REMNANT-BASED METRICS FOR SIMULATOR FIDELITY

Tyler Wei, Kirill Zaychik

Department of Mechanical Engineering
Thomas J. Watson School of Engineering and Applied Science
State University of New York, Binghamton

Submitted: Friday, May 12th, 2016

Project Sponsored by the NASA – NY Space Grant Fellowship

ACKNOWLEDGMENTS

I would like to thank the former contributions to this study exercised by Noah B. Singer, Nathan Sprenkle, and Dr. Frank Cardullo. I would also like to thank the participating members of the 33rd Annual Flight and Ground Vehicle Simulation Course at Binghamton University.

REMNANT-BASED METRICS FOR SIMULATOR FIDELITY

Tyler Wei, Kirill Zaychik
Department of Mechanical Engineering
Thomas J. Watson School of Engineering and Applied Science
State University of New York, Binghamton

ABSTRACT

The examination of realism and fidelity for scientific modelling and simulation is critical to the advancement of developing better models of teleoperation in extreme conditions. For more than over a century, the subject of teleoperation has been investigated in the field of man-machine systems. Teleoperation, the method of remotely controlling a device from a distance has many practical applications, namely operations in outer-space which are commandeered from Earth. One of the largest issues which pose to be problematic in Earth-Space teleoperations is that over large distances, signal delays exist. This particular study explores the effects of a time and signal delay on the ability of the operator to perform the teleoperation task. The task, manipulation of a robotic inceptor to dock on a marked target on an orbiting satellite in outer-space around Earth through a computer simulation, had various time delays between 0 milliseconds to 2000 milliseconds. The computer simulation was developed in MATLAB and the experiment was conducted by having 20 subjects each undergo 15 different trials of the simulation with various time parameters changed each time. From the testing and analysis, it was revealed that as the signal delays increase, there is more chaotic motion as well as uncertainty in the pilot control behavior.

INTRODUCTION

Since the late 19th century, teleoperation has been a topic of interest as pioneered by Nikola Tesla to control from a distance, the operation of propelling engines, the steering apparatus, and other mechanisms carried by moving bodies[1]. Teleoperation is defined as a method of controlling a device from a distance and it allows for human control and remote manipulation of objects in extreme or hazardous environments. One such extremity is outer-space where the vacuum, the lack of protection from radiation, and extreme temperature changes which make teleoperation ideal[3].

In vast distances such as that from a control station on Earth to geosynchronous transfer orbit, the delay is merely a fraction of a second. However, to the moon, the delay increases to 1.3 seconds; this delay becomes minutes between planets, and then years between stars as distances increase[4]. The experiment outlined aims to construct a better understanding on how communication signal delays affects the operator performance in a teleoperation situation via testing wider ranges of delays with higher precision[3].

Current research has been concentrated on investigating the effects of human interaction with exposure to virtual environments such as vehicle simulators. Typically, these simulators are utilized to remotely operate equipment or vehicles over large distances, namely from earth to outer-space. It is well known that the existing time delays between the simulator operation and the real-time vehicle movement may affect the operator performance. However, there is little data on how the different ranges of the signal

[1] N. Tesla, *Patent US613809 – Method of and Apparatus for Controlling Mechanism of Moving Vessels or Vehicles* (1898)
[2] Kirill Zaychik and Frank Cardullo. *Simulator Sickness: The Problem Remains*, AIAA Modeling and Simulation Technologies Conference and Exhibit, Guidance, Navigation, and Control and Co-located Conferences (2003)
[3] N.B. Singer, et al. *A study of Human Teleoperation of a Space Manipulators Using Simulation* (2016)
[4] W. Stallings, *Wireless Communications and Networking* (2002)

delay discrepancies influence the operator workload. It is speculated that the operator workload is directly correlated with the amount of time/signal delay. There is some anecdotal evidence indicating that under certain conditions the performance is maintained by an operator at a certain level whereas workload is increased[2]. It is proposed to measure both performance and workload during the simulator run in order to prove or disprove this hypothesis.

One of the prime objectives of this investigation is to develop new metrics for simulator fidelity assessment based upon objective parameters. Data acquired throughout the experimental test will further bolster the aforementioned metrics. In this study, fifteen trials of time delays were recorded from 0 milliseconds to 2000 milliseconds. The objective for this research is to study and acquire data for objective benchmarks for simulator fidelity via operator control behavior analysis. A quantitative metric is to be derived to correlate user workload and ease of simulator control. These results can be utilized to provide enough data to recreate a model of the human operator through the theories of soft computing based on objective parameters as well[2]. Such a model would enable further research and studies in the controls of space manipulators between earth and outer-space and how the various time and signal delays can have an effect on the operator performance.

