THREE-DIMENSIONAL QUADCOPTER MODELING AND SIMULATION USING REAL-TIME BRAIN MACHINE INTERFACE

A Thesis by

Jesus Enrique Gomez Llanos Tirado

Bachelor of Science in Aerospace Engineering, Wichita State University, 2017

Submitted to the Department of Aerospace Engineering and the faculty of the Graduate School of Wichita State University in partial fulfillment of the requirements for the degree of Master of Science

December 2019
THREE-DIMENSIONAL QUADCOPTER MODELING AND SIMULATION USING REAL-TIME BRAIN MACHINE INTERFACE

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science, with a major in Aerospace Engineering.

James Steck, Committee Chair

Atri Dutta, Committee Member

Jaydip Desai, Committee Member
DEDICATION

To my family, friends, mentors and everyone who has provided me guidance and support along the way
With ingenuity and humility, there are no boundaries for science and technology.
ACKNOWLEDGMENTS

I would like to thank my advisors, Dr. James Steck and Dr. Jaydip Desai, for their guidance, and feedback through the work of this thesis, as well for the opportunity and encouragement of research with this project. Both have demonstrated me that their impact and role model goes beyond the professionalism in class. Similarly, I would like show gratitude to Dr. Atri Dutta for his feedback on this thesis.

I would also like to thank Adam Reust and Yrithu Pillay for their friendship, guidance and support through different stages of this project. They dedicated hours to aid with the preparation, testing and simulation, which lead to the possibility to develop and finalize this project.

Finally, I would like to thank my family and friends who have provided with constant support and encouragement through the years. Especially to my parents and grandmother who have sacrificed everything to give me all the opportunities and challenges I set in front of me. This thesis would not have been possible without their help to overcome obstacles and inspiration of not giving up despite the difficulties.

This material is based upon work supported by the National Aeronautics and Space Administration (NASA) and the Kansas NASA EPSCoR Program (KNEP) under the Seed Research Initiation (SRI) grant. It is noted that any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Aeronautics and Space Administration or the Kansas NASA EPSCoR Program.
ABSTRACT

The recent progresses in the area of neuro-robotics science and technology has provided the possibility to explore its practicability in aviation, especially in the area of flight controls through brain machine interfaces. This research explores the development and reliability of an outer loop electroencephalography (EEG) controller implemented on a quadcopter using steady P300 Speller technology. An inner-loop quadcopter state controller communicates through user datagram protocol (UDP) connections with the on-board computer that provides stability with an embedded controller. The creation and modeling of the inner-loop, outer-loop controllers and interface with BCI2000 occurred with the use of MATLAB/Simulink. In order to test the controllers, a user intended to complete a flight mission path by simulating the selection of a command letter with EEG signals by clicking a letter on the BCI2000 interface. Similarly, the interface and controllers were tested using EEG signals of a test subject. The test in these two cases implemented and transmitted commands to a quadcopter to perform them in real time. Prior to the flight mission, a user was briefed with the tasks to be performed and how a flashing letter would be associated with an action of the quadcopter. Observation of the flight and analysis of data acquired during the flight demonstrated a 100% accuracy of the quadcopter performing the actions coming from the brain-machine interface (BMI) software. However, this accuracy depends on the accuracy of EEG data acquisition from the brain and the selection of the letter targeted by the user. Further challenges, such as drifting and range of motion limits of the quadcopter need to be addressed to consider this research as reliable technology. Nevertheless, the modeling and simulation of a quadcopter BMI provides a promising future in the area of aviation and neuro-robotics.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>1.</td>
</tr>
<tr>
<td>1.2</td>
<td>2.</td>
</tr>
<tr>
<td>1.3</td>
<td>3.</td>
</tr>
<tr>
<td>1.4</td>
<td>5.</td>
</tr>
<tr>
<td>1.5</td>
<td>8.</td>
</tr>
<tr>
<td>2.</td>
<td>9.</td>
</tr>
<tr>
<td>2.1</td>
<td>9.</td>
</tr>
<tr>
<td>2.1.1</td>
<td>9.</td>
</tr>
<tr>
<td>2.1.2</td>
<td>9.</td>
</tr>
<tr>
<td>2.2</td>
<td>11.</td>
</tr>
<tr>
<td>2.3</td>
<td>11.</td>
</tr>
<tr>
<td>2.3.1</td>
<td>14.</td>
</tr>
<tr>
<td>2.3.2</td>
<td>16.</td>
</tr>
<tr>
<td>2.3.3</td>
<td>18.</td>
</tr>
<tr>
<td>2.3.4</td>
<td>20.</td>
</tr>
<tr>
<td>2.3.5</td>
<td>24.</td>
</tr>
<tr>
<td>2.3.6</td>
<td>25.</td>
</tr>
<tr>
<td>3.1</td>
<td>26.</td>
</tr>
<tr>
<td>3.2</td>
<td>27.</td>
</tr>
<tr>
<td>3.2.1</td>
<td>28.</td>
</tr>
<tr>
<td>3.2.2</td>
<td>29.</td>
</tr>
<tr>
<td>3.2.3</td>
<td>30.</td>
</tr>
<tr>
<td>3.3</td>
<td>30.</td>
</tr>
<tr>
<td>3.4</td>
<td>31.</td>
</tr>
<tr>
<td>4.</td>
<td>34.</td>
</tr>
<tr>
<td>4.1</td>
<td>34.</td>
</tr>
<tr>
<td>4.2</td>
<td>36.</td>
</tr>
<tr>
<td>4.3</td>
<td>37.</td>
</tr>
<tr>
<td>4.4</td>
<td>38.</td>
</tr>
<tr>
<td>5.</td>
<td>41.</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (continued)

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Feedback Controller Response ........................................... 41</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Yaw Controller Response .................................................. 42</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Pitch Controller Response ................................................. 44</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Roll Controller Response .................................................. 46</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Height Controller Response ................................................ 46</td>
</tr>
<tr>
<td>5.2</td>
<td>Flight Mission .............................................................. 48</td>
</tr>
<tr>
<td>5.3</td>
<td>EEK Flight Control .......................................................... 52</td>
</tr>
<tr>
<td>6.</td>
<td>CONCLUSION ................................................................. 57</td>
</tr>
<tr>
<td>6.1</td>
<td>Technology Reliability ..................................................... 58</td>
</tr>
<tr>
<td>6.2</td>
<td>Future Work and Recommendations ......................................... 58</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>................................................................. 61</td>
</tr>
<tr>
<td>APPENDIXES</td>
<td>................................................................. 64</td>
</tr>
<tr>
<td>A.</td>
<td>Drone State Request Function ............................................ 65</td>
</tr>
<tr>
<td>B.</td>
<td>Data Synchronization Function ............................................ 69</td>
</tr>
<tr>
<td>C.</td>
<td>AR.Drone Data Decoding Function .......................................... 71</td>
</tr>
<tr>
<td>D.</td>
<td>Letter Selected into MATLAB Script ..................................... 74</td>
</tr>
<tr>
<td>E.</td>
<td>Individual P300 Speller Stimulus Data Sets .......................... 79</td>
</tr>
<tr>
<td>F.</td>
<td>Flight Mission Graphs ...................................................... 83</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Sensors of the AR.Drone 2.0</td>
<td>9</td>
</tr>
<tr>
<td>2. Characters Equivalent to Quadcopter Maneuvers</td>
<td>18</td>
</tr>
<tr>
<td>3. Dimensions of Obstacles along Trajectory</td>
<td>38</td>
</tr>
<tr>
<td>4. Rates of Change for States</td>
<td>41</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Axes and rotations of the Parrot AR.Drone 2.0</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Angular speed variation to achieve (a) vertical motion, (b) θ, (c) φ and (d) ψ</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Flowchart of simulation</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>Script to initialize Parrot AR.Drone 2.0 MATLAB/Simulink model</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>Parrot AR.Drone 2.0 system</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>Time Filter subsystem</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>Function to limit the input command</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>EEG Command Controller subsystem</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>Classification of input received function</td>
<td>17</td>
</tr>
<tr>
<td>10</td>
<td>Feedback control loop</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>Feedback Controller subsystem</td>
<td>19</td>
</tr>
<tr>
<td>12</td>
<td>AR.Drone Plant subsystem</td>
<td>20</td>
</tr>
<tr>
<td>13</td>
<td>Send Commands subsystem</td>
<td>21</td>
</tr>
<tr>
<td>14</td>
<td>AR.Drone subsystem</td>
<td>21</td>
</tr>
<tr>
<td>15</td>
<td>Decode States subsystem</td>
<td>22</td>
</tr>
<tr>
<td>16</td>
<td>Reset Yaw function</td>
<td>23</td>
</tr>
<tr>
<td>17</td>
<td>Position Estimation subsystem</td>
<td>24</td>
</tr>
<tr>
<td>18</td>
<td>Visualization of States subsystem</td>
<td>25</td>
</tr>
<tr>
<td>19</td>
<td>Electrode placement</td>
<td>26</td>
</tr>
<tr>
<td>20</td>
<td>BCI2000 configuration window</td>
<td>27</td>
</tr>
<tr>
<td>21</td>
<td>Copy mode window</td>
<td>28</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>Average P300 Speller stimulus of six runs of data acquisition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Linear classification for real-time control mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>Signal Processing system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>Command Classification function</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>Equipment to obtain EEG signals composed by (a) g.GAMMAcap,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b) g.USBamp and (c) g.GAMMAbox</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>Arduino Mega 2560 setup</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>Quadcopter planned trajectory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>Vicon Tracker camera setup</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Flight mission obstacles on Vicon Tracker</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>Quadcopter (a) with markers and (b) representation on Vicon Tracker</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>Positive yaw command and response on the quadcopter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>Negative yaw command and response on the quadcopter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>Positive pitch command and response on the quadcopter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>Negative pitch command and response on the quadcopter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>Roll command and response on the quadcopter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>Increase height command and response on the quadcopter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>Decrease height command and response on the quadcopter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>Isometric view of the flight mission path</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>Position comparison between Vicon Tracker and Simulink</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>Commands sent through the quadcopter along the EEG flight control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>42</td>
<td>Yaw response along the EEG flight control</td>
<td>53</td>
</tr>
<tr>
<td>43</td>
<td>Pitch response along the EEG flight control</td>
<td>54</td>
</tr>
<tr>
<td>44</td>
<td>Roll response along the EEG flight control</td>
<td>55</td>
</tr>
<tr>
<td>45</td>
<td>Height response along the EEG flight control</td>
<td>55</td>
</tr>
<tr>
<td>46</td>
<td>P300 Speller data set #1</td>
<td>79</td>
</tr>
<tr>
<td>47</td>
<td>P300 Speller data set #2</td>
<td>80</td>
</tr>
<tr>
<td>48</td>
<td>P300 Speller data set #3</td>
<td>80</td>
</tr>
<tr>
<td>49</td>
<td>P300 Speller data set #4</td>
<td>81</td>
</tr>
<tr>
<td>50</td>
<td>P300 Speller data set #5</td>
<td>81</td>
</tr>
<tr>
<td>51</td>
<td>P300 Speller data set #6</td>
<td>82</td>
</tr>
<tr>
<td>52</td>
<td>X-Z plane view of the flight mission path</td>
<td>83</td>
</tr>
<tr>
<td>53</td>
<td>Y-Z plane view of the flight mission path</td>
<td>84</td>
</tr>
<tr>
<td>54</td>
<td>X-Y plane view of the flight mission path</td>
<td>84</td>
</tr>
<tr>
<td>55</td>
<td>X-Z plane view of position comparison between Vicon Tracker and Simulink</td>
<td>85</td>
</tr>
<tr>
<td>56</td>
<td>Y-Z plane view of position comparison between Vicon Tracker and Simulink</td>
<td>85</td>
</tr>
<tr>
<td>57</td>
<td>X-Y plane view of position comparison between Vicon Tracker and Simulink</td>
<td>86</td>
</tr>
<tr>
<td>58</td>
<td>Commands sent through the quadcopter along the flight mission</td>
<td>86</td>
</tr>
<tr>
<td>59</td>
<td>Yaw response along the flight mission</td>
<td>87</td>
</tr>
<tr>
<td>60</td>
<td>Pitch response along the flight mission</td>
<td>87</td>
</tr>
<tr>
<td>61</td>
<td>Roll response along the flight mission</td>
<td>88</td>
</tr>
<tr>
<td>62</td>
<td>Height response along the flight mission</td>
<td>88</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>3D</td>
<td>Three Dimensional</td>
<td></td>
</tr>
<tr>
<td>BMI</td>
<td>Brain-Machine Interface</td>
<td></td>
</tr>
<tr>
<td>CG</td>
<td>Center of Gravity</td>
<td></td>
</tr>
<tr>
<td>COM</td>
<td>Communication</td>
<td></td>
</tr>
<tr>
<td>DOF</td>
<td>Degrees of Freedom</td>
<td></td>
</tr>
<tr>
<td>EEG</td>
<td>Electroencephalography</td>
<td></td>
</tr>
<tr>
<td>IMU</td>
<td>Inertial Measurement Unit</td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
<td></td>
</tr>
<tr>
<td>PID</td>
<td>Proportional-Integral-Derivate</td>
<td></td>
</tr>
<tr>
<td>PVC</td>
<td>Polyvinyl Chloride</td>
<td></td>
</tr>
<tr>
<td>RPM</td>
<td>Revolution per Minute</td>
<td></td>
</tr>
<tr>
<td>SSVEP</td>
<td>Steady State Visually Evoked Potential</td>
<td></td>
</tr>
<tr>
<td>UAV</td>
<td>Unmanned Aerial Vehicle</td>
<td></td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
<td></td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
<td></td>
</tr>
</tbody>
</table>
LIST OF SYMBOLS

