated above showing the current implemented GUI in V-rep. This system allows the drone to move with only two commands and corrects for external factors that cause the drone to tilt at unspecified angles.



Remote API functions are written to pass data between server and client.

* Hypothetical closed loop remote control testing



The block diagram above shows how a test would be conducted using a closed loop control system.

* + To take the drone off the ground an automatic take-off sequence is started with the press of the Take-off feature on the GUI. This feature causes the propellers to start up and increase thrust to maximum.
  + Then the forward throttle is increased to maximum; much like open loop remote control, the propellers face in the ‘forward’ direction and move so.
  + A change of terrain height is then presented to the drone and, with the barometer and ultrasonic sensors constantly feeding back height data, an increase in throttle upward allows the drone to be able to avoid all obstacles that may potentially collide with the drone.
  + After clearing all obstacle below, the drone is issued a turn command for five seconds to test by
  + Next, we have the forward to backwards test. This test assesses a drone’s RC response to a significant amount of unforeseen pitch and pitch velocity, by switching the throttle back and forth in quick succession, to account for collisions, heavier than normal winds, or simple piloting errors.
  + Finally at the end of the experiment, the autonomous landing feature is tested with the press of the button on the GUI, and the propellers, given the data of height and vertical velocity, correctly adjust propeller speeds to zero before the drone touches the ground.
* Due to our limited programming knowledge in C++ and running server and client commands and in the absence of a fully developed or updated V-rep user manual for remote API, closed loop remote control implementation was unable to be completed on time.
* Given more time to work with interfacing V-rep with VS Code, closed loop remote control may have been completely completed and tested. However, for reference, when seeking help from our former TA, [Alexey Munishkin](mailto:amunishk@ucsc.edu), who has had previous experience with remote programming in V-rep interfacing with Matlab, he had told us that learning remote API had taken him close to a year to do.
* In retrospect, Gazebo, another robotic operating system we had in mind when deciding on a simulation environment, may have been better suited for operating our drone’s model due to its support of fluid dynamic principles, specifically buoyancy and fluid density. The primary justification we had in mind when choosing V-rep over Gazebo was that Gazebo could not operate under a Windows operating system and was required to be installed through Ubuntu packages on Linux. The irony of this decision lies in the fact that in order for the remote API to compile C++, Ubuntu had to be installed.

Section 5: Verification of Physical Dimensions

Based on new fabrication model V-rep simulations were run

# Chapter 8 Draft

A drone simulation was needed to best replicate real world conditions within a controlled environment, so we chose V-rep as our simulation platform. Within this chapter, we will explain how we designed our system and the functions we implemented to apply internal buoyancy, external drag, and propeller forces. Using these fundamental forces, we were able to create a full system design of our drone with a rudimentary open loop remote control system, which allowed us to verify whether we would meet our STR 2.0.0, Minimum Drone Speed. Though we were not able to implement the closed loop remote control, we have listed out the tasks we were able to perform and possible future testing that could be done on a completed RC system. Overall, the simulation chapter also covers physical responses to remote controller inputs, noisy sensor inputs, and control system responses to identify necessary flight responses before physical testing to inform of any prerequisite design changes.

## Section 8.1: Choosing a Full System Simulation Environment for the Buoyant Drone

Before simulation began we looked at a variety of options for simulation environments where we would be able to test our drone to its full capacity. We ended up finding two programmable simulation environments that were open source and the most recommended for beginners, since nobody on our team had any simulation engineering experience. As our first primary choice, Gazebo had many of the great features V-rep included, such as sensor integration in robotic hardware, testing multiple robotic controls simultaneously, and an active developing community for newer users to be able to ask questions. But the drawbacks that kept us from using Gazebo were primarily that the installation of Gazebo required Ubuntu packages making it very difficult to run on Windows and CAD files could not be imported into Gazebo, whereas Unity was used instead.

On the other hand, while a week was spent trying to get Gazebo to install on our device, V-rep had taken only a single day. Although V-rep utilizes a robust real-time physics engine to simulate a diverse multitude of robotic designs, its support for aerodynamic features, an essential parameter our design takes advantage of, is fairly lacking. Due to the constraints of time spent on our simulation design, we ended up deciding on V-rep as our platform of choice, and so our implemented solution to the lack of aerodynamic support is provided in the following section.

## Section 8.2: Necessary Building Blocks for a Practical Simulation Environment

Within the simulation we chose the following forces to closely represent real world conditions: buoyancy, gravity, thrust, and drag. The implementation of each of these forces will be explored in further detail within this section.