METHODOLOGY

In this experiment, the subjects controlled and guided a robotic inceptor in outer-space to dock on a marked target on a satellite through a computer simulation. The main controller used to undergo the simulation was the Logitech Force 3D Pro Joystick. This joystick is a 3-axis joystick allowing for roll, pitch, and yaw with various buttons and a throttle paddle. These controls allowed for the computer simulation robotic inceptor arm to move up and down, left and right, and forward and backwards are various speeds. The robot inceptor, modeled as a silver rectangular box on the bottle center of the screen was to be moved into the center of the white square on the satellite surrounded by the red box using the joystick. Once successfully placed the joystick would provide haptic feedback, the red box would turn green, and the word “latched” would appear at the top of the screen. Throughout the testing, each of the subjects would complete 15 trials with various simulation parameters changed each time, specifically the time delay

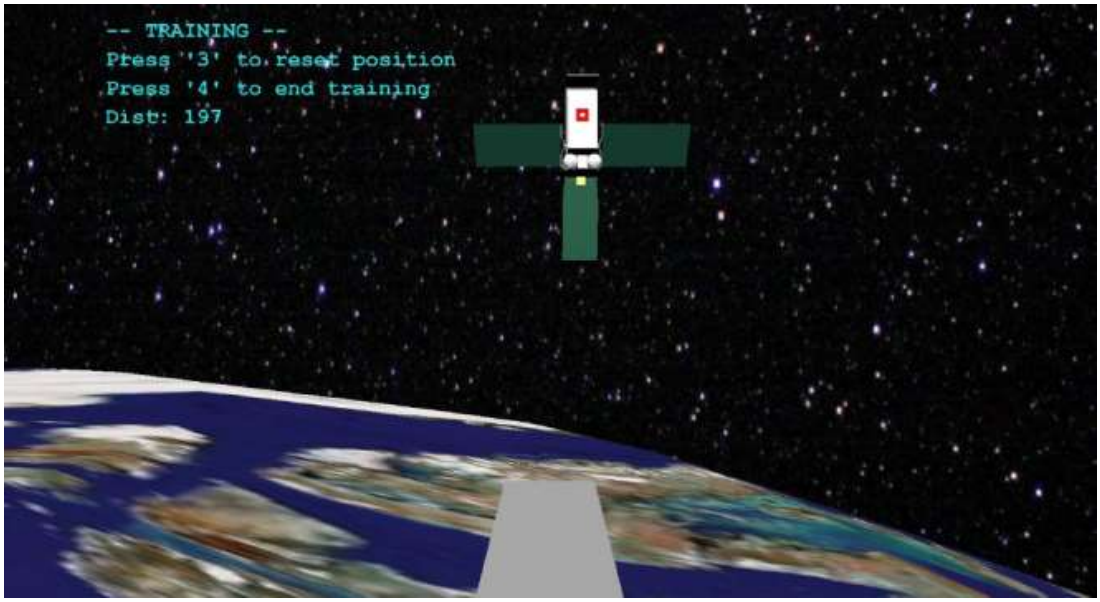

Figure 1. The Simulation Virtual Environment. Inceptor is at the bottom center.

[2] Kirill Zaychik and Frank Cardullo. *Simulator Sickness: the Problem Remains,* AIAA Modeling and Simulation technologies Conference and Exhibit, Guidance, Navigation, and Control and Co-located Conferences (2003)

ranging from 0 milliseconds to 2000 milliseconds. These experimental variables were added to intentionally cause latency between the time the operator sent a command through the joystick and when the inceptor would move on the screen. The signal delay was added by sampling joystick movements at regular periodic intervals (40 Hz) and storing into a delay buffer[3]. The simulation of differing delays was accomplished by having the program script select a joystick sample at a fixed depth in the buffer.

Collisions and latch positions with the satellite had to be inspected programmatically. Essentially, to differentiate docking and collisions, the coordinates of the marked target were checked via test statements. In the event of a collision, the inceptor arm was given a velocity negative to the collision direction. This would knock the arm away from the satellite.

Figure 1 is a screenshot of the simulation environment, created through the use of Simulink and MATLAB. Simulink was utilized to exhibit the virtual outer-space environment read from a VRML (VR Modelling) file. This consisted of 4 primary elements: the effector (robotic inceptor) arm, the satellite, the planet, and the star field. The only interactive element was the robotic inceptor, which was controlled and manipulated via the MATLAB script when running the simulation. The satellite, planet, and star field were all environmental elements, designed to adjust according to the inceptor movement to provide visual reference points which aids in ease of user interaction. These objects were all imported from the Simulink object library with different graphics mapped over them as skins. The simulation also provided a HUD (Heads Up Display) on the top left corner which included the trial/test status, instructions, as well as the distance to the target.