Hz  Hertz
i   Input or Command
j   Output or Actual State
k   Error
ms  Milliseconds
r^2 Coefficient of determination
rad Radian
u   Forward velocity
v   Lateral velocity
w   Vertical velocity
x   Longitudinal axis
X   Position along x on the inertial reference frame
y   Lateral axis
Y   Position along y on the inertial reference frame
z   Vertical axis
Z   Position along z on the inertial reference frame or height
\omega_1 Angular speed of rotor 1
\omega_2 Angular speed of rotor 2
\omega_3 Angular speed of rotor 3
\omega_4 Angular speed of rotor 4
°   Degrees
"   Inches
\[ \pm \quad \text{Plus-minus} \]
\[ \frac{deg}{s} \quad \text{Degrees per second} \]
\[ \frac{m}{s} \quad \text{Meters per second} \]
\[ \phi \quad \text{Pitch} \]
\[ \theta \quad \text{Roll} \]
\[ \psi \quad \text{Yaw} \]
CHAPTER 1

INTRODUCTION

Flight control has been present since the beginning of aviation with the notion that a manual input needs to be implemented in order to reach the desired output as commanded by a pilot. Those manual inputs take place with the pilot being behind the control wheel of the aircraft during motion or at a distant remote control device. About six decades after the first flight of an aircraft, one of the most revolutionary periods in aviation came with the incorporation of fly-by-wire systems, which greatly reduced the physical demands of operating an aircraft. However, it is only until recent times that technology has allowed the exploration of new methods of flight control besides the traditional cockpit’s control wheel or remote control. Now, EEG provides a wide range of applications and research with the use of the P300 Speller control as a possible technology to operate complex systems, including aircraft.

1.1 Motivation

The current state of neuro-robotics science and technology has opened countless possibilities of applications on multiple fields of study. Nevertheless, aviation has not seen major advancements related to neuro-robotics in comparison with medicine, biology and manufacturing. Besides the main idea to provide novel technologies for the traditional flight control systems, another consideration of this research is aimed to narrow the gap between certified pilots and those limited to operate them due to the lack of an extremity, motor disability or experience with the system. Leeb et al. [1] concluded that even in cases of severe motor disabilities, the users controlling a telepresence robot for displacement around rooms of a building through BMI from a distance of 100 kilometers had almost the same performance as users without any disabilities. As a result of this, we can say that there are many opportunities to explore systems that could be
controlled by brain signals in order to avoid human exposure to hostile or unreachable environments like space.

1.2 Existing challenges

Studies, like those previously mentioned, demonstrate some potential in other applications and areas, but we have to consider that tasks can be more demanding as systems tend to be more complex, especially on spacecraft and aircraft. For this reason, this study is initially limited to the research of a quadcopter’s ability to perform commands through a BMI. Another consideration in the complexity of these systems comes from the results obtained by Vecchiato et al. [2], in which the current highest performance of BMI control occurs at 83% of accuracy while piloting an aircraft. However, pilots do not operate an aircraft by only adjusting the control wheel. They have to perform simultaneous tasks during flight, which according to this study, could reduce the accuracy of BMI’s by 20%. These outcomes suggest that rather than controlling the aircraft with BMI, they would recommend the pilot to manipulate other less critical tasks, such as communications with crewmembers or filling logs for the aircraft.

Challenges for EEG technology are not only limited to issues of accuracy and performance due to complexity. Saha et al. [3], studied different cases in which EEG signals may suffer variability during motor imagery data collection. They obtained results where signals of a subject can deviate from 7.52% to a maximum of 31.0% during data collection on sessions not executed in the same day. Similar to other studies, they contribute those deviations from secondary parameters that are not included in their initial considerations to investigate. Those parameters could be psychological, such as current mood and depression, or mental state, such as fatigue and frustration. Other researchers, such as Yuan-Pin and Tzyy-Ping [4], have tried to understand and reduce those day-to-day deviations by considering emotions as an important parameter.
Nevertheless, they recognize the complexity of the topic and that further studies need to address these deviations due to the fact that there can exist overlapping patterns or variables not included in their results.

It is important to note that for this research, as well for most of the research previously done; the researchers create an environment where distractions and the removal of external elements is essential to avoid uncontrolled parameters or unexpected results. Menon et al. [5] mentioned that different environments might produce a variety of results if the user diverges from a controlled environment. Another consideration from these authors was the possibility to implement BMI for space control systems, in which they mention that gravity, radiation and other unknown elements could produce different results with brain signals than a system developed for Earth purposes. This is an important argument, as future work will expand on the parameter of gravity affecting this research.

1.3 Research Background

For this research, two main studies provide a clear outline on how to proceed. The first was committed to the development of BMI within a flight simulator, while the second one emphasizes some of the initial efforts to apply EEG technology into a quadcopter. It is important to note that the researchers from the first study had three collaborations with different authors across multiple universities due to the difficulty of the project. As for the second one, multiple researchers have attempted similar projects, but they do not represent a considerable research extension from its initial development and application.

To begin with, Fricke et al. [6], as a cooperation between the Technical University of Berlin and Technical University of Munich, replaced some manual inputs on a Diamond DA42 aircraft flight simulator with a BMI to control one degree of freedom (DOF). In this case, the yaw angle
was the only BMI input from the user, while the control of other movements occurs with an algorithm created by the authors in order to achieve the landing of the airplane. Despite only one DOF being controlled, only two out of seven pilots with different backgrounds and flying experience were able to succeed on the mission described by the researchers. Based on machine learning, that study trained a computer to distinguish between different thoughts of performing an action. In order to be able to understand the action imagined by the user during the flight session, each user needed to attend preliminary sessions of data acquisition to train the system.

Secondly, Fricke et al. [7] proceeded to further explain the potentials and challenges for the type of technology that they were exploring. In this scenario, the authors did not perform any experiments or simulations to demonstrate their arguments. The main challenge considered was the acceptance by the community who currently accepts a traditional flight control systems. However, the main proposal was to implement this new technology as a way to bring more reliability to current practices due to the claim that it can bring more benefits than challenges, such as a valuable implementation for the physically disabled and the elimination of physical exhaustion from the pilot by serving as an imaginary mechanical link with the aircraft.

Lastly, Fricke et al. [8] advanced their research by proposing a new method called “operant” in which the users will learn beforehand how to control the interface. This provided better results than their first research based on machine learning algorithms. The advantages of this method compare to their first research is that instead of adapting the interface for each user and session, the user will know how to better control the interface after a few days of training. Nevertheless, the authors mentioned that this did not prove more successful than their first method due to the lack of proper training and limitations of variables while comparing their testing, specially related to having the same subjects for both studies.
As for the quadcopter research, LaFleur et al. [9] were able to control three DOF, but the pilot only controlled two of them with EEG signals. The imagination of movements from the body would control the yaw angle and altitude, while a constant forward velocity would be maintained when the user would not control any of the other two states. The authors claim a 90.5% success rate of completing all their targets along an obstacle course. They implement similar machine learning algorithms of the previously mentioned research, but the idea is to introduce the user to BMI’s from basic to more complex scenarios provides a better background to operate the quadcopter. The subjects gain familiarity with BMI interfaces by being first introduced to two one DOF scenarios, then to a two DOF scenario, later to a virtual scenario of the quadcopter and finally an application with the real world quadcopter. Most of the previous studies that use EEG signals as the source of their commands, state that the success on their results come from the implementation of training sessions in order to allow the users to get familiar with the interface.

In terms of previous studies for other aspects of this research, Ribeiro and Oliveira [10] developed controllers on MATLAB/Simulink for external systems. Their implementation was directed to unmanned aerial vehicles (UAV) on real world environments and flight simulations on X-Plane. They explain the knowledge required for this research in terms of creating interfaces to communicate between software using UDP connections. Other authors, such as Paiva et al. [11], developed equations of motions for quadcopters with the purpose to create proportional-integral-derivate (PID) controllers after the modeling of the system.

1.4 Fundamentals of SSVEP, P300 Speller and BMI

The technology to extract the EEG signals can be invasive or non-invasive. All of the BMI technology discussed in this research utilizes a non-invasive. In order to achieve this, experiments
utilize an EEG cap with electrodes and electrode gel to improve the readings of electrical activity between the skin and the electrode. The interface developed in this research is aimed to be utilized with P300 Speller technology. Steady state visually evoked potential (SSVEP) signals tend to be a response of humans to visual stimulation at certain frequencies. In this research, the P300 Speller signals acquired for the control of the quadcopter occur 300 ms after the visual stimulation. Typically, these frequencies fluctuate between 8 and 28 Hz.

İşcan and Nikulin [12] explained how SSVEP focuses on the occipital and parietal lobes of the brain. They added how the flickering of a light source at a constant frequency stimulates these regions. One of the main benefits of this type of BMI is the high signal to noise ratio, making it easier to obtain the corresponding signals to the experiment. Because of this, multiple developers have agreed that SSVEP tends to have a better processing time than other BMI’s, such as the imagination of body movements without performing them.

Kuş et al. [13] described how SSVEP and flickering signal classification mostly take place in three main categories. Low frequency response is located around 10 Hz, medium frequency from 13 to 25 Hz and high frequency from 40 to 60 Hz. The range of frequency values will ultimately depend on the activity performed in front of the user. This activity could involve a certain type or a variety of characters, such as letters, numbers, or symbols at different rates of flickering. A specific character is associated with a specific amplitude and frequency signal. However, these last two characteristics of a wavelength may vary from person to person.

Most of the BMI development has occurred in areas of prosthetics and robotics with the purpose to assist people missing a limb or with motor disabilities. For assisting purposes of motor disabilities, Kim et al. [14] designed a BMI applied to a wheelchair using motor imagery based control. They implemented their system with aid of technology from the company g.tec, which is
capable of acquiring EEG signals and MATLAB/Simulink to analyze and process the data in real time. The initial data for training was acquired using an arrow stimuli environment, in which a direction was related to a specific motor imagery action. The researchers were able to accommodate five different commands, such as left, right, left-diagonal, right-diagonal and forward. A rest command would take place if the BMI module received none of the previous commands. Their commands were sent to the wheelchair based on a workflow classification and transmitted to the wheelchair using UDP connections.