8.2.1 The Buoyant Moment  
 Buoyancy is the essential lifting force of our drone, derived from the lift bag. However, one specific keynote to take into consideration when adding buoyancy is that the physics engines in V-rep were not built to handle the complicated nature of aerodynamics principles, such as the density of a fluid, volume of liquid displaced, and local acceleration of uneven surfaces due to buoyant factors. Simply stated, an atmosphere in V-rep does not exist.

Due to the inaccuracies in our ability to simulate these properties of real-life fluids, we rely on the most simple, symbolic form of the buoyant force, a single upward force vector. And to apply this buoyant force, a lift bag object is mounted within the larger envelope to imitate the physical design. The purpose of this design feature is not only to keep the masses of the lift bag and envelope separate, allowing their own local areas of gravitational forces, but also to shift the buoyant force higher than the center of mass, creating a buoyant moment in the correct location.

8.2.2 Individual Gravity Forces and Momentums

Given a working CAD model with various interconnected parts, masses, and individual orientations, individual gravity forces can be applied locally on a full system model. Assuming evenly distributed parts, these individual gravity forces are applied directly to the center of that object. All parts are assigned masses to them, which include the envelope, lift bag, gondola with all electronics and hardware within it, ultrasonic, propulsion system mounts along with the embedded servo weights, D-shafts, and the propellers with their attached motor weights. In addition, these parts are provided with surrounding respondable objects that react to collision according to the momentum they are moving with.[5[4](#94m4jh9x0h)]

8.2.3 Implementing Propeller Forces

Propeller forces are implemented as a secondary lifting force we are able to control using the propulsion system. Therefore to make the propeller forces realistic,the propeller forces written are non-linear and adjusted entirely by the propeller speeds they are provided. We use this equation to calculate the forces required, simplifying all coefficients and adjusting the coefficients to scale alongside the max thrust force. The equation in figure 8.1 is based on(4.1), where the constants are defined by our propellers dimensions. For more information about the propeller force equation see Chapter 4.

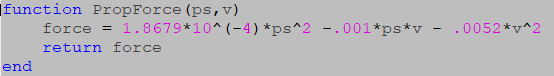


Fig. 8.1. Propeller Force Function where ‘ps’ Represents the Propeller Speeds and ‘v’, the Wind Velocity

The forces are applied to the respondable propeller objects and speeds are applied to a non-respondable propeller object separately, since once again buoyancy is not inherently within the sim so these forces must be manually coded to work as intended. To test that these forces were working, we turned the propeller speeds to maximum on individual propellers so we would know that they provided an accurate torque to the overall drone. When tested, each corner provided the appropriate amount of torque to rotate the drone in the direction that a propeller pointed, along the curvature of the envelope, from one corner to the opposite.

8.2.4 Implementing Drag Forces

As a factor limiting our drone’s capabilities, drag forces were implemented as two distinct forces in two distinct areas, drag forces due to a wind’s vector force and drag forces due to the air displaced by the movement of the drone. In order to change the wind’s direction to point in any direction we chose, the coordinates were aligned on a spherical axis: wind speed changes along the radius, theta along the x-y plane similar to yaw, and phi along the z-axis much like pitch. The vector in spherical coordinates is then converted to rectangular coordinates and applied to the drag equation, (2.2) and as shown in Figure 8.2, providing individual drag forces in every direction. The coefficients were based on the dimensions found in section 3.1.4.

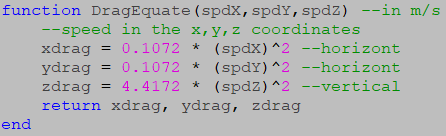


Fig. 8.2. Drag Equate Function Showing the Drag Force Equations in Cartesian Coordinates.

The drag force equations in the ‘DragEquate’ function, figure 8.2, are simplified to a single coefficient and variable of airspeed as shown above. Drag force, due to air displaced in the drone’s direction of motion, uses the same equations to calculate drag force applied by the wind; however, the variables of velocity in three directions are replaced using the velocities of the drone and the force is applied opposite only to the direction the drone is moving. By applying both of these forces separately, three dimensional forces can be applied on cartesian coordinates without the assistance of a rotational matrix at whichever direction drone or wind is specified. By operating both drag forces and propeller forces together, the drone demonstrates capability of moving forward within a specified quantity of headwind.

## Section 8.3: Open Loop Remote Control Implementation

In this section the attempted verification of the following STRs will be discussed:

**2.0.0** Drone shall be able to fly at least 5mph in winds up to 15mph.

**3.0.0** 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.

8.3.1 Open Loop Remote Control GUI Explained

In an open loop remote control, propeller speeds were adjusted manually and for all motors the same propeller speeds were applied.