Initially, the MATLAB script was a very simple kinematic method of position calculation. The magnitude of joystick roll, pitch, and yaw was recorded to translate the end effector that magnitude in the virtual environment, which was then scaled down or up via the throttle for speed control. In reality for simulator fidelity, the element of dynamics had to be added. The dynamics model was added by adding an integral controller to the roll, pitch, and yaw of the joystick input. Figure 2 below indicated the block diagram noting the feedback loop which added dynamics. Figure 3 shows how this was implemented into the MATLAB script.

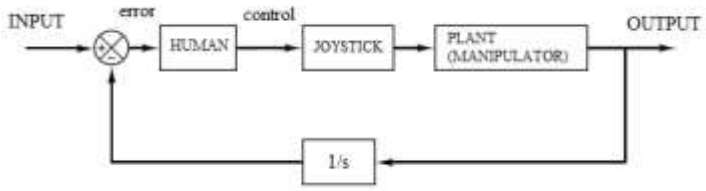

Figure 2. (Above) Dynamics modeled into the block diagram. The integral controller was added via the "1/s" feedback

```
%% SIMULATION LOOP
%-------------------------------------------
roll_d_old = 0;
roll_old = 0;
pitch_d_old = 0;
pitch_old = 0;
yaw_d_old = 0;
yaw_old = 0;
%% READ JOYSTICK
%-------------------------------------------
    % read joystick vals
    [axes, buttons, cap] = read(joy);

    roll = axes(1);
    pitch = axes(2);
    yaw = axes(3);
    throttle = -axes(4);

    roll_d = roll_d_old+0.0167*roll_old;
    pitch_d = pitch_d_old+0.0167*pitch_old;
    yaw_d = yaw_d_old+0.0167*yaw_old;

>>>BREAK IN CODE<<<

    sens_mult = .51 + (throttle * .5);

    sens_x = 1 * sens_mult;
    sens_y = 1 * sens_mult;
    sensy_z = .6 * sens_mult;

    % joystick readings - velocities
    % V = [right_left up_down front_back pitch yaw
roll];
    v_x = roll * sens_x;
    v_y = pitch * sens_y;
    v_z = front_back * sensy_z;

    roll_d_old = roll_d;
    roll_old = roll;
    pitch_d_old = pitch_d;
    pitch_old = pitch;
    yaw_d_old = yaw_d;
    yaw_old = yaw;
```

Figure 3. (Above) Dynamics modelled into MATLAB (Highlighted). NOTE: ">>>BREAK IN CODE<<<" was put into substitute several lines of code.

[3] N.B. Singer, et al. *A study of Human Teleoperation of a Space Manipulators Using Simulation* (2016)

This added difficulty to the experiment and allowed the end effector to coast when impulse is added suddenly against the original movement direction[3].

For experimentation and the testing procedure, data was collected from 20 test subjects, 19 male subjects and 1 female subject. The ages for the subjects ranged from 19 to 31 with the median age being 20 and the mean age being 21. Each of these test subjects were individually tested all 15 trials with the simulation.

The testing included 2 steps: training and recorded trials. The training period was before the data acquisition began. The subjects had the opportunity to complete training runs until reaching the learning curve to feel comfortable using the joystick to control the robotic inceptor. Each subject was encouraged to take as many training runs and as much time as necessary to feel comfortable with the joystick. Once the facilitator was assured that the subject was comfortable enough to proceed with the simulation and joystick interface controls, the subject was able to proceed to the recorded trials phase. During the recorded trials, the 15 trials with changing signal delays were run. Each of the subjects performance metrics were measured and recorded during this time for each of the trials. The following data was recorded: the time to complete each trial, the number of collisions with the satellite during each trial, and the number of restarts for each trial. The time parameters which changed each trial were the time delays in such order: 0 ms, 600 ms, 1000 ms, 1400 ms, 1000 ms, 2000 ms, 1600 ms, 400 ms, 1800 ms, 200 ms, 1200 ms, 2000 ms, 600 ms, 800 ms, and 1600 ms. The main indicator was the time it took for the subject to reach the target.

RESULTS

Data was acquired in the form of a 7 column .csv file by which were represented by the delay, the time, roll stick, pitch stick, yaw stick, throttle, and front-back (forward or reverse). This information was collected for all fifteen trials for all 20 subjects and the controller variables in roll, pitch, yaw, and throttle were plotted against time as such in figure 4.