There are other cases in which researchers focus on the replacement of a limb through prosthetics. Bright et al. [15] focused their prosthetic as an arm replacement, but only considered the movement of the hand through BMI. In this case, the researchers used a Neurosky Mindwave Mobile headset to acquire motor imagery EEG signals and MATLAB scripts to classify them. The movement of their hand only responded to three commands; 1) flexion, 2) extension of all fingers and 3) pinching. The raw signals classification had three categories coming from the headset. Flexion corresponded from the value 20 to 49, extension from 50 to 69 and pinching from 70 to 100. Besides accomplishing movements on the hand, they compared the accuracy rate based on training sessions. The user with a month of training achieved up to an 80% of accuracy, while a user with a week of training reached 45% and a user with no training only 20% of accuracy.

However, there are also BMI’s with applications to robotics that cannot be categorized in the previous fields of study. Bhattacharyya et al. [16] developed a BMI that controls the motion of a robot on a surface with the purpose of incorporating this technology into current military robots. The researchers used NeuroWin as the system with electrodes that acquire the motor imagery EEG signals from the scalp. Shoulder, finger and elbow movement of each arm gave six commands to the BMI. However, the commands to the robot were sent after a sequence of four commands. This
gave 16 possible actions by the robot, but brought complexity for the user when recognizing the sequences corresponded to the robot’s actions. Similar to wheelchair research, these authors used an arrow stimuli scenario to obtain and train the interface with the use of MATLAB to perform the processing of data.

1.5 Research Objectives

As discussed in previous sections, earlier studies set the foundations of this research. In order to create the ability to manipulate three DOF of a quadcopter through EEG signals on a given environment, and due to the complexity of the system, the objectives for the current investigation will limited to the following:

1. Understand, develop and simulate a reliable model of a quadcopter.

2. Demonstrate the BMI control of three DOF of the quadcopter in order to provide the ability to move competently in a three dimensional (3D) environment.

3. Determine the inner-loop quadcopter state controller or feedback loop that offers the best response to the system for the different types of motion.

4. Test and understand the reliability and potential of outer-loop EEG control into flight control systems.
CHAPTER 2
MODELING AND SIMULATION

2.1 Overview of the Parrot AR.Drone 2.0

The AR.Drone 2.0 is a quadcopter manufactured by Parrot. It has Wi-Fi connection, which allows communication with any device with the same property, such as cellphone, tablet or computers. Some other specifications of the AR.Drone 2.0 [17] include the presence of two cameras, one facing to the front and one facing downwards, a weight of 420 grams and four motors of 14.5 Watts and 28,500 revolutions per minute (RPM).

2.1.1 Sensors

The inertial measurement unit (IMU) onboard of the quadcopter contains multiple sensors that provide its current Euler angles and position. Table 1 summarizes some of these sensors utilized for this study.

TABLE 1
SENSORS OF THE AR.DRONE 2.0

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ultrasound sensor</td>
<td>Measures altitude from surface</td>
</tr>
<tr>
<td>Barometer</td>
<td>Measures global altitude</td>
</tr>
<tr>
<td>3 axes gyroscope</td>
<td>Measures angular velocity of the drone</td>
</tr>
<tr>
<td>3 axes magnetometer</td>
<td>Measures global orientation</td>
</tr>
<tr>
<td>3 axes accelerometer</td>
<td>Measures acceleration acting on the drone</td>
</tr>
</tbody>
</table>

2.1.2 Flight Dynamics

The quadcopter follows the conventional axes and rotations of an aircraft. In this case, the x-axis points in a positive direction towards the front of the quadcopter, the y-axis is positive towards the right of the quadcopter as it is orthogonal with the x-axis and the z-axis points in a
positive direction downwards orthogonal to the previous two axes. This creates the positive rotations as follow: (1) $\theta$ when $x$ rotates on itself and $y$ initially heads downwards; (2) $\phi$ when $y$ rotates on itself and the $x$ initially heads upwards; and (3) $\psi$ by rotating $z$ on itself and the positive $x$ heading towards the position of positive $y$. Figure 1. Shows these positive axes and rotations of the quadcopter.

Figure 1. Axes and rotations of the Parrot AR.Drone 2.0

Figure 2 shows how the drone achieves $\theta$, $\phi$ and $\psi$ while $\omega_1$, $\omega_2$, $\omega_3$ and $\omega_4$.

Figure 2. Angular speed variation to achieve (a) vertical motion, (b) $\theta$, (c) $\phi$ and (d) $\psi$
Quadcopters achieve lift when opposite rotors rotate the same direction. In this case, rotors one and three rotate counterclockwise, while rotors two and four rotate clockwise. It is important to note that case (a), produces a hovering position due to all rotor’s angular speeds being the same. Vertical motion in either direction occurs when all the speeds are increased or reduced at the same time. Positive $\theta$ (b), which results in a displacement in the positive y, occurs when $\omega_1$ and $\omega_4$ are higher than $\omega_2$ and $\omega_3$. The positive $\phi$ (c), which results in a displacement in the negative x, occurs when $\omega_1$ and $\omega_2$ are higher than $\omega_3$ and $\omega_4$. The positive $\psi$ (d), which produces a pure rotation on z, occurs by a higher $\omega_1$ and $\omega_3$ in comparison with $\omega_2$ and $\omega_4$.

2.2 Simulation System

The simulation of this research starts from recreating the selection of a letter by brain signals until it achieves the control of drone’s motion in real-time. Figure 3 shows the flowchart of the system’s simulation. The “BCI2000”, “MATLAB Interface” and “Signal Processing MATLAB/Simulink Model” are discussed in the following chapter as pertains to the EEG interface, while the “Parrot AR.Drone 2.0 MATLAB/Simulink Model” is the focus of this chapter.

2.3 Parrot AR.Drone 2.0 MATLAB/Simulink Model

The initial model used in this research comes from the work done by Sanabria and Mosterman [18] in the “ARDrone Simulink Development Kit”, which is available through “File Exchange” in MathWorks. All the files within this kit only apply to the Parrot AR.Drone 2.0. The
kit contains two controllers focused on the hovering and tracking predetermined points in a 3D environment. Therefore, modifications were essential in order to accomplish the tasks for this research. Figure 4 shows the script to initialize and set up the Simulink model. After cleaning the workspace, the script opens the folder in which the model is located, opens the Simulink model and compiles it. A sample time of .065 seconds denotes how often data from the quadcopter records on the computer for further analysis. It is important to note that the simulation does not automatically starts after the compilation ends. An operator will start the simulation once the user indicates the start.

Cleaning workspace and open directory

```matlab
bdclose all;
clear all;
clc
cd 'C:\Users\BioMed\Documents\Drone Project Fall 2019';
```

Sample time of Simulink model

```matlab
sampleTime = 0.065;
```

Initialize and compile model

```matlab
ARDrone_EEG; % Open model
rtwbuild('Parrot_AR_Drone_2_0'); % Building model using RTWT.
```

Figure 4. Script to initialize Parrot AR.Drone 2.0 MATLAB/Simulink model

Once the script opens the Simulink system “Parrot_AR_Drone_2_0” shown in Figure 5, MATLAB will compile the model. The subsystem contains other subsystems, such as “Time Filter”, “EEG Command Controller”, “Feedback Controller”, “AR.Drone Plant”, “Position Estimation” “and “Visualization of States”. Outside of these subsystems, there are other commands essential to operation of the simulation. The input to this model occurs through a UDP connection with the “Signal Processing MATLAB/Simulink” model.
Figure 5. Parrot AR.Drone 2.0 system
The switch “Fly” sends a command to the quadcopter to start the rotation of the propellers to take-off or reduces the rotation until the quadcopter lands. The “Take Off” option will make the quadcopter go from rest on the ground to an altitude approximately of .8 meters. The switch “Enable References” allows to send commands to the quadcopter if enabled. This allows the EEG commands to control the quadcopter translation and rotation as chosen by the user. Otherwise, the quadcopter will maintain a hovering position. The switch “Emergency Stop” sends a command to halt any action of the quadcopter or rotation of the propellers. In addition, at this level of the model we can observe the “Battery Display”, which shows the percentage of battery remaining from zero to 100. The “AR.Drone Plant” and “Position Estimation” subsystems determine the inner-loop quadcopter state controller, and the “Time Filter”, “Feedback Controller” and “EEG Command Controller” subsystems constitute the outer-loop EEG controller.

2.3.1 Time Filter

The signals coming out of the EEG interface last about 14 s until the next output transmits to the quadcopter. The 14 s are implemented because the collection of more EEG data along a range of time improves the accuracy of the letter selection. The delay of command could produce unexpected results during flight, such as crashing or missing the target destination, condition or rotation. As a result, this subsystem limits the output of the EEG interface to three and a half seconds and then return the value to a hovering position. This provides more time to users to make a decision about their path. Figure 6 shows the structure of the “Time Filter” subsystem.

![Figure 6. Time Filter subsystem]
The “Clock” keeps the time of the simulation, the “Input” comes directly from the BCI2000 interface and the “Command” is the value limited on time.

Figure 7 shows the function of the subsystem that limits the command sent to the quadcopter. The variables related to time in this function will restart every time the simulation receives a new command.

```
% Function to limit the time of the input into the quadcopter.
function y = fcn(time,u)
    persistent catch_time;
    persistent last_in;
    if isempty (catch_time)
        catch_time = time;
    end
    if isempty (last_in)
        last_in = 9999;
    end
    if last_in ~= u
        y = u;
        catch_time = time;
        last_in = u;
    else
        if (time - catch_time) <= 3.5 % Seconds to activate input
            y = u;
        else
            y = 0;
        end
    end
end
```

Figure 7. Function to limit the input command

After three and a half seconds have passed, the command or output of this function will switch from the value of the EEG interface to zero, which is described as hovering position on the next subsystem.
2.3.2 EEG Command Controller

Figure 8 shows the “EEG Command Controller” subsystem structure that processes the command by the user into a specific quadcopter’s action. This subsystem contains the current states coming from the “AR.Drone Plant”, “Position Estimation” and the value from one to seven coming from the “Signal Processing” subsystem. After the command is processed, one of the three states adjusts accordingly based on the function “Classification of Input Received”. The pitch angle command keeps a constant gain, either positive or negative, while chosen and restarts to zero when other commands execute. As contrast with the yaw and height, in which is important to track and maintain the current yaw angle and altitude value. A unit delay helps to avoid an algebraic loop into the yaw angle calculations.

![Diagram of EEG Command Controller subsystem]

The function inside of the “EEG Command Controller”, which is shown on Figure 9, processes and sends the commands to the quadcopter. Small gains were decided to give a better reaction time for the users. The value of these gains were chosen from testing and observation of flight performance. The yaw and pitch command values change into radians with the purpose of
matching the data processed on the drone’s IMU. The testing of the model revealed that there were some problems translating the commands of yaw angle equal or greater than ±180º.

% Function to change states based on EEG command
function [Yaw1,Pitch1,Height1] = fcn(Yaw0, YawA,Height0, u)

    % Initialize variables and maintain current states
    Height1=Height0;
    Pitch1=0;
    Yaw1=Yaw0;

    if u == 0 || u == 1 || u == 5 || u == 9 % Maintain current states
        Height1=Height0;
        Pitch1=0;
        Yaw1=Yaw0;
    elseif u == 2 % Increase height .1 m every second
        Height1 = Height0+.075;
        if Height0 > 3.5 || Height1 > 3.5
            Height1 = 3.5; % Constraint of altitude
        end
    elseif u == 3 % Negative pitch of 1.25 degrees every second
        Pitch1 = -1.25*(pi/180);
    elseif u == 4 % Negative yaw .5 degrees every second
        Yaw1 = Yaw0-(.35*(pi/180));
        if YawA < -3.125 || Yaw1 < -3.125 % Constraint of rotation
            Yaw1 = -3.125;
        end
    elseif u == 6 % Positive yaw .5 degrees every second
        Yaw1 = Yaw0+(.35*(pi/180));
        if YawA > 3.125 || Yaw1 > 3.125 % Constraint of rotation
            Yaw1 = 3.125;
        end
    elseif u == 7 % Positive pitch of 1.25 degrees every second
        Pitch1 = 1.25*(pi/180);
    elseif u == 8 % Decrease height .1 m every second
        Height1 = Height0-.075;
        if Height0 > 3.5 || Height1 > 3.5 % Constraint of altitude
            Height1 = 3.5;
        end
    end
end
end

Figure 9. Classification of input received function
In this number, there is a consideration based on the overshoot done by the system. In addition, the condition takes into account the yaw angle commanded by the user and the yaw angle performed by the quadcopter, as there may be a delay of the response from the controller to the system. A similar constraint was implemented for the height. A lower limit of 1 m and upper limit of 3.5 m were chosen to help the user to stay in a region that would avoid unfavorable conditions to continue the mission. These conditions could add stress to the user and reduce their performance.