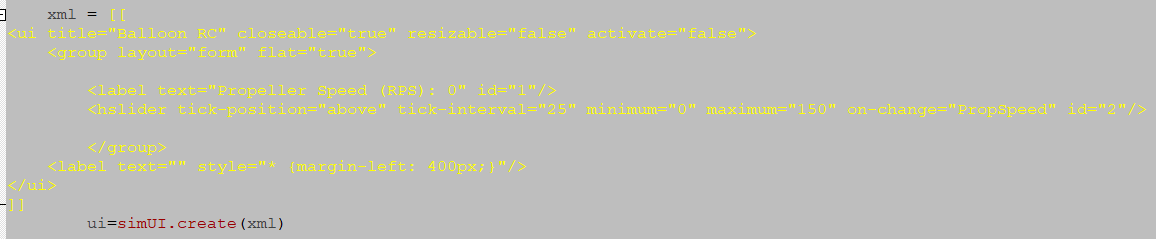


Fig. 8.3.a. Example Code Snippet, Which Shows how a GUI Slider for “Propeller Speed” is Created Using XML Code in V-rep.

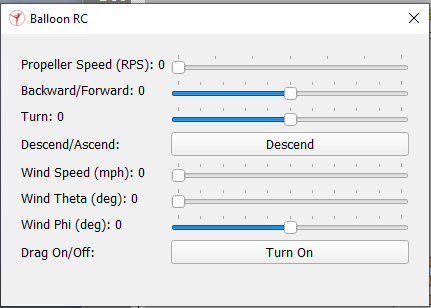


Fig. 8.3.b. Input Layout of Open Loop Control is Generated in GUI Format as Shown Above.

As mentioned in the previous section 8.2, the propeller speed and wind controls, illustrated above, control the propeller forces and wind vector, respectively. However, the propeller forces are not fully completed without adjusting the propeller force directions. Therefore, an open loop control must be created to test the drone within varying flight conditions. As previously mentioned in the controls system Chapter 6, open loop control was designed to change the direction of the propellers, by using factors for thrust and turn to rotate the servos in pairs.

8.3.2 Open Loop RC Testing

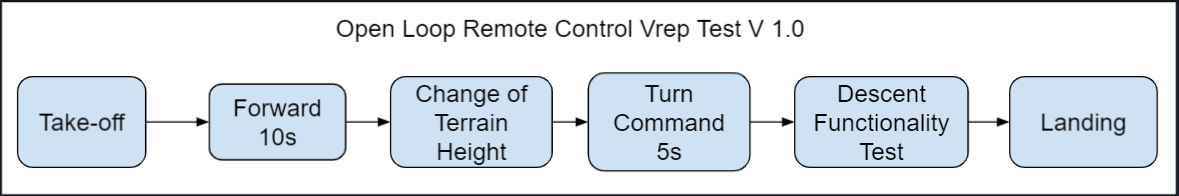


Fig. 8.4. Block Diagram of a Series of System Level Tests

The block diagram, shown in figure 8.4, shows the order in which separate tests are performed to verify both a working drone model and open loop control system. Before conducting a test, drag should be turned on. Starting at the left side of the simulation platform, take off is initiated and propellers speeds are set to max throttle until a height of 5m is reached. Using the forward slider, the propellers are tilted forward at around 70 degrees at maximum propeller speeds for optimal lift capacity, when the propeller force’s vertical acceleration cancels out its gravitational acceleration.

(8.1)