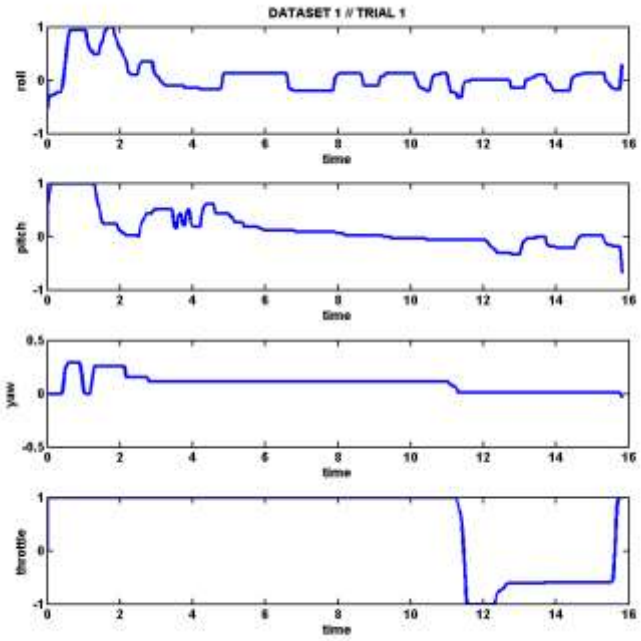

Figure 4. Dataset 1 Trial 1 Joystick Data Plots vs. Time

Once the control behavior data was plotted, some form of metric needed to be identified and to distinguish "good" control behavior and "bad" control behavior. The two general methods for identifying metrics are analytically and Human Operator in-the-loop Analysis[5]. For the analytical approach, time and frequency response analysis is required and for the Human Operator in-the-loop Analysis, the typical analyses are Root Mean Square (RMS) Error, Power Spectral Density (PSD) Analysis, and the NASA TLX[5]. This project utilized the NASA TLX (Task Load Index) to gauge which trials the operators had the most difficulty with, combined with the Analytical Approach of Time and Frequency Analysis, and most importantly,

[3] N.B. Singer, et al. *A study of Human Teleoperation of a Space Manipulators Using Simulation* (2016)
[5] Cardullo, Frank. *33rd Annual Flight and Ground Vehicle Simulation Course* Presentation. (2017)

the PSD Analysis. This study specifically utilized the Integrated PSD Analysis in that this measures and calculates the power that is in a signal by a mathematical technique in the MATLAB Signal Processing Toolbox similar to the Fast Fourier Transform[5]. From the collected pilot signal, the PSD could be obtained at different frequencies. Ideally, the peak migrations of the PSD plots are indicative of the delay. A more pronounced peak shift is the result of more operator movement, therefore meaning that the operator is working harder, thus producing more power due to there being more delay. In this study, rather than seeing distinguishable peak shifts with large delays, the peaks themselves become much more pronounced as exhibited in Figure 5. PSD Analysis were performed on the roll stick, pitch stick, and the "Combined Roll+Pitch" or Combined. The Combined Roll+Pitch was calculated as shown.

$$Combined = \sqrt{Roll^2 + Pitch^2}$$

This was applied to each of the individual time recorded time signals per trial. This was justified because it is a method to transform the rotational joystick movements to a translational straight-line movement analogous to vector addition. Since y-axis simulation translation is a function of roll and x-axis simulation translations is a function of pitch, the resultant translation can be expressed as the sum of roll and pitch.

Next, the PSD for delays were integrated with respect to frequency to generate the Integrated PSD Charts for the No Delay (0 ms), Medium Delay (1000 ms), and High Delay (2000 ms). As depicted in Figure 6, as higher delays get introduced, there is more power provided by the operators to the joystick to try an offset the errors caused by the delay.

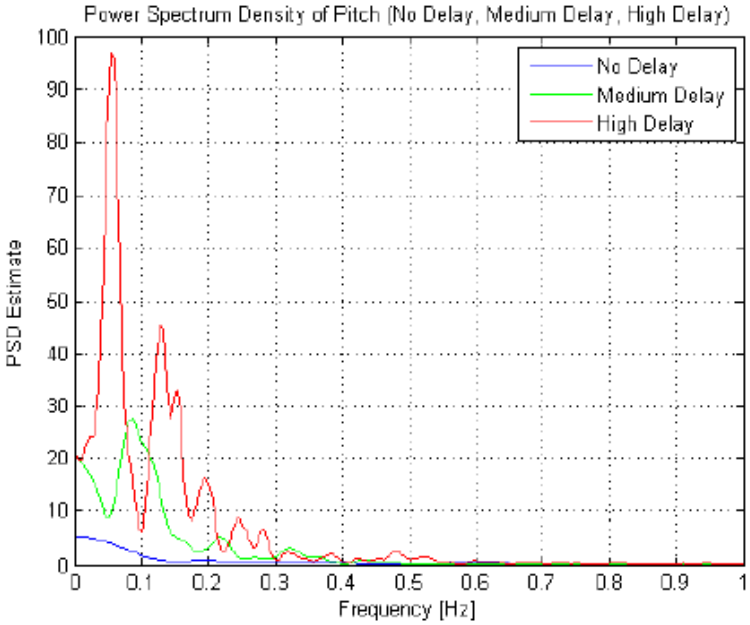

Figure 5. PSD Plot of Pitch under No Delay (0 ms), Medium Delay (1000 ms), and High Delay (2000 ms)