The command values assignment corresponds to the location of the characters on the BCI2000 interface. Rest contains multiple options because the interface contains two “S” characters, “X” is a hidden character on the “BCI2000 Interface” and because the “Time Filter” subsystem restarts the command to zero after three and a half seconds. Table 2 summarizes the characters and values corresponding to the quadcopter’s action.

**TABLE 2**

<table>
<thead>
<tr>
<th>Character</th>
<th>Value</th>
<th>Quadcopter Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>S and X</td>
<td>1, 9, 5 and 0</td>
<td>Rest</td>
</tr>
<tr>
<td>U</td>
<td>2</td>
<td>Increase height</td>
</tr>
<tr>
<td>I</td>
<td>3</td>
<td>Negative pitch</td>
</tr>
<tr>
<td>L</td>
<td>4</td>
<td>Negative yaw</td>
</tr>
<tr>
<td>R</td>
<td>6</td>
<td>Positive yaw</td>
</tr>
<tr>
<td>O</td>
<td>7</td>
<td>Positive pitch</td>
</tr>
<tr>
<td>D</td>
<td>8</td>
<td>Decrease height</td>
</tr>
</tbody>
</table>

**2.3.3 Feedback Controller**

The design of the “Feedback Controller” subsystem follows the rationale of the diagram shown in Figure 10. In this case, i corresponds to the input value of the system, j is the output
obtained from the sensors of the quadcopter and k marks the difference or error between those values. After obtaining k, the value goes to the “AR.Drone Plant” in order to obtain the desired response. The roll has an i value of zero as it is a state that will not be controlled by the user. While i originates from the “EEG Command Controller” and p comes from the “AR.Drone Plant” and “Position Estimation” subsystems, the value of k initiates in this subsystem.

![Figure 10. Feedback control loop](image)

Figure 10. Feedback control loop

Figure 11 shows the structure of the “Feedback Controller” subsystem as based on the previous feedback control loop with unity gain. A feedback loop was created for the stability of $\phi$, $\theta$, $\psi$ and $Z$.

![Figure 11. Feedback Controller subsystem](image)
2.3.4 AR.Drone Plant

Piskorski et al [19] indicates how the quadcopter processes communication within the IMU, how data transmits through Wi-Fi connection, commands and the language needed for the necessary modifications. The system consists of three other subsystems called “Send Commands”, “AR.Drone” and “Decode States”. The first subsystem processes the commands by the user and sends them into the quadcopter to perform an action, the second sends and receives data from the drone, and the third obtains the current states of the quadcopter processed by the IMU. Figure 12 shows the structure of the quadcopter’s plant subsystem.

![Diagram of AR.Drone Plant subsystem]

Figure 12. AR.Drone Plant subsystem

The “Send Commands” subsystem as shown in Figure 13 is where the interface prepares to send the commands to the quadcopter as directed by the user. The inputs originated from the “EEG Command Controller” and processed through the “Feedback Controller” are received as height, roll angle, pitch angle and yaw reference. This system also manages the inputs for the emergency stop and the fly command. The “Drone State Request” function, found in appendix A, prepares all the data sent to the quadcopter.
Figure 13. Send Commands subsystem

Figure 14 shows the “AR.Drone” subsystem in which the communication to and from the quadcopter occurs. The “Data to Drone” comes from the information computer on the “Drone State Request” subsystem, while the “Data from Drone” is processed in the “Decode States” subsystem. The “Commands Enable” and “Command Active” information only passes through this block because the information is needed in the two adjacent subsystems for data processing.

Figure 14. AR.Drone subsystem
Three UDP blocks connected through the 192.168.1.1 IP address are located in the system. The “Packet Output” blocks located in the “Input to Drone” are in charge to send the commands to the drone. The UDP labeled as “1B2h” connects through the UDP port 5554, while the “1B4h” connects through the UDP port 5556. The “1B2h” block only sends two elements of data type uint8 to initialize an output array to the drone about the information of the drone. However, the “1B4h” block has a length of 150 elements and it is in charge to modify and control the commands on the drone. The “Packet Input” in the “Output from Drone” transmits information about the drone’s states from the quadcopter’s computer. This block also uses the UDP port 5554.

Figure 15 shows the “Decode States” subsystem where the information received from the drone is processed. The data coming from the drone is in the data type of uint8 as an array of 500 elements. The array received every iteration accumulates coming out of the UDP block. Therefore, the function “Data Synchronization” implements a limit of 500 elements per array and transmits it as a frame to the following step. Appendix B shows the script developed in this function.

![Diagram of Decode States subsystem](image)

Figure 15. Decode States subsystem
The “AR.Drone Data Decoding” function classifies the information into bytes that belong to the state of interest for this project. As the information is decoded, the units for the angles are degrees, meters per second for the velocities, meters for the height and percentage for the battery. All of these data changes to double data type for easier processing of data on the controllers. The “status” bytes provide information as to flying or grounded condition. Appendix C shows the script developed in this function.

The “Function to Reset Yaw” is in control of resetting the yaw angle to zero degrees after the “Enable Commands” switch activates. Figure 16 shows the script written for this function.

```matlab
% Function to reset the yaw once the commands are enabled
function yawOut = fcn(isFlying, enableCmds, yaw)
    yawOut = yaw;
    persistent yaw0 ;
    persistent mode ;

    if isempty (yaw0)
        yaw0 = 0;
    end
    if isempty (mode)
        mode=0;
    end

    if isFlying ==1 && enableCmds==1
        yawOut = yaw - yaw0;
        mode =1;
    elseif isFlying ==1 && mode ==0
        yaw0 = yaw;
    end

    yawOut = mod(yawOut,360) ;
    if yawOut > 180
        yawOut = -360 + yawOut ;
    end
end
```

Figure 16. Reset Yaw function
After the quadcopter takes off, it creates a disturbance on the gyroscope, which makes the sensors receive information that the yaw angle has increased. This does not become a problem if the “Enable Commands” switch does not activate because the quadcopter is only maintaining the states. However, if the angle does not return to zero degrees, once that the switch activates, the quadcopter will immediately try to reach the angle from the disturbance created at take-off and the system will become unstable.

2.3.5 Position Estimation

The “Position Estimation” subsystem structure as shown in Figure 17 compiles all the information processed in the “AR.Drone Plant” that serves as the states of interest for analysis and control of the system. The vehicle velocities, Euler angles and height do not undergo any modifications from their information transmitted by the sensors. The velocities in the x and y-axis multiply a transformation matrix using Euler angles to obtain X and Y position of the drone after integrating these results.

Figure 17. Position Estimation subsystem
The initial developers claim that this subsystem provides only an estimation of the position because the velocities provided by the onboard flight computer may be inaccurate. This is where the developers recommend implementing patterns on the floor, as the downward camera can provide assistance estimating the velocity of the drone.

2.3.6 Visualization of States

As shown in Figure 18, the “Visualization of States” subsystem shows in real time six states of the quadcopter. Those states are $\phi$, $\theta$, $\psi$ angles on degrees and $X$, $Y$ and $Z$ on meters. The states of $u$, $v$ and $w$ appear along the previous six into a file directed to the workspace for analysis after the simulation has ended. The velocities appear on the file as $m/s$.

Figure 18. Visualization of States subsystem
3.1 EEG signals

EEG signals are electrical waveforms produced by the activity of the brain. These can be obtained through electrodes, which are conductors placed on the scalp. Electrodes usually do not make direct contact with scalp. Therefore, a conductive gel is applied between the electrode and the scalp to improve the acquisition of signals for a specific region of the brain. We utilize P300 Speller over other BMI’s, such as motor imagery because of the ability to select a higher number of commands with higher accuracy through EEG signals.

As previously mentioned, P300 Speller focuses on the brain activity presented on the occipital region of the brain during stimulus due to the flickering of a light source at different frequencies. Figure 19 shows the electrode placement on the EEG cap to obtain signals related to the P300 Speller interface of this research.

Figure 19. Electrode placement.
Based on the 10-20 system as shown in the previous figure, P3, Pz, P4, PO7, POZ, PO8, O1 and OZ are the eight channels acquiring the EEG signals. NZ marks the electrode for ground and A2 for reference signal.

3.2 BCI2000 Interface

BCI2000 is a software used for data acquisition of brain signals. The software allows obtaining data using hardware from the different distributors, such as g.tec, while other software and hardware could be used for the export or implementation of the data obtained, such as MATLAB. In this research, BCI2000 receives the information transmitted by the eight electrodes placed on the scalp of the users.

Figure 20 shows the BCI2000 configuration window needed for the interface. In the “Config” section, we set up the parameters needed for this research. The flickering rate, the length of data acquired for each letter, the display of the window and other signal conditions are all edited in this button. This section also allows loading a file that contains the information of the tasks planned for the software. In addition, a UDP is implemented to be able to acquire the information for external software or hardware. The “Set Config” option will load the parameters from the “Config” section into the software.

Figure 20. BCI2000 configuration window

P300 Speller is the module of BCI2000 utilized for the experiment. In this case, a window with matrix of three by three will show up after the “Start” button has been pressed. Each cell
contains a letter in the order of S, U, I, L, R, O, D and S. The matrix cell between the letter L and R will appear as blank on the window, but we will obtain an X letter if the interface selects that position.

3.2.1 Initial EEG Data Acquisition Mode

The acquisition of the initial EEG signals for training the simulation occurred with the implementation of the interface called “copy mode”. In this case, the BCI2000 is set up as explained before but without any connection to the MATLAB interface explained later in this chapter or any of the models of the previous chapter. The only difference on this interface shows up on the window flickering the letters. Figure 21 shows the copy mode window used to obtain the initial data. A predetermined and randomized list of the eight letters is presented at the top of the window. This will let the users know which letter they should focus next. While the user focuses their sight and attention to that letter, data will be obtained from that specific letter to train the interface.

![Figure 21. Copy mode window](image-url)
3.2.2 Training Mode

Six runs of the copy mode were executed on a user in order to obtain data for the training of the interface. Data for each letter was collected three times in each run. Once that a run is finished, a file is saved on the computer for further processing. Each of the runs were uploaded to a MATLAB tool called “P300GUI”, which is provided with the BCI2000 software. This tool creates graphs that demonstrate the activity in each of the channels (electrodes) during the flickering of the letters. Figure 22 shows the average of the six trials from the data acquisition. They are combined in order to obtain a better statistical sample of the signals. Appendix E shows the individual P300 Speller data set for each run. Red shows a higher activity, compared to blue showing lowering activity. In this case, channel six, which corresponds to the electrode PO7, demonstrates the highest activity. Then, the times of the highest r² values of the data are noted for the setup of the interface. All the values taken for the configuration of the interface are found above the quantity of .006 r².

![Graph showing r² values between conditions 1 and 2](image)

Figure 22. Average P300 Speller stimulus of six runs of data acquisition
Once that the times are noted, they are input on the BCI2000 “Config” Section. This helps the interface to take the times at which the highest stimulus will occur during the real-time control mode. Figure 23 shows the linear classification utilized for real-time control mode based on the values of the previous figure.