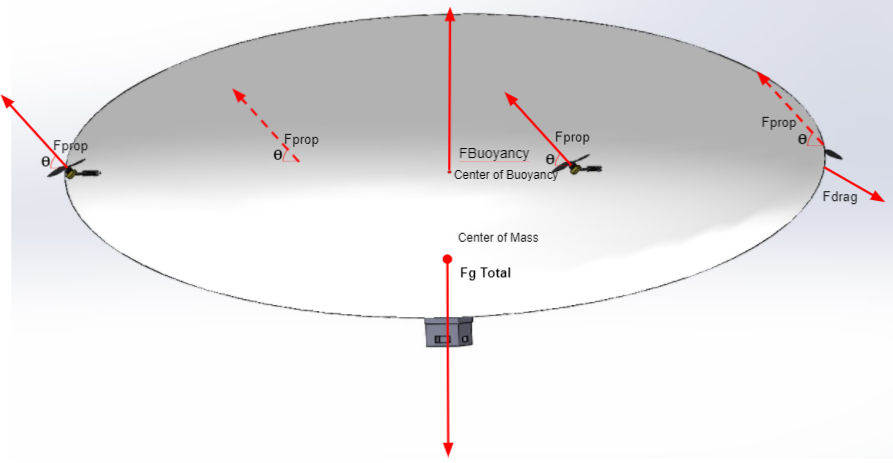


Fig. 2.1. Force Diagram

The drone is then presented with a change in terrain to test its ability to overcome obstacles in a drone’s flight path, either by adjusting propeller speeds or propeller direction. A turn command is then issued by increasing or decreasing the turn factor slider, which causes the balloon to rotate either clockwise or counterclockwise. Descent functionality flips the propeller controls across the x-y plane, which could be used if the drone encounters heavy updrafts of wind. And at the end of a test, landing the drone requires utmost precision when adjusting propeller speeds in order to reduce vertical velocity as close to zero as possible before landing.

8.3.2 Open Loop RC Analysis of Tests

Under no wind conditions, drone flight was seen to be wobbly due to an uneven distribution of mass between the gondola and envelope. The drone was capable of making wide turns with slight difficulty, which was made possible with multiple turn commands.

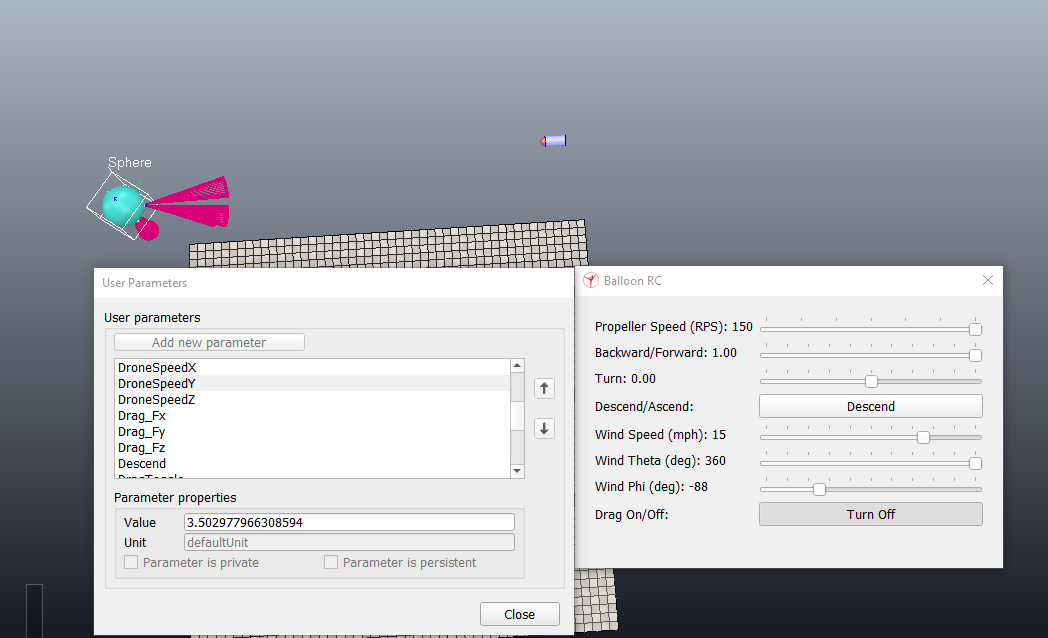


Fig. 8.5. Drone Speed is Shown to be 3.502 m/s on the left column reading ‘User Parameters’ during a flight test with drag turned on

With the throttle and wind speed set to 1.00 and 15mph, respectively, on the right hand RC GUI panel, it was possible to verify that our drone could fly at least 5 mph within a 15 mph headwind, This verified STR 2.0.0, Drone Speed. for our drones optimal design. At the same time with maximum velocity wind conditions, steering the drone using entirely open loop controls was very difficult. Since the drone relies on its ability to move in the direction of either forward or backward, slight misdirections due to wind can cause the drone to veer away from its intended direction. With a large central mass, rotational momentum can be difficult to correct. Therefore, once the drone begins rotating, propellers will need to correct by applying a torque in the opposite direction of the drone. Though it is possible to correct for drone movement using an open loop remote control system, closed loop remote control needs to be implemented in order for the drone to be able to withstand faster wind conditions and make wider turns with less corrective measures needing to be issued.

## Section 8.4: Sensor Array Implementation

In this section the attempted verification of the following STRs will be discussed:

**4.2.1** The sensors shall be able to monitor the area in front of the drone in order to maintain a constant height of 1 m.