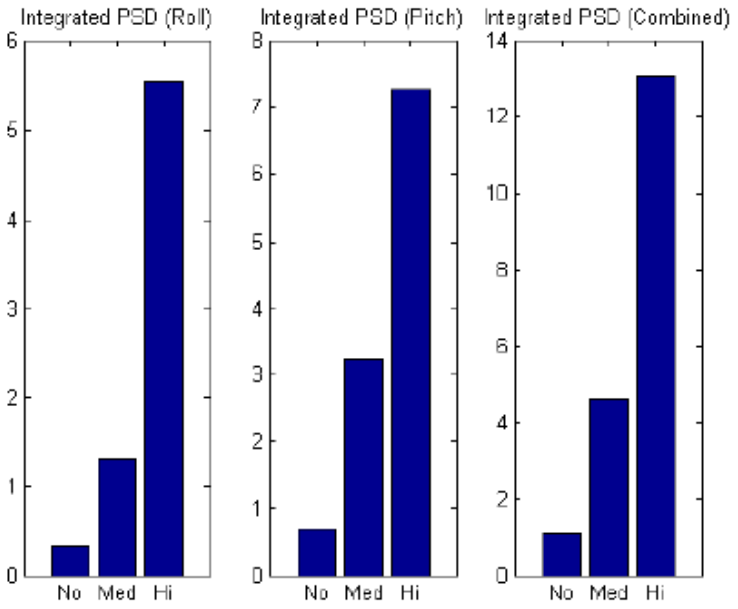

Figure 6. Integrated PSD Plots of Roll, Pitch, and Combined Roll+Pitch

[5] Cardullo, Frank. *33rd Annual Flight and Ground Vehicle Simulation Course* Presentation. (2017)

From the resulting Integrated PSD charts, it seemed to be the general trend that there is more power applied because of added delays. To further this claim, the PSD data was put together for all 20 Datasets which resulted in the development of Figure 7.

As evident in the chart, as a larger signal delay is introduced within the simulation, not only was there more power, but also more deviation within the collected data. This is indicative of the original hypothesis that with the introduction subsequently increasing time delays, there is more difficulty in acquiring a prediction the pilot control behavior. Essentially, under very high signal delays, there is both more unnecessary operator calibration movements and more data deviation as well as pilot behavior unpredictability.

DISCUSSION

Due to the fact that much of this study was pioneered from *A Study of Human Teleoperation of a Space Manipulators Using Simulation* (2016) by Noah B. Singer, Nathan Sprenkle, and Dr. Frank Cardullo at the Man-Machine Systems Lab at Binghamton University, it is critical to review some of the metrics covered by their team such as trial time as well as collision count.

The data was collected from all of the subjects into one aggregate dataset, which was then analyzed using Microsoft Excel. As mentioned earlier, two additional metrics on top of the PSD and Integrated PSD Analysis are the measured time trials and the collision count. The time trials are simply just the time the subject took to traverse the simulator world with the end effector from the initial starting position to the satellite target.