<table>
<thead>
<tr>
<th>Linear classification matrix in sparse representation</th>
</tr>
</thead>
<tbody>
<tr>
<td># of columns</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input channel</th>
<th>input element (bin)</th>
<th>output channel</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 23. Linear classification for real-time control mode

### 3.2.3 Real-time Control Mode

The window for the real-time control is similar to the copy mode window, but in this case, the predetermined list of letters does not show up. Letters will start showing up as a list on top of the window, similar to the copy mode window, after the interface has printed the specific letter that the user was focusing on. The list will extend as the simulation runs. As soon as the letter is printed in this window, that information will be sent to the MATLAB interface through the UDP connection.

Two cases were simulated for this research with the use of this real-time control mode window. In the first case, the BCI2000 interface was set up without any connection to the g.tec technology, as we were not acquiring data from brain signals. After the user clicks on the letter in this window, the selected letter prints after every five seconds, which is the time that takes to send
the commands to the quadcopter from the BCI2000 interface. For the case of control with EEG signals, letters were selected after every 14 seconds while using g.tec technology.

3.3 MATLAB Interface

The interface from BCI2000 is achieved through the UDP settings within the parameters of the software. Next, the UDP on a MATLAB script obtains the information of the letter selected by the user in form of a sentence. Appendix D shows the script developed for the MATLAB interface. Therefore, the script splits the sentence and obtains the letter selected. Then, a case activates according to the letter. Based on the case, the script writes a one on a digital pin for a high value and zeros for the rest of the pins. A pause was required between each pin command because the script would process faster than the ability to write the values on the pins.

Before running the previous script, the first “Arduino Mega 2560” microcontroller needs to be set up with the line of code “p=arduino('COM5','Mega2560’)”. The COM port corresponds to the connection of the board to the computer being utilized. This will allow writing the information into the digital pins of the board. Then, the output sends to the second “Arduino Mega 2560” through the connections of digital pins between each board. The “Signal Processing MATLAB/Simulink Model” will read the information transmitted from the second board as described on the previous chapter. Both Arduino boards are connected through universal serial bus (USB) to a single computer. The first Arduino board connects to the computer handling the EEG data acquisition and training, while the second one connects to the computer processing the “Signal Processing” and the “Parrot AR.Drone 2.0 MATLAB/Simulink Model”.

3.4 Signal Processing MATLAB/Simulink Model

The “Signal Processing” model processes the information coming from the EEG signals acquisition combination of systems explained in the following chapter. Figure 24 shows the
structure of the “Signal Processing” system in charge to read the signals coming from the Arduino board. The nine “Arduino Digital Input” blocks are available after downloading the “Simulink Support Package for Arduino Hardware” hardware support developed by MathWorks Simulink Team. Each block corresponds to a digital pin used on the Arduino board. The nine blocks correspond to the total number of letters available on the BCI2000 simulation. Then, a function classifies all nine outputs into a single output in order to send it to the following step on the simulation. A UDP port within the computer needs setup to create a connection between the models. As a result, the line “echotcpip('on',50000)” on the command window of MATLAB enables the UDP port 50,000. The port 50,001 on the “UDP Command Output” blocks corresponds to the remote UDP port where the next model receives the single output.

Figure 24. Signal Processing system
Certain configuration parameters need to be edited to run this model, such as “Arduino Mega 2560” in hardware board and the communication (COM) port number corresponding to the connection of the Arduino board into the computer. This port number can be verified through “Device Manager” of the computer operating system.

The function in this system allows to receive the information from the nine digital pins and classifying which of these nine values has a higher quantity than the rest. As the user selects a letter from the BCI2000 interface, the pin corresponding to that letter demonstrates a higher value than the rest. Each classification of the digital pins contains a specific output that will later serve as the command for the quadcopter. Figure 25 shows the “Command Classification” function.

```matlab
% Function to classify the command from multiple signals into a single output
function y = fcn(u1,u2,u3,u4,u5,u6,u7,u8,u9)
    if u1>u2 && u1>u3 && u1>u4 && u1>u5 && u1>u6 && u1>u7 && u1>u8 && u1>u9
        y=1; % If value of pin 2 is higher than the rest, print 1
    elseif u2>u1 && u2>u3 && u2>u4 && u2>u5 && u2>u6 && u2>u7 && u2>u8 && u2>u9
        y=2; % If value of pin 3 is higher than the rest, print 2
    elseif u3>u1 && u3>u2 && u3>u4 && u3>u5 && u3>u6 && u3>u7 && u3>u8 && u3>u9
        y=3; % If value of pin 4 is higher than the rest, print 3
    elseif u4>u1 && u4>u2 && u4>u3 && u4>u5 && u4>u6 && u4>u7 && u4>u8 && u4>u9
        y=4; % If value of pin 5 is higher than the rest, print 4
    elseif u5>u1 && u5>u2 && u5>u3 && u5>u4 && u5>u6 && u5>u7 && u5>u8 && u5>u9
        y=5; % If value of pin 6 is higher than the rest, print 5
    elseif u6>u1 && u6>u2 && u6>u3 && u6>u4 && u6>u5 && u6>u7 && u6>u8 && u6>u9
        y=6; % If value of pin 7 is higher than the rest, print 6
    elseif u7>u1 && u7>u2 && u7>u3 && u7>u4 && u7>u5 && u7>u6 && u7>u8 && u7>u9
        y=7; % If value of pin 8 is higher than the rest, print 7
    elseif u8>u1 && u8>u2 && u8>u3 && u8>u4 && u8>u5 && u8>u6 && u8>u7 && u8>u9
        y=8; % If value of pin 9 is higher than the rest, print 8
    elseif u9>u1 && u9>u2 && u9>u3 && u9>u4 && u9>u5 && u9>u6 && u9>u7 && u9>u8
        y=9; % If value of pin 10 is higher than the rest, print 9
    else
        y=0; % Any other case prints a value of 0
    end
end
```

Figure 25. Command Classification function
4.1 Equipment Overview

The EEG acquisition of signals was done using g.tec equipment from the department of biomedical engineering at Wichita State University. The g.GAMMAcap is the EEG cap that the user places on his head to accommodate the distribution of electrodes across the scalp. g.GAMMgel is placed inside of each electrode cavity for better conductivity. Each electrode is connected to the g.GAMMAbox, which serves as the power supply and interface for 16 communication channels. Then, the g.GAMMAbox connects to the g.USBAmp to amplify, acquire and process the signals. Lastly, this information transfers to the computer through a USB connection for analysis and applications. Figure 26 shows those three main components.

![Figure 26. Equipment to obtain EEG signals composed by (a) g.GAMMAcap, (b) g.USBAmp and (c) g.GAMMAbox](image)
Figure 27 shows the setup of the Arduino boards as described on the previous chapter. The two boards are required to send the information from the MATLAB command window to the Simulink models. This setup was needed because once that Simulink starts running the models, it does not receive real-time information from the MATLAB command window. Each board connects to the computer through USB connections. Nine jumper wires will connect the digital pins from two to 10 from one board to another.

The computer needed Wi-Fi capabilities to perform the experiment due to the UDP connection with the quadcopter. In addition, the experiment used MATLAB R2018b with Simulink, Aerospace Blockset, Simulink Check and Simulink Desktop Real-Time for the quadcopter processing, and MATLAB and Simulink Support Package for Arduino Hardware for the processing of signals coming from the electrodes. The experiment also included the Parrot AR.Drone 2.0 quadcopter with its respective batteries and some markers to be able to track the quadcopter with a motion tracking system. The “Vicon Tracker” motion tracking system will be
discussed in one of the following sections. In addition, as mentioned in the previous chapter, BCI2000 is the software dedicated for the acquisition and processing of EEG data.

4.2 Test Setup

The testing occurred in the Controls Laboratory located in the John Bardo Center at Wichita State University. This laboratory has two main sections; one bounded with a netted area with the purpose for testing and tracking objects, and another where all the equipment is kept away from possible collisions. The quadcopter and the obstacles were located inside of the netted area, but the rest of the equipment and test subjects were safely located outside of the netted area. The downward camera of the quadcopter has the ability to distinguish patterns beneath it and reduce drift. Therefore, a grid of 200 inches squared with subdivisions of 20 inches of separation was set up on the floor and inside of the netted area of the laboratory in order to help tracking of the quadcopter’s position. This can be seen on the next figure that shows the flight mission path.

One subject tested the controller with manual inputs and another subject controlled the quadcopter with EEG signals. As the subjects arrived to the laboratory, the premise of the research was explained and the subjects were given the opportunity to ask questions related to the experiment. Then, the users sat down in front of the computer where they were going to control the quadcopter. In the case of controlling the quadcopter through brain signals, the user was a 20-year-old right-handed male. Once the user felt in a comfortable position, the researchers placed the EEG cap and gel in the appropriate places. The electrodes were already in their respective positions before the EEG cap was placed on the subject. The possibility of external stimuli affecting the response of the subject resulted in the need of a controlled environment with minimal noise and visual distractions. Only for this simulation, the experiment occurred during the night, the lights of the laboratory and the corridor leading into the laboratory were turned off to minimize
the external visual stimuli. Some uncontrollable noise was present from the ventilation system of the building through the experiment.

### 4.3 Flight Mission

After setting up the grid, the testing involved a planned trajectory for the user to follow. Polyvinyl chloride (PVC) pipes and connectors form the structure of the obstacles and markers for the start/finish location. Figure 28 shows the trajectory and location of obstacles. The path starts in the square marker on the floor labeled “Start” as the quadcopter faces the entrance of the first obstacle. Then, the user would move in order from the first to third obstacle, and finish in a square marker on the floor labeled “Finish”. This trajectory covers the control of the three DOF, as it is required to move forward/backwards, adjust height, and rotate in each direction.

![Quadcopter planned trajectory](image)

Figure 28. Quadcopter planned trajectory

The reason to include obstacles resides on the premise to evaluate how reliable this technology and controller could be in real life scenarios. The flight mission was only tested with
the simulation of selecting the letter manually on the BCI2000 interface. Table 3 summarizes the dimensions of the obstacles and start/finish mark along the trajectory that the quadcopter needed to follow with commands implemented by the user.

**TABLE 3**

**DIMENSIONS OF OBSTACLES ALONG TRAJECTORY**

<table>
<thead>
<tr>
<th>Obstacle</th>
<th>Width</th>
<th>Length</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start/Finish</td>
<td>25.50”</td>
<td>25.50”</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>24.00”</td>
<td>44.50”</td>
<td>49.50”</td>
</tr>
<tr>
<td>2</td>
<td>24.00”</td>
<td>45.00”</td>
<td>38”</td>
</tr>
<tr>
<td>3</td>
<td>24.00”</td>
<td>44.50”</td>
<td>49.50”</td>
</tr>
</tbody>
</table>

### 4.4 Vicon Tracker

Vicon Tracker is a combination of hardware and software with the ability to track objects in a 3D environment. Figure 29 shows the camera distribution in the laboratory.

![Figure 29. Vicon Tracker camera setup](image-url)
This creates a virtual scenario grid visible on the computer that tracks objects with small spherical markers on them. Figure 30 shows the 3D virtual scenario and obstacle areas that the quadcopter must go through its flight mission path. The grid does not relate to the grid placed on the floor of the laboratory. These renderings appear real time by the Vicon Tracker system in the computer software. In order to obtain the representations as shown in the next figure, each corner of the obstacle had a marker. After placing the markers, the software allows you to create an object in the virtual scenario after only selecting the group of markers belonging to a specific object. In this case, green denotes the objects for obstacles from one to three while orange represents the objects for the markers of start and finish. The default option for the origin of the body-referenced frame is the centroid between all the markers placed on the object, and the orientation of its axis are along the axis coordinates of the virtual grid.

Figure 30. Flight mission obstacles on Vicon Tracker

When a recording occurs using this software, the resulting file only records the coordinate’s origin of the body-referenced frame. In addition, the creation of an object creates a file that contains the
coordinates of each marker placed on the objects’ surface. MATLAB will employ the coordinates obtained from the objects and flight recordings for further analysis.