**4.2.2** The sensor shall be able to monitor altitudes above 4m for drone altitude awareness.

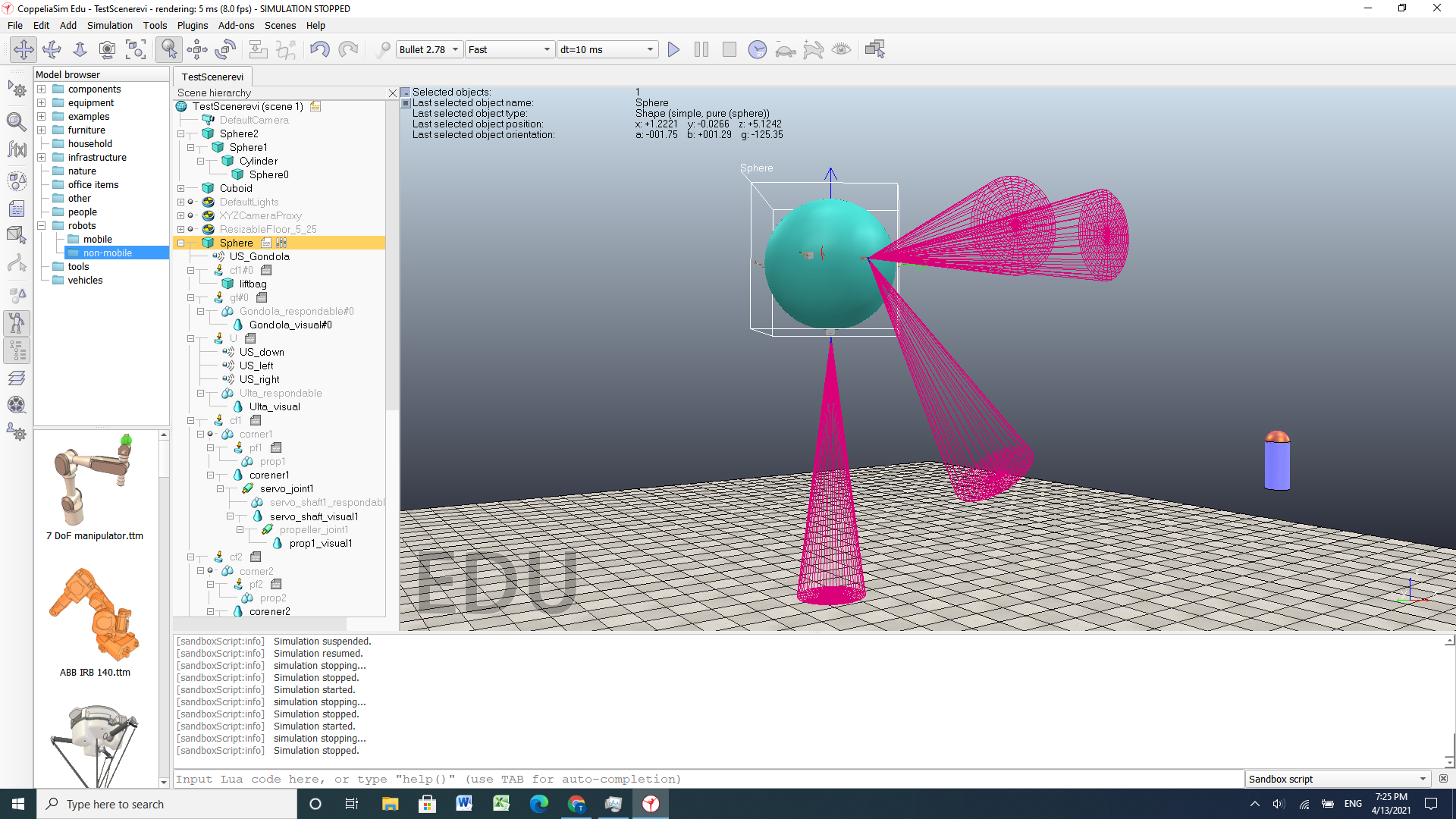


Fig. 8.6. Drone in V-rep Simulation Shown with Ultrasonic Sensors.

Sensor data was recorded for the drone, which included data from the barometer, GPS, ultrasonic, and IMU. In order to account for the noise during actual flight and test our RC response to the varying tolerances of the sensors, we decided to add noise to these sensors by applying a Gaussian distribution centered around their ideal values to simulate more realistic data for our control system. Provided a variance and mean, our function outputs a randomly generated number, which is located along the gaussian curve. The variances we found were determined by data sheets with the following values: Barometer: ± 0.4 kPa; GPS: ± 50m (CEP); Ultrasonic: ± 3mm; and IMU: ± 1.5%.