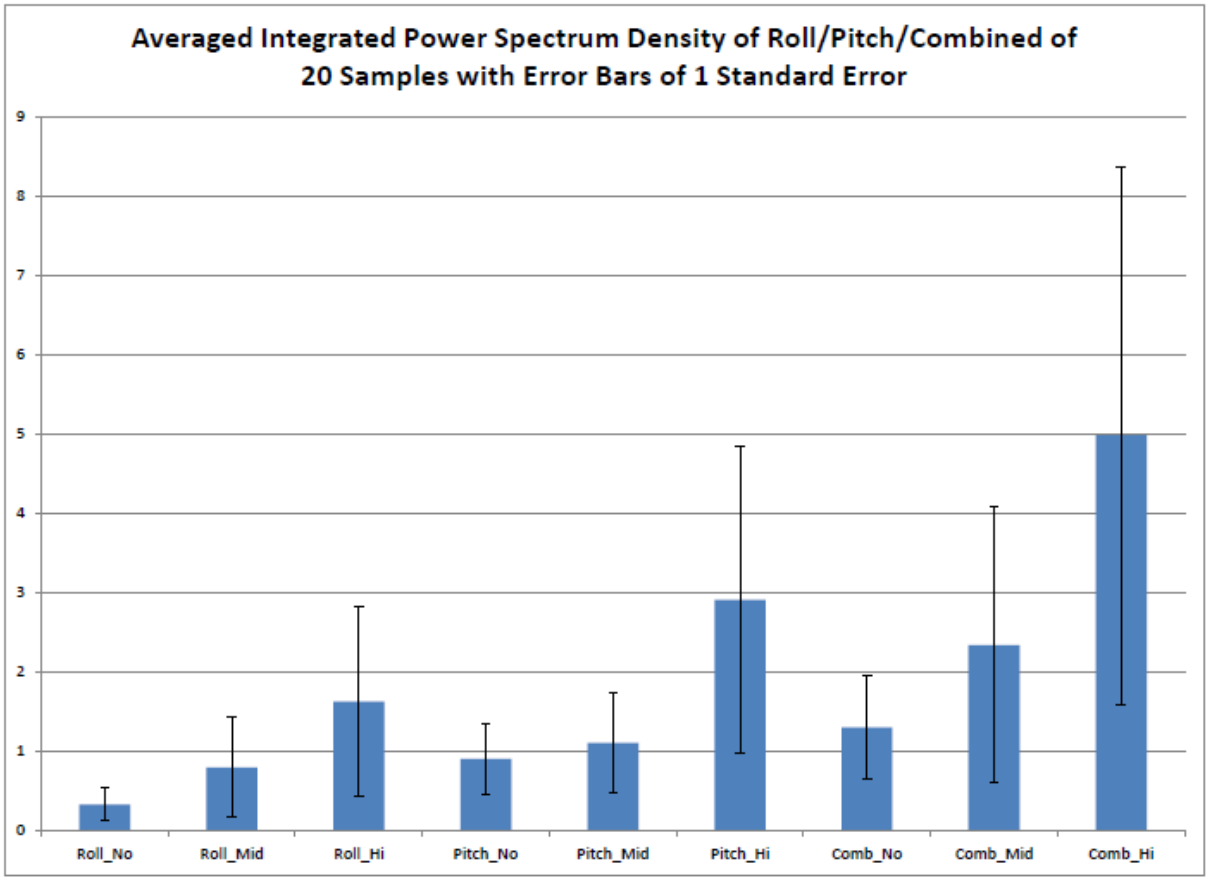

Figure 7. Integrated Power Spectral Density Charts were averaged based on stick control and delay for all 20 subjects. Error bars of one standard error were implemented.

The data for that is presented graphically below in Figure 8.

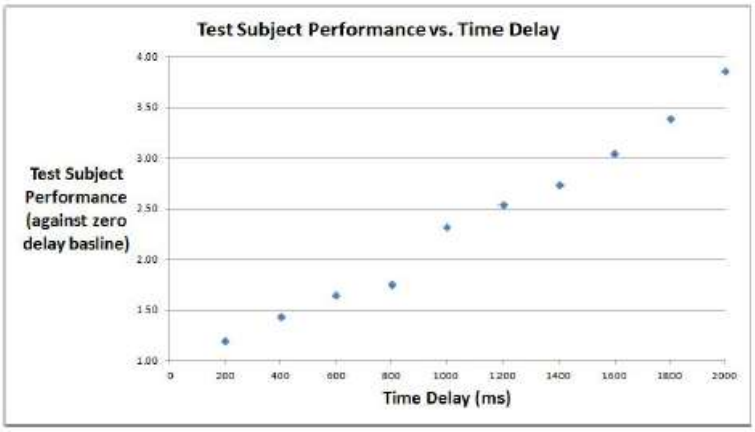

Figure 8. Time trials with introduced delays

The x-axis represents the time delay in milliseconds and the y-axis represents the average performance of the test subjects at that time delay. The data was normalized using the baseline performance time of zero delay so that at each time delay, the value of the Test Subject Performance is how many time longer it took the subjects to complete the trial at that delay versus no time delay.

There is a clear trend demonstrated by the data: as time delay increases, the time required to complete the trial increases. At a delay of 1 second, it took the subjects 2.32 times longer to complete the trial than at zero delay. At a delay of 2 seconds (the maximum delay used in this study), subjects took 3.86 times longer to complete the trial than at zero delay.

It appears that the relationship between time delay and subject performance is linear with an R-squared value of 0.985, however, further studies with larger ranges of time delays and more data points are required to definitively determine the mathematical relationship between time delay and subject performance. It is very clear from the data that time delay, and thus communication delay, has a significant effect on an operator's ability to control a robot.

Another set of data that were collected were the number of collisions with the satellite during each trial. A collision in this simulation is defined as when the controlled end effector makes contact with any part of the satellite which is not the target. When a collision occurs, the joystick produces a haptic feedback informing the user of a collision. This data is important because in a real life scenario, every collision between 2 objects in space is likely to cause damage and be expensive to repair. The collision data was averaged from all of the test subjects for each trial and is reported below in Figure 9.