Figure 31 shows the (a) placement of five markers along the surface of the quadcopter and its (b) representation on the Vicon Tracker system.

![Quadcopter (a) with markers and (b) representation on Vicon Tracker](image)

Figure 31. Quadcopter (a) with markers and (b) representation on Vicon Tracker

The only difference for this case is the modification of the body-referenced frame to follow the convention of Figure 1. The origin of the frame is still located at the centroid of the markers, but the axes set up has the necessary rotations. In the figure, the red axis represents +x, green for +y and blue for +z.
CHAPTER 5
RESULTS

5.1 Feedback Controller Response

In order to understand the stability of each state, each command was tested individually on a separate flight. The response of each command leads to the assumption that there is a closed-loop controller present on the inner-loop of the “AR.Drone 2.0 Plant” model. In other words, a PID controller or any other type of control is not required for the inner-loop as the on-board computer and processor of the quadcopter’s attempts to maintain stability. However, as shown in the next figures, the system could improve the response of its states. Despite the fact of observing noise and disturbance on some states, those were not notably observed during flight. Nevertheless, drifting was present while the quadcopter was performing certain actions, especially during a yaw command.

The values coming from the “EEG Command Controller” are constant, but they become rates while running the simulation because they are implemented along a period of time. The rates of change for $\phi$, $\theta$, $\psi$ and $w$, which are dependent on the values defined on the “Classification of input received” function of this controller, are summarized on Table 4. The response of these rates will be described for each command.

<table>
<thead>
<tr>
<th>TABLE 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>RATES OF CHANGE FOR STATES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Rate</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pitch</td>
<td>1.25</td>
<td>$deg/s$</td>
</tr>
<tr>
<td>Roll</td>
<td>0</td>
<td>$deg/s$</td>
</tr>
<tr>
<td>Yaw</td>
<td>.35</td>
<td>$deg/s$</td>
</tr>
<tr>
<td>Height</td>
<td>.1</td>
<td>$m/s$</td>
</tr>
</tbody>
</table>
5.1.1 Yaw Controller Response

Figure 32 shows the i, j and k values of the simulation and flight states while commanding a positive yaw command. In this case, the take off switch was activated at 5.5 s and the enable commands switch was activated at 12.9 s. The behavior between these two values present the offset disturbance when taking off described in a previous chapter. The positive yaw started being implemented at 14.8 s at a constant rate until the quadcopter reached 1 rad or 57.3°.

![Positive Yaw Command along Time](image)

Figure 32. Positive yaw command and response on the quadcopter

During the simulation, the greatest margin of error occurs during the yaw command. As the current state tries to track the command signal, the error percentage of 45.88% is equivalent to 2.73°. The commanded state is averagely reached .45 s after being sent to the quadcopter. Once that the signal reached the desired angle, the system shows an overshoot of .23°. As the simulation
progresses, the steady state error decreases up to .01°, but presents an oscillatory behavior. As seen in the previous figure, this state tracks the command very effectively.

Figure 33 shows the i, j and k values of the simulation and flight states while commanding a negative yaw command. For this simulation, the offset disturbance presents between take off at 7.8 s. and enable commands at 17 s. The negative yaw started being implemented at 20.8 s at a constant rate until the quadcopter reached -1 rad or -57.3°.

![Negative Yaw Command along Time](image)

**Figure 33.** Negative yaw command and response on the quadcopter

In this case, the greatest margin of error occurs during the yaw command. As the current state tries to track the command signal, the error percentage of 87.55% is equivalent to 5.53°. The commanded state is averagely reached .33 s after being sent to the quadcopter. Once that the signal reached the desired angle, the system shows an overshoot of 2.93°, which is greater than the
previous case. As the simulation continued, the steady state error does not decreases as desired because an average of 2.13° were present with an oscillatory behavior.

5.1.2 Pitch Controller Response

The pitch is one of the states that demonstrate the noise. These characteristics are most notable because the responses and gains occur in a smaller margin compared to the yaw response. Despite the fact that the signal contains a great quantity of noise without any signs of dissipation, the signal keeps bouncing around the zero value while a rate is not applied. This noise is mostly contained within .5°, but there are cases where spikes of 1.25° deviate from the desired value. Figure 34 shows the response to a positive rate on the pitch state. The quadcopter translated 2.58 m during 21.91 s, which means it moves forward at a velocity of .12 \( \text{m/s} \).

![Positive Pitch Command along Time](image)

Figure 34. Positive pitch command and response on the quadcopter

44
Figure 35 shows the response to a negative rate on the pitch state. The spike of .96° at .78 s demonstrate the disturbance at take-off, while the oscillation of 3.1° at 5.8 s occurs due to the activation of the enable commands switch. In this case, the quadcopter translated 1.99 m during 14.69 s, which means it moves backwards at a velocity of .14 \( m/s \).

**Negative Pitch Command along Time**

![Negative Pitch Command along Time](image)

Figure 35. Negative pitch command and response on the quadcopter

Despite the fact of applying the same rate to the previous commands, there is a difference of .02 \( m/s \) on the commands. Several flights demonstrated that the negative pitch tends to perform at a higher rate. The current value of the pitch follows the pattern to increase or decrease based on the command, but it never reaches the desired rate. An input to the system produces 40% of the desired value, which means a 60% error percentage below steady state. The small gains of 1.25° represent the limit for the forward and backward velocity from the “EEG Command Controller”.
5.1.3 Roll Controller Response

The roll is the other state where noise appears as a factor during flight. Similar to pitch, the state operates on small values, but in this case, a constant command of zero is implemented throughout the entire flight. Figure 36 shows the response feedback from the roll state along a flight with no other input being applied. The disturbance of .55° at .85 s occurs at take-off, while the oscillation of 2.73° at 5.1 s occurs due to the activation of the enable commands switch. Once that the major vibrations have settled down, the steady state error oscillates around .35°.

Figure 36. Roll command and response on the quadcopter

5.1.4 Height Controller Response

The height operates different to the other states in the sense that the command input is the same as the current state as long as there is no desired to increase or decrease the quantity. Since
there is no reference value for height, the error or difference value corresponds to the rate of change selected on the controller. This makes the steady state and overshoot to behave in a different way compared to the previous cases due to the controller incorporated in the computer’s processor. Figure 37 shows the flight starting from the ground, going through take-off and increasing the height to maintain a steady state. After take-off, the height reaches a quantity, but does not overshoot. Then, the quantity drops and reaches steady state after 7.99 s of the initial spike. As the controller is trying to track the desired signal, it takes the quadcopter an average of 1.17 s to reach the command.

**Figure 37. Increase height command and response on the quadcopter**

Figure 38 shows the flight starting from a high point position, decrease the height, maintain a steady state and then proceed to land. The pattern follows the similar quantities and rates than an
increase height command. In this case, it takes an average of 1.3 s to reach the desired value from the moment the command was implemented. Once on steady state, the height oscillates within a range of .053 m.

**Decrease Height Command along Time**

![Decrease Height Command along Time](image)

**Figure 38. Decrease height command and response on the quadcopter**

In every flight, the importance of the delay between the activation of the take-off and enable commands switch resides on the quadcopter timing to designate the front and back. The designation is always the same, but this helps the system to process the commands faster.

### 5.2 Flight Mission

The flight mission to complete was attempted with manual inputs simulating the selection of letters by brain signals. The setup described in the previous chapter with obstacles and markers was plotted into MATLAB. Figure 39 shows the flight path as recorded by the Vicon Tracker.
Figure 39. Isometric view of the flight mission path
The blue square denotes the start marker and the red rectangles represent the first and second obstacle. The third and finish marker do not show in the figure, as the quadcopter did not complete the mission. The dot inside of the markers marks the centroid of the object. Lastly, the path of the quadcopter shows as a line based as the movement of its markers’ centroid. The quadcopter successfully passed through the first and second obstacle, but while rotating to align with the third obstacle, the quadcopter had an overshoot of above 180°, the system became unstable and it attached itself to the second obstacle. After this, the command to land was pressed on the Simulink model to stop the simulation, as the quadcopter could not detached from the PVC pipe. The main challenge of the flight mission was the ability of the user to compensate for the drifting presented on the system. The maneuvers to align the quadcopter to pass the second obstacle were manageable, but they needed a quick reaction time to avoid the system to get in a situation that it was going to be complicated to overcome.

The Vicon Tracker system also served as validation of the “Position Estimation” block of the “Parrot AR.Drone 2.0 MATLAB/Simulink” model. Figure 40 shows the comparison of path as recorded by the Vicon Tracker (red) and by the Simulink model (black). When comparing the paths of both methods of data acquisition, it can be seen that both system almost recorded the same patterns of movement across the 3D environment. Nevertheless, it is clear that as the quadcopter moves along the trajectory, distances are noticeable not aligned to each other. In this case, the Vicon Tracker is considered as a more accurate of the quadcopter’s path along the flight mission, while the result from the “Position Estimation” subsystem is inaccurate, but gives a general idea of the location. Both systems start recording the initial position in the same location. However, the final position is .283 m in the x-direction and 1.042 in the y-direction. If we account for the percentage error for each direction, we have 14.5% in the x-direction and 34% in the y-direction.
Figure 40. Position comparison between Vicon Tracker and Simulink
The inaccuracies of the “Position Estimation” in these two directions are not considered a problem for the system since this information does not go through the feedback loop control system. As for the height, a couple of points were considered to calculate the error. The Vicon Tracker system recorded an altitude of 1.54 m when the quadcopter attached to the second obstacle, while the ultrasound sensor on the quadcopter recorded 1.52 m. This state only shows an error of 1.23%, which provides a good reliability on the feedback for the controller.

Appendix F offers graphs in more detail about the results discussed in this chapter. Plane views of the flight mission path are shown from Figure 52 to 54, plane views for the comparison of position systems between the Vicon Tracker and Simulink model are shown from Figure 55 to 57, and the data of states acquired through the flight mission are shown from Figure 59 to 62. The commands graph shows the value of the letter selected on the BCI2000 interface.

5.3 EEG Flight Control

The flight with EEG signals as the input for the control of the quadcopter occurred in the 3D environment of the laboratory with no obstacles. A path was not planned as the goal for this flight was to understand the behavior of each individual command. Figure 41 shows the commands selected on the BCI2000 Interface and sent through the simulation until reaching the quadcopter. All the possible letters were selected in this flight. The red dotted line represents the total duration of each command as sent by the BCI2000 until the next command was received in this interface. The command filtered with the time limit sent to the quadcopter is shown as a black dashed line. A total duration of each command averaged around 15.3 s from the BCI2000 until the next letter was printed, while the command sent to the quadcopter averaged around three and a half seconds before returning to a hovering position. A letter was placed above the numeric value command to easily understand the letter printed on the interface.
Figure 41. Commands sent through the quadcopter along the EEG flight control

Figure 42 shows the yaw response to the commands selected by EEG signals. Based on the previous figure, the negative yaw starts as expected at 51.35 s into the flight and lasts for 3.45 s.

Figure 42. Yaw response along the EEG flight control
The positive yaw starts at 116.1 s and lasts for 3.4 s. These actions returned the quadcopter almost to the current position in terms of the yaw rotation. However, there is a disturbance between these commands due to the collision with one of the limits of the 3D environment.

Figure 43 shows the pitch response to the commands selected with EEG signals along the flight. The two largest spikes in the middle of the flight occurred at the time of the collision with the boundaries of the flight area. Based on the commands graphs and verified with this figure, the negative pitch occurs at 147.4 s into the flight and lasts about 3.4 s. The positive pitch is implemented at 195.8 s and lasts for 3.5 s. Similar to the test of individual commands earlier in this chapter; the behavior of not reaching the fully desired command state is still present.

![Pitch response along the EEG flight control](image)

Figure 43. Pitch response along the EEG flight control

Figure 44 shows the roll response along the flight controlled with EEG signals. As expected, there is no command received on this state as the controller attempts to maintain the state closer to zero. The spike of 15.7° occurs at the time of the collision.
Figure 44. Roll response along the EEG flight control

Figure 45 shows the height response along the flight. Two commands to increase and one to decrease height were implemented through the flight.