Two standard deviations were recorded for each value meaning pseudo noise provides values 95% of time within the given values of noise. Once a sensor array is completed with the appropriate considerations for noise, sensor data could be collected and employed within a closed loop remote control system to maintain further stability.

## Section 8.5: Closed Loop Remote Control Implementation

In this section the attempted verification of the following STRs will be discussed:

**3.0.0** 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.

**3.1.0** The software shall be fast enough to respond quickly to all user commands and error handling.

A closed loop remote control would allow our drone the ability to navigate itself with greater controllability in high velocity wind conditions, due to the self correcting characteristics of PID controllers. Although we were not able to complete the closed loop remote control implementation , we have listed out the tasks we were able to perform and possible future testing that could be done on a completed closed loop RC system.

8.5.1 Implementing a Communication Network Between V-rep and Externally Written Code

One of the first tasks to implement a closed loop remote control in V-rep is establishing a communication network using remote application programming interface(API) so that V-rep could interact with our control system code written outside V-rep. In our case, the external code was written in Matlab. However, interfacing V-rep with Matlab code proved to be a much more daunting task than we had anticipated. Simulink, the native language of Matlab, is incompatible with remote controlled code and therefore needed to be converted to C or C++ for its capability to compile with the greatest run speed on our microcontroller. Before compiling C++ on a Windows OS, first of all, an Ubuntu terminal is required to run gcc commands. However, because the Ubuntu OS does not run on Windows, Linux is required to be installed prior to running any terminal commands.

Once all packages are installed and Ubuntu is able to launch, in order to connect the server side, V-rep, to our client program, Visual Studio Code(VS Code), we needed to include a number of required V-rep files into VS Code. We found that due to vague or unspecified instructions on outdated V-rep user manuals[52], when the required files were found, further files from further directories within V-rep needed to be included as well. To solve the problem of missing file inclusions, we ended up editing an example file, which utilizes a remote API, written directly in the V-rep source folder, which is why the file retains its former name, “bubbleRobClient”. Finally, by using Cmake to build a makefile, our program is compiled directly by calling the file and specifying the port number, to which V-rep has connected to, while it is running. Figure 8.7 and figure 8.8, shown below, demonstrate an example of a successful remote API communication thread between V-rep and VS Code:

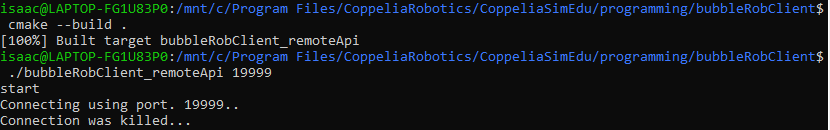


Fig. 8.7. Code Snippet Taken from an Ubuntu Terminal Showing Compilation of Code Written in VS Code and a Connection to the Port, 19999.

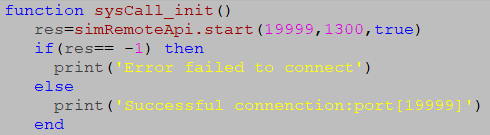




Fig. 8.8. Code Snippet and Terminal Output Shown in Lua, V-rep’s Embedded Scripting Language, Managing the Connection to Port 19999.

8.5.2 Direct Interface of the Closed Loop Remote Control Function

For the purposes of interfacing V-rep with a closed loop remote control system, a fundamental understanding of the closed loop RC is necessary to be established.

The closed loop function, described in further detail in Chapter 6, takes in the four inputs: forward; turn; maximum force of propellers; and an array of six variables listed as pitch, roll, height and their corresponding derivatives, alternatively known as “PRH states”. And the output of the closed loop RC function provides the necessary angles and throttles of each individual propeller for the drone to operate using.

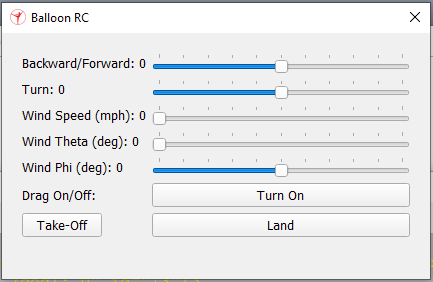


Fig. 8.9. Controller Input Layout of a Closed Loop Remote Control in GUI Format is Shown Above.