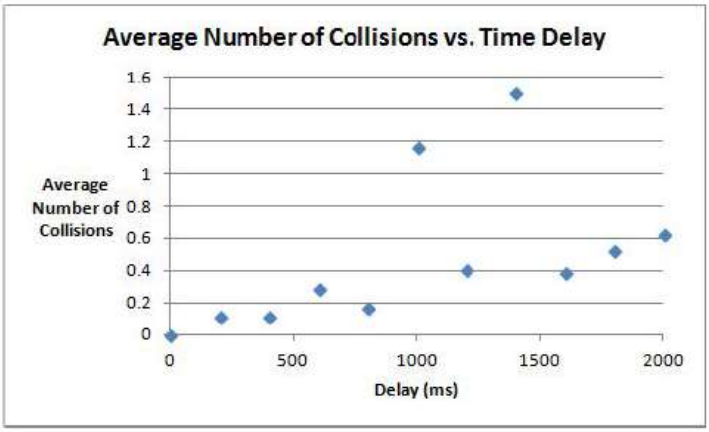

Figure 9. Average number of collisions per time delay

It is difficult to draw conclusions from the collision data. Although there appears to be a positive upward trend in the number of collisions as time delay increases, the presence of outliers makes it difficult to draw any significant conclusions. More testing, over a larger range of time delays, coupled with a larger pool of test subjects, may show that there is indeed a relationship therefore, more testing is needed in the future.

Investigations were also made into whether the amount of training runs the subject used to familiarize themselves with the simulation and joystick affected the statistical spread of their data. There appears to be no correlation between the standard deviation of the data and the amount of training runs taken by the subject based on the data collected from this experiment. However, the scope of this study is limited so further investigations into the effect of training on the statistical spread of the data are recommended.

CONCLUSION

Teleoperation and simulation is a huge topic as well as a sector in applied science and engineering in that it allows for the human remote controlling of machines and devices into places inaccessible by humans. This paves the path for a world of endless opportunities for research, learning, and development.

Communications delay can negatively affect the abilities of a human operator who is controlling a robot in space from the surface of Earth. To study the effects of communication delay, a simulation was developed in MATLAB and Simulink. The simulation allows subjects to be tested for the effects of communication delay by simulating the control of a robotic inceptor in space which is intended to latch with a satellite. The test subject controls the inceptor with a joystick and completes 15 trials with different communications delays.

Overall, 20 test subjects were tested using the simulator. The data from the simulator shows a clear trend that with an introductions with a larger signal delay in the simulation, there was more power, meaning that there was more operator to joystick movement and more deviation within the collected data. This bolsters the claim of the original hypothesis that with subsequently increasing signal delays, there is more difficulty in acquiring an operator control-behavior prediction. Essentially, with high signal delays, there is both more unnecessary pilot calibration movements to correct the path error and more data deviation as well as a more chaotic pilot control behavior.

FUTURE PLANS

The simulation is improved from the initial stage, there is still much that could be done to improve the experiment for future years. These changes include improving the UI, recording additional data, improving satellite bounce, and refining delay techniques.

Additional Data Acquisition: One issue that emerged after the human subject trials was the lack of recorded real time position tracking. During the simulation, the position of the end effector was tracked however, this was neither recorded nor stored within the .csv file. This posed to be a problem in trying to figure out the distance-to-target vs. time comparisons within different signal delays. It is crucial that the distance-to-target is recorded vs. time for the future.

Improved UI: Currently the simulation and UI are functional but simplistic. To make the test appear more realistic, better models for the Earth, satellite, and starfield could be used.

Improved Satellite Bounce: The simulation currently only handles collisions between the main body of the satellite and the effector. This was improved from the previous system which included more complete bounce off logic but poor bounce off effect, simply resetting the subject's position. Ideally the simulation would be able to detect collision with the peripheral panels and components attached to the satellite body. Additionally, the current rebound algorithm is simple: apply a negative velocity in the axis of collision for a set number of cycles. A more realistic simulation might more accurately model rebound off the satellite body.

Delay Accuracy: Additionally, the way delays are implemented is rather simplistic. As mentioned earlier, a fixed delay is built into the program loop and control data is put into a buffer to delay controls data manifesting in the simulation. A more exact simulation might dynamically add delay to ensure each simulation loop executed in as close to uniform time as practical in a MATLAB script running on a non-realtime system.

Additional Analysis: There are some analyses on the data that due to time constraints could not be analyzed. One is to analyze to see if there is any improvement throughout the trials as the experiment progresses. Since a rough mathematical model could be developed to fit the data, the user's performance could be predicted at each delay value. Then, by ordering in the order that the user was tested in rather than ordering by increasing delay and comparing actual performance to modeled performance, it

could be seen how much the user improves over the course of the experiment and remove this possible learning bias from the data.