Figure 45. Height response along the EEG flight control
The first command to increase the height occurred at 34.77 s into the flight and lasted for 3.45 s. The command to decrease the height occurred at 85.41 s, but only lasted 2.5 s because the simulation understood that the quadcopter was reaching an altitude lower than 1 m. This is based on the constraint implemented on the “EEG Command Controller”. The two spikes after this command occurred at the time of the collision with the boundaries. The second command to increase the height occurs at 162.1 s and lasts for 3.5 s.

The flight trajectory for this flight is not shown, as the purpose was not to demonstrate that the quadcopter could move competently around the 3D environment. The sole purpose was to show that the commands were received from the EEG data acquisition to the commands implementation.
CHAPTER 5
CONCLUSION

A reliable model of the quadcopter was developed and successfully simulated in real time through Simulink. Further work could improve the interaction between Simulink and the AR.Drone 2.0 or create a drone with better equipment. However, this drone was chosen due to the simplicity of building a model in Simulink and prove the concept that a BMI could control a quadcopter efficiently. This BMI demonstrated the control of three DOF of the quadcopter in a three dimensional (3D) environment. The researcher tested the three DOF individually before a user tested them on a flight mission path. Despite the fact that the user wanted to control the roll to simplify the movement of the quadcopter to reach the next obstacle, the control of pitch, yaw and height were sufficient to complete two obstacles of a course. The same effect of roll was achieved through a yaw rotation, a pitch command and yaw rotation opposite to the first one.

In order to be considered a BMI, the quadcopter needed to establish a communication with BCI2000. This software will utilize P300 Speller for future research with the BMI developed in this research. A MATLAB script and Simulink were developed and tested to establish a communication between the BCI2000 and the quadcopter Simulink model. A delay could be perceived in computers with low processing capabilities or when the entire simulation was run in the same computer, but the setup described in this research achieved a delay that was not noticeable when selecting the commands from the BCI2000 interface and implementing them on the quadcopter.

The inner-loop quadcopter state controller counted with an embedded controller for the stability of the system that helped to obtain an enough adequate response of the system for the different types of motion. While the communication in the inner-loop occurs in the drone itself,
the Simulink model was required to establish a communication and conversion of data between the commands and the quadcopter. The development of the outer-loop EEG control proved to perform as desired. The feedback loop controller with unity gain in the Simulink model serves the function of a tracking controller. Besides the noise or disturbances of the states, and the efficiency of the pitch command, the tracking controller eventually settles down at the desired steady-state.

5.1 Technology Reliability

After testing the outer-loop EEG controller, it can be said that it has potential to be included into flight control systems. A lot of research and development needs to be performed before being implemented into more complex systems, such as airplanes, but this research hopes to demonstrate that aviation could include neuro-robotics technology. Quadcopters are a starting point, as these systems tend to be more stable. On the other hand, EEG technology also needs to advance in the efficiency of data acquisition and interpretation in order to achieve a complete accuracy and not put at risk lives.

The interface of this study has been validated to be a successful BMI by establishing a communication in real time from the simulation of letter selection with brain signals to the actual movement of the quadcopter. Even though the pitch state only reached around 40% of the steady state, the desired commands sent to the quadcopter were performed with a 100% accuracy. However, when applying this development to the actual EEG signals acquisition, the accuracy of the desired states will only be as accurate as the brain interface recognizes the letter that the user was focusing on.

5.2 Future Work and Recommendations

In airplanes, coupling occurs when the movement about one axis causes disturbance about another due to asymmetrical mass distribution across the body. In this case, the quadcopter would
also create coupling due location of sensors, in-board computer, other hardware and an irregular shaped frame. This could be one of the main contributors to the noise obtained on the quadcopter’s states or the small quantity of drift on its motion. Further development could consider coupling in the inner-loop controller. Consequently, the stability could achieve a response closer to steady state. Other contributors to the noise and disturbance on the signal acquired by the sensors could be attributed to the condition of the quadcopter’s frame and other components not completely fixed to the main body.

For this study, MATLAB/Simulink does not support the stream of video format H.264 coming from the front camera of the quadcopter. MATLAB add-ons helped to obtain a stream from the quadcopter’s camera through a script independently from the Simulink model, but they cannot operate at the same time because both would try to use the same IP address. Other attempts successfully obtained a data array within the Simulink model processing with help of UDP connection blocks and port 5555, but the conversion into live pictures was not successful. This issue could be explore in more depth with the purpose to simplify the use of devices.

Through testing, the Vicon Tracker system in the laboratory presented some small vibrations or disturbances while recording data due to external systems coming from the building. This could produce a great impact on systems that are more critical on the information obtained by the Vicon Tracker, such as trajectory planning systems that obtain feedback from the cameras or systems that could pose a risk due to distances variations within one inch. These disturbances could also make a difference in the validation when comparing the position estimation of the Vicon Tracker and Simulink.

Currently, the entire simulation requires two computers for the entire simulation. One computer processes the BCI2000 and MATLAB interface, while the other computer manages the
Simulink Interfaces. This simulation could be simplified to avoid multiple steps between the acquisition of brain signals and sending the commands to the quadcopter. Nevertheless, the setup of two computers helped the systems to compute and process the information faster than performing all the simulation in only one computer.
REFERENCES


APPENDIX A

DRONE STATE REQUEST FUNCTION

```matlab
function [dataControl, isFlying] =
  fcn(status, fly, stop, enableRefs, HeightRef, rollAngRef, pitchAngRef, yawRateRef)

  persistent SequenceNumber;

  if isempty(SequenceNumber)
    SequenceNumber = 1;
  end

  isFlying = status(32);

  strOut = uint8(zeros(150, 1));

  cmd = ''; % String that contains a variable size command.
  cmd is concatenated with previous commands at each sample

  if (SequenceNumber <= 1 || mod(SequenceNumber, 100) == 0)
    SequenceNumber = SequenceNumber + 1;
    strCmd = getString(SequenceNumber);
    cmd = ['AT*CONFIG=' strCmd ', "general:navdata_demo", "TRUE" ' char(13)];
  end

  SequenceNumber = SequenceNumber + 1;
  strCmd = getString(SequenceNumber);
  cmd = [cmd 'AT*COMWDG=' strCmd char(13)];

  if stop == 1 % Emergency stop
    cmd = ['AT*REF=' strCmd ',290717952' char(13)];
  else % Not emergency stop
    if SequenceNumber > 20 && SequenceNumber < 22
      % Trim vehicle state estimation / angles set to zero
      SequenceNumber = SequenceNumber + 1;
      strCmd = getString(SequenceNumber);
      cmd = [cmd 'AT*FTRIM=' strCmd char(13)];
    end
  end
```

elseif SequenceNumber >=22 && SequenceNumber <25
    cmd = [cmd 'AT*CONFIG=' strCmd '"conrol:altitude_max","100000"' char(13)] ;
elseif SequenceNumber >=25
    if ( fly == 1 )
        if (isFlying ==0)
            SequenceNumber =SequenceNumber+1;
            strCmd = getString(SequenceNumber) ;
            cmd = [cmd 'AT*REF=' strCmd ',290718208' char(13)] ;
        elseif isFlying==1 & & enableRefs==1
            SequenceNumber =SequenceNumber+1;
            strCmd = getString(SequenceNumber) ;
            cmd = [cmd 'AT*PCMD=' strCmd ',1,' ] ;
        end
    aux = double (float2IEEE754 (rollAngRef) ) ;
    strAux = getString(abs(aux)) ;
    if rollAngRef<0
        strAux = [char(45) strAux] ;
    end
    cmd = [cmd strAux ','] ;
    aux = double (float2IEEE754 (pitchAngRef) ) ;
    strAux = getString(abs(aux)) ;
    if pitchAngRef<0
        strAux = [char(45) strAux] ;
    end
    cmd = [cmd strAux ','] ;
    aux = double(float2IEEE754 (HeightRef)) ;
    strAux = getString(abs(aux)) ;
    if HeightRef<0
        strAux = [char(45) strAux] ;
    end
    cmd = [cmd strAux char(13)] ;
    aux = double (float2IEEE754 (yawRateRef) ) ;
    strAux = getString(abs(aux)) ;
    if yawRateRef<0
        strAux = [char(45) strAux] ;
    end
    cmd = [cmd strAux char(13)] ;
end
elseif (fly==0)
    if isFlying ==1
        SequenceNumber = SequenceNumber + 1;
        strCmd = getString(SequenceNumber);
        cmd = [cmd 'AT*REF=' strCmd ',290717696' char(13)];
    end
    end
end

if (length(cmd)<150) % cmd must contain less than 150 bytes.
    strOut(1:length(cmd)) = cmd;
end
dataControl = strOut;
end

Function str2num that allows for code generation

function [strOut] = getString(a)
    d=1;
    if (a<10)
        d=1;
    end
    if (a >=10 && a <100)
        d=2;
    end
    if (a >=100 && a <1000)
        d=3;
    end
    if (a >=1000 && a <10000)
        d=4;
    end
    if (a >=10000 && a <100000)
        d=5;
    end
    if (a >=100000 && a <1000000)
        d=6;
    end
    if (a >=1000000 && a <10000000)
        d=7;
    end
end
\begin{verbatim}
d=8;
end
if (a >=100000000 && a <1000000000)
d=9;
end
if (a >=1000000000 && a <10000000000)
d=10;
end
if (a >=10000000000 )
d=11;
end
digit = uint8(zeros(1,d)) ;
resTotal = 0;
for k=1:d
    res1 = mod(a,10^k)/(10^(k-1)) ;
digit(d-k+1) = char(48+res1);
    resTotal = (10^(k-1))*res1+resTotal;
end
strOut = char(digit) ;
end
\end{verbatim}

Function to obtain a IEEE754 representation of a float number

\begin{verbatim}
function [ iVal ] = float2IEEE754( fVal )
    binData = dec2bin(typecast (single(fVal), 'uint8'),8)
    binData2 = [ binData(4,:) binData(3,:) binData(2,:) binData(1,:)]
    iVal =0;
    for k=32:-1:1
        if (binData2(k) =='1')
            bit = 1;
        else
            bit =0;
        end
        if k == 1
            iVal = iVal - bit*2^(31);
        else
            iVal = iVal + bit*2^(32-k) ;
        end
    end
    iVal = int32(iVal)
end
\end{verbatim}
function frameOut = fcn(dataIn)

    % Persistent variables
    persistent frame;
    persistent lastFrame;
    persistent count;
    persistent countHeader;
    persistent countBuffer;
    persistent buffer;

    sizePayload = 496; % Size of byte array containing the data
    nBytes = 500; % Size of the array read each time step

    % Initialization of persistent variables
    if isempty(buffer)
        buffer=uint8(zeros(1024,1));
    end
    if isempty(countBuffer)
        countBuffer=0;
    end
    if isempty(frame)
        frame=uint8(zeros(sizePayload,1));
    end
    if isempty(lastFrame)
        lastFrame=uint8(zeros(sizePayload,1));
    end
    if isempty(count)
        count=0;
    end
    if isempty(countHeader)
        countHeader=0;
    end

    % Filling the buffer
    buffer(countBuffer+1:countBuffer+nBytes) = dataIn;
    bytesRead = countBuffer+nBytes;