A completed closed loop remote control removes the need for a propeller speed variable, as seen in figure 8.9 which shows the current implementation of the GUI in V-rep. This system allows the drone to move with only two commands, forward throttle and turn factor, and corrects for external factors that cause the drone to tilt at unspecified angles.

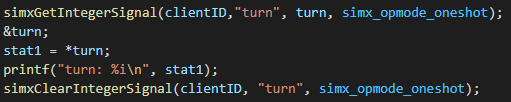


Fig. 8.10. Remote API Functions are Written to Pass Data Between Server and Client.

In figure 8.10, a single turn command is being retrieved from V-rep with a non-blocking function call, allowing data to be sent without waiting for a reply.[5[5](#wez5d9ftcpoh)] These remote API function calls are essentially the fundamental building blocks of transferring data between server and client programs.

Although the theory behind developing a closed loop remote control system was generally understood, performing these function calls required debugging the errors we found in much greater detail. So it is at this stage of our design where the practical simulation engineering section concludes and the beginning of our hypothetical tests as well as analysis of our overall simulation design begins.

8.5.3 Hypothetical Testing of Closed Loop Remote Control

If the implementation of closed loop RC were to be completed, STR 3.0.0, Remote Control, would be verified in this section.

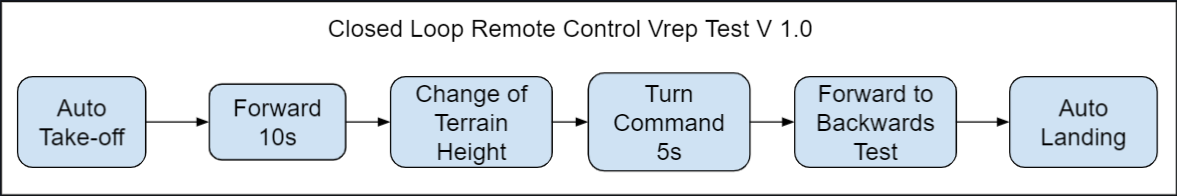


Fig. 8.11. The Block Diagram Above Shows How a Test Would be Conducted Using a Closed Loop Control System.

Firstly, to take the drone off the ground, an automatic take-off sequence is engaged with the press of the Take-off feature on the GUI. This feature causes the propellers to start up and increase thrust to maximum. Then, the forward throttle is increased to maximum; much like open loop remote control, the propellers face in the ‘forward’ direction and move so, adjusting in mid-air for any slight misdirections, caused by any uneven distributions of mass, thus torque. A change of terrain height is then presented to the drone and, with the barometer and ultrasonic sensors constantly feeding back data of height and altitude, an increase in throttle upward allows the drone to be able to avoid all obstacles that may potentially collide with the drone. After clearing a provided terrain height, the drone is issued a turn command for five seconds to test a range of narrow to wider turn radiuses. Next, we have the forward to backwards test. This test assesses a drone’s RC response to a significant amount of unforeseen pitch and pitch velocity, by switching the throttle back and forth in quick succession, to account for collisions, heavier than normal winds, or simple piloting errors. Finally at the end of an experiment, the autonomous landing feature is engaged with the press of the button on the GUI. The propellers, given the data of height and vertical velocity, correctly adjust propeller speeds to zero before the drone touches the ground, so as to prevent any high speed collisions with expensive hardware.

8.5.4 Further insights into the Closed Loop Remote Control Implementation

Due to our limited programming knowledge in C++, inexperience with communication networks, or in the absence of a fully developed or updated V-rep user manual for remote API[5[3](#60duwrvvqwtc)], closed loop remote control implementation was unable to be completed on time. Given more time to work with interfacing V-rep with VS Code, closed loop remote control may have been completely completed and tested. However, for reference, when seeking help from our TA, [Alexey Munishkin](mailto:amunishk@ucsc.edu), who has had previous experience with remote application programming in V-rep, interfacing with Matlab, he had told us that learning remote API in V-rep had taken him close to a year to do. In retrospect, Gazebo, an alternative robotic operating system we had in mind when deciding on a simulation environment, may have been a better option for operating our drone model due to its support of aerodynamic principles, specifically buoyancy and fluid density. The primary justifications we had in mind when choosing V-rep over Gazebo was modularity and the fact that Gazebo could not operate under a Windows operating system and was required to be installed through Ubuntu packages on Linux. The irony of this decision lies in the fact that in order for our client program(VSC) to compile C++, Ubuntu had to be installed anyway.

## Section 8.6: Verification of the Prototype’s Fabricated Dimensions

In this section the attempted verification of the following STR will be discussed:

**2.0.0** Drone shall be able to fly at least 5mph in winds up to 15mph.

Based on the newest fabricated model, discussed in further detail in Chapter 3, V-rep simulations were run to test potential flight speeds we would expect simply based on the model’s dimensions.

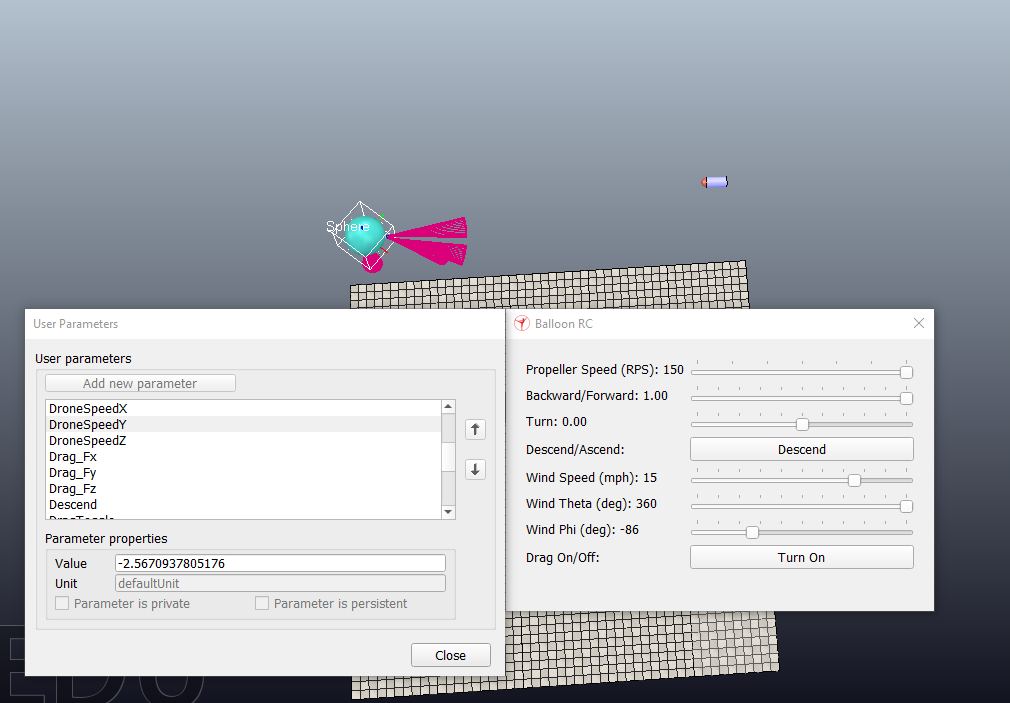


Fig. 8.12. Drone Speed is Shown to be -2.567 m/s on the Left Column Reading ‘User Parameters’

With the throttle and wind speeds set to maximum, the fabricated V7 envelope is calculated to fly with 40N drag in a 15 MPH head wind. Simply stated, the drone cannot move forward in the x-y plane against the maximum headwind at maximum throttle. As shown in figure 8.12, our drone is unable to fly within 15 MPH of headwind. Although horizontal drag had immensely overshot our expectations, vertical drag was significantly decreased, which would mean vertical flight testing was still within reasonable expectations to be achieved in physical testing.

## Chapter 8 Conclusion

Overall, the simulation discussed in this chapter provided results that we did not fully expect to get. At the same time, we were not able to see the conclusion of all the tests we expected to complete, specifically those of closed loop remote control and autonomous. However, we were able to verify STR 2.0.0, Drone Speed, in simulation tests that we were not able to retrieve in physical testing such as drone speed and controllability. Due to the failure to implement remote API, STR 3.0.0, Remote Control could not be verified.

# 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.

[51]

Boeing, Adrian & Braunl, Thomas. (2007). Evaluation of real-time physics simulation systems. 281-288. 10.1145/1321261.1321312.

[52]

“Enabling the Remote API - Client Side.” User Manual, CoppeliaSim, https://www.coppeliarobotics.com/helpFiles/en/remoteApiClientSide.htm.

[53]

“Remote API Functions (C/C++).” User Manual, CoppeliaSim, https://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctions.htm#simxGetIntegerSignal.

[54]

“Shape Dynamics Properties.” User Manual, CoppeliaSim, https://www.coppeliarobotics.com/helpFiles/en/shapeDynamicsProperties.htm.

[55]

“Remote API Modus Operandi.” User Manual, CoppeliaSim, https://www.coppeliarobotics.com/helpFiles/en/remoteApiModusOperandi.htm.