Accuracy Data: Though we already collect many different data points, there is even more data we considered collecting but did not. First was accuracy of docking. The area that a subject can successfully dock in is about twice the area of the effector head. Anywhere inside this area is considered a successful dock but the position from ideally centered docking is not recorded. Other data that might prove insightful could be deviation off of the ideal path to target (averaged or maximum) to measure overcorrecting movement by the test subject.

Spinoff Experiment: As mentioned above, we believe a more ideal experiment would test overcorrection and accuracy in addition to (or in lieu of) time to complete task. Something that was observed was that in addition to the task taking longer with the delay, subjects began to make large over corrections at higher delays. A possible experiment may include markers along an ideal path that the subject must follow to reach the target, measuring deviation from the path. This is explored briefly by C.M Korte, et. al. in *Effect of Time Delay on Teleoperation Accuracy and Efficiency* but could be further expanded on. A variant of the experiment might simply move the user forward at a constant speed and measure accuracy along a marked path and overcorrection for greater insight on accuracy with delay.

LITERATURE CITED

[1] N. Tesla, *Patent US613809 – Method of and Apparatus for Controlling Mechanism of Moving Vessels or Vehicles* (1898)

[2] Kirill Zaychik and Frank Cardullo. *Simulator Sickness: The Problem Remains*, AIAA Modeling and Simulation Technologies Conference and Exhibit, Guidance, Navigation, and Control and Co-located Conferences (2003)

[3] N.B. Singer, et al. *A Study of Human Teleoperation of a Space Manipulators Using Simulation* (2016)

[4] W. Stallings, *Wireless Communications and Networking* (2002)

[5] Cardullo, Frank. *33rd Annual Flight and Ground Vehicle Simulation Course* Presentation. (2017)

APPENDIX I: USER GUIDE

To run the simulation, a Windows computer is needed with a joystick. The joystick must have at least 2 axis of control, a throttle control, and 4 buttons. In our procedure, 2 monitors were used. One with the simulation for the test subject, and one, facing another direction for the facilitator to fill out information and observe experiment data. First open MATLAB with the most recent file (simulation_joystick_0510.m as of the righting of this guide). Run the script.

The following prompt will show "Collect Data? (1/0)". For debugging and writing code without saving data to the excel sheet, enter 0, otherwise enter 1. If collecting data with a test subject enter the following data when scripted, Age numeric, Sex as a single letter ("M" or "F"), and Subject Number as a number. Subject number determines which row in the data file to save the subject's data to (currently Simulation_Data_Spring_2016.xlsx). **Note:** This spreadsheet *must* be closed before starting a trial or the program will throw an error.

Simulink should open with the environment described by the most recent WRL file (currently satellite3.wrl).

When testing subjects, first read them the first half of the test protocol (Test Protocol 2016.docx) explaining training and the simulator environment. As outlined in the test protocol, the goal is to maneuver the end effector (grey box) into the center of the red box on the satellite. Controls are as follows:

- Joystick axis 1 (+/): up/down movement
- Joystick axis 2 (+/): left/right movement
- Joystick buttons 1 & 2 (trigger and left side grey button): forward/back movement
- Joystick throttle (grey paddle at base of joystick): movement speed
- Joystick button 3 : start trial or reset position
- Joystick button 4 : begin testing

Have the test subject experiment with the controls and the simulation. We suggest a minimum of 5 successful dockings. When the subject feels they have a solid understanding of the simulation environment, read them the second half of the test procedure. This outlines that delay will be introduced in various trials and the procedure for restarting trials; have them begin the trials by starting a training run, then pressing 4 on the joystick, as prompted on screen. The text "WAITING: TRIAL 1" should be displayed on screen, before experimentation begins.

Have the subject press 3 to begin a trial. Each trial with show the trial number and delay amount on the experimenter's MATLAB console. Observe their progress throughout the 15 trials. After completing the final trial, press 3 on the joystick so that the text "TRIALS COMPLETE" appears on screen. The facilitator's MATLAB console will read "Press enter to exit". After hitting enter, all data should have been recorded and the simulation may be safely exited.

APPENDIX II: MATLAB SIMULATION SOURCE CODE

```
%% JOYSTICK/WORSKPACE INIT
%--------------------------------------------------------------------------

% clearing the workspace and screen
clear all
clc

% setup joystick
id = 1;
joy = vrjoystick(id, 'forcefeedback');

% return joystick capabilities
c = caps(joy);

%% DATA COLLECTION
%--------------------------------------------------------------------------
testing = input('Collect data? (1/0)');
age = 0;
gender = '';
subj_num = 50;

% collect subject details
if testing
    age = input('enter age: ');
    gender = input('enter gender(m/f): ','s');
    subj_num = input('enter your subject number: ');
end

%% FILE I/O
%--------------------------------------------------------------------------

% subject numbering starts at row 5, so offset by 4
row_num = 4 + subj_num;

% open excel file to write data (write to 3 sheets)
file = ('Simulation