    % Reading buffer
    for k=1:bytesToRead
        if(countHeader==4 && count<sizePayload)
            bytesLeft = sizePayload-count;
            frame(count+1:sizePayload) = buffer(k:k+bytesLeft-1);
count = sizePayload;
    countBuffer = bytesRead-bytesLeft;
    buffer(1:countBuffer) = buffer(k+bytesLeft:bytesToRead);
    break;
end
if(countHeader==3 && buffer(k)==85)
    countHeader=4;
    if(bytesRead-k > sizePayload)
        count =sizePayload;
        frame(1:count) = buffer(k+1:k+sizePayload);
        countBuffer =bytesRead-k-sizePayload ;
        buffer(1:countBuffer) = buffer(k+sizePayload+1:bytesToRead);
    else
        count =bytesRead-k;
        if count > 0
            frame(1:count) = buffer(k+1:bytesToRead);
        end
        countBuffer = 0;
    end
    break;
end
if(countHeader==2 && buffer(k)==102)
    countHeader=3;
end
if(countHeader==1 && buffer(k)==119)
    countHeader=2;
end
if(countHeader==0 && buffer(k)==136)
    countHeader=1;
end
if count==sizePayload
    frameOut = frame;
    flagOut = 1;
    lastFrame = frame;
    frame=uint8(zeros(sizePayload,1));
    count=0;
    countHeader=0;
else % In case the buffer does not have an entire payload
    frameOut = lastFrame;
    flagOut = 0;
end
function [status,battery,yawAngle,u,v,w,height,rollAngle,pitchAngle] = fcn(data)

% Checksum array. cksum(1) contains the computed checksum, cksum(2) contains the
checksum sent by the ARDrone.
cksum = [0 0];

persistent hPrev; % Previous value of height
persistent dataPrev; % Previous data packet

% Initialization of persistent variables
if isempty(hPrev)
    hPrev=0 ;
end
if isempty(dataPrev)
    dataPrev = uint8(zeros(496,1)) ;
end

% Computing checksum
cksum(1) =  sum(double(data(1:end-8))) +442;
% Checksum given by the drone
cksum(2) = double(data(end-3)) + double(data(end-2))*256 + double(data(end-1))*256^2 +
double(data(end))*256^3 ;
% Checksum error
cksumError = abs(cksum(1)-cksum(2));
% Change the following if statement to ensure that the data checksum is valid
if (data(15) == 148)
    dataPrev = data;
else
    data = dataPrev;
end
% Converting bytes to drone information
status = getDroneState(data(1:4));
battery = decode(data(21:24),1); % unit: percent
pitchAngle = decode(data(25:28),0)/1000; % unit: deg
rollAngle = decode(data(29:32),0)/1000; % unit: deg
yawAngle = decode(data(33:36),0)/1000; % unit: deg
height = decode(data(37:40),0)/1000; % unit: m
u = decode(data(41:44),0)/1000; % unit: m/s
v = decode(data(45:48),0)/1000; % unit: m/s
w = decode(data(49:52),0)/1000; % unit: m/s
if (status(32) ==1 && height< 0.2)
    height = hPrev;
elseif status(32)==1
    hPrev= height;
end
end

Function to classify and convert states bytes to information

function res = decode(input, mark)
    hex_value = dec2hex(input);
    hex_value = [hex_value(4,:),hex_value(3,:),hex_value(2,:),hex_value(1,:)];
    if mark ==1
        res = hex2dec(hex_value);
    else
        res = typecast(uint32(hex2dec(hex_value)), 'single');
    end
end

Function to classify and convert the status bytes to information

function bin = getDroneState(input)
    bin = zeros(32,1);
    hex = dec2hex(input);
    hex = [hex(4,:),hex(3,:),hex(2,:),hex(1,:)];
    for i=1:length(hex)
        if hex(i)=='F'
            bin((i*4)-3:i*4)=[1 1 1 1];
        elseif hex(i)=='E'
            bin((i*4)-3:i*4)=[1 1 1 0];
        elseif hex(i)=='D'
            bin((i*4)-3:i*4)=[1 1 0 1];
        elseif hex(i)=='C'
            bin((i*4)-3:i*4)=[1 1 0 0];
        elseif hex(i)=='B'
            bin((i*4)-3:i*4)=[1 0 1 1];
        elseif hex(i)=='A'
            bin((i*4)-3:i*4)=[1 0 1 0];
        elseif hex(i)=='9'
            bin((i*4)-3:i*4)=[1 0 0 1];
        elseif hex(i)=='8'
            bin((i*4)-3:i*4)=[1 0 0 0];
        end
    end
APPENDIX C (continued)

```matlab
elseif hex(i)=='7'
    bin((i*4)-3:i*4)=[0 1 1 1];
elseif hex(i)=='6'
    bin((i*4)-3:i*4)=[0 1 1 0];
elseif hex(i)=='5'
    bin((i*4)-3:i*4)=[0 1 0 1];
elseif hex(i)=='4'
    bin((i*4)-3:i*4)=[0 1 0 0];
elseif hex(i)=='3'
    bin((i*4)-3:i*4)=[0 0 1 1];
elseif hex(i)=='2'
    bin((i*4)-3:i*4)=[0 0 1 0];
elseif hex(i)=='1'
    bin((i*4)-3:i*4)=[0 0 0 1];
elseif hex(i)=='0'
    bin((i*4)-3:i*4)=[0 0 0 0];
end
end
end
```
APPENDIX D

LETTER SELECTED INTO MATLAB SCRIPT

```matlab
u = udp('localhost',20349,'LocalPort',20350,'Terminator','CR/LF','Timeout',100);
fopen(u);
s = fgetl(u);
while true
    s = fgetl(u);
    C = strsplit(s);
    O = C(1,2);
    A = cell2mat(O);
    switch A
        case 'S'
            i=1
            writeDigitalPin(p,'D2',1)
            pause(0.5)
            writeDigitalPin(p,'D3',0)
            pause(0.5)
            writeDigitalPin(p,'D4',0)
            pause(0.5)
            writeDigitalPin(p,'D5',0)
            pause(0.5)
            writeDigitalPin(p,'D6',0)
            pause(0.5)
            writeDigitalPin(p,'D7',0)
            pause(0.5)
            writeDigitalPin(p,'D8',0)
            pause(0.5)
            writeDigitalPin(p,'D9',0)
            pause(0.5)
            writeDigitalPin(p,'D10',0)
            pause(0.5)
        case 'U'
            i=2
            writeDigitalPin(p,'D2',0)
            pause(0.5)
            writeDigitalPin(p,'D3',1)
            pause(0.5)
            writeDigitalPin(p,'D4',0)
            pause(0.5)
            writeDigitalPin(p,'D5',0)
            pause(0.5)
            writeDigitalPin(p,'D6',0)
```
APPENDIX D (continued)

```c
pause(0.5)
writeDigitalPin(p,'D7',0)
pause(0.5)
writeDigitalPin(p,'D8',0)
pause(0.5)
writeDigitalPin(p,'D9',0)
pause(0.5)
writeDigitalPin(p,'D10',0)
pause(0.5)

case 'I'
i=3
writeDigitalPin(p,'D2',0)
pause(0.5)
writeDigitalPin(p,'D3',0)
pause(0.5)
writeDigitalPin(p,'D4',1)
pause(0.5)
writeDigitalPin(p,'D5',0)
pause(0.5)
writeDigitalPin(p,'D6',0)
pause(0.5)
writeDigitalPin(p,'D7',0)
pause(0.5)
writeDigitalPin(p,'D8',0)
pause(0.5)
writeDigitalPin(p,'D9',0)
pause(0.5)
writeDigitalPin(p,'D10',0)
pause(0.5)

case 'L'
i=4
writeDigitalPin(p,'D2',0)
pause(0.5)
writeDigitalPin(p,'D3',0)
pause(0.5)
writeDigitalPin(p,'D4',0)
pause(0.5)
writeDigitalPin(p,'D5',1)
pause(0.5)
writeDigitalPin(p,'D6',0)
pause(0.5)
```
writeDigitalPin(p,'D7',0)
pause(0.5)
writeDigitalPin(p,'D8',0)
pause(0.5)
writeDigitalPin(p,'D9',0)
pause(0.5)
writeDigitalPin(p,'D10',0)
pause(0.5)
case 'X'
i=5
writeDigitalPin(p,'D2',0)
pause(0.5)
writeDigitalPin(p,'D3',0)
pause(0.5)
writeDigitalPin(p,'D4',0)
pause(0.5)
writeDigitalPin(p,'D5',0)
pause(0.5)
writeDigitalPin(p,'D6',1)
pause(0.5)
writeDigitalPin(p,'D7',0)
pause(0.5)
writeDigitalPin(p,'D8',0)
pause(0.5)
writeDigitalPin(p,'D9',0)
pause(0.5)
writeDigitalPin(p,'D10',0)
pause(0.5)
case 'R'
i=6
writeDigitalPin(p,'D2',0)
pause(0.5)
writeDigitalPin(p,'D3',0)
pause(0.5)
writeDigitalPin(p,'D4',0)
pause(0.5)
writeDigitalPin(p,'D5',0)
pause(0.5)
writeDigitalPin(p,'D6',0)
pause(0.5)
writeDigitalPin(p,'D7',1)
pause(0.5)
writeDigitalPin(p,'D8',0)
pause(0.5)
writeDigitalPin(p,'D9',0)
pause(0.5)
writeDigitalPin(p,'D10',0)
pause(0.5)

```c
    case 'O'
        i=7
        writeDigitalPin(p,'D2',0)
pause(0.5)
        writeDigitalPin(p,'D3',0)
pause(0.5)
        writeDigitalPin(p,'D4',0)
pause(0.5)
        writeDigitalPin(p,'D5',0)
pause(0.5)
        writeDigitalPin(p,'D6',0)
pause(0.5)
        writeDigitalPin(p,'D7',0)
pause(0.5)
        writeDigitalPin(p,'D8',1)
pause(0.5)
        writeDigitalPin(p,'D9',0)
pause(0.5)
        writeDigitalPin(p,'D10',0)
pause(0.5)
    ```

```c
    case 'D'
        i=8
        writeDigitalPin(p,'D2',0)
pause(0.5)
        writeDigitalPin(p,'D3',0)
pause(0.5)
        writeDigitalPin(p,'D4',0)
pause(0.5)
        writeDigitalPin(p,'D5',0)
pause(0.5)
        writeDigitalPin(p,'D6',0)
pause(0.5)
        writeDigitalPin(p,'D7',0)
pause(0.5)
        writeDigitalPin(p,'D8',0)
pause(0.5)
        writeDigitalPin(p,'D9',0)
pause(0.5)
        writeDigitalPin(p,'D10',0)
pause(0.5)
```
writeDigitalPin(p,'D9',1)
pause(0.5)
writeDigitalPin(p,'D10',0)
pause(0.5)
case 'S'
i=9
writeDigitalPin(p,'D2',0)
pause(0.5)
writeDigitalPin(p,'D3',0)
pause(0.5)
writeDigitalPin(p,'D4',0)
pause(0.5)
writeDigitalPin(p,'D5',0)
pause(0.5)
writeDigitalPin(p,'D6',0)
pause(0.5)
writeDigitalPin(p,'D7',0)
pause(0.5)
writeDigitalPin(p,'D8',0)
pause(0.5)
writeDigitalPin(p,'D9',0)
pause(0.5)
writeDigitalPin(p,'D10',1)
pause(0.5)
end
end
fclose(u);
delete(u);
APPENDIX E

INDIVIDUAL P300 SPELLER STIMULUS DATA SETS

Figure 46. P300 Speller data set #1
APPENDIX E (continued)

Figure 47. P300 Speller data set #2

Figure 48. P300 Speller data set #3
Figure 49. P300 Speller data set #4

Figure 50. P300 Speller data set #5
Figure 51. P300 Speller data set #6
APPENDIX F

FLIGHT MISSION GRAPHS

Figure 52. X-Z plane view of the flight mission path
APPENDIX F (continued)

Figure 53. Y-Z plane view of the flight mission path

Figure 54. X-Y plane view of the flight mission path
APPENDIX F (continued)

Figure 55. X-Z plane view of position comparison between Vicon Tracker and Simulink

Figure 56. Y-Z plane view of position comparison between Vicon Tracker and Simulink
Figure 57. Y-Z plane view of position comparison between Vicon Tracker and Simulink

Figure 58. Commands sent through the quadcopter along the flight mission
Figure 59. Yaw response along the flight mission

Figure 60. Pitch response along the flight mission
Figure 61. Roll response along the flight mission

Figure 62. Height response along the flight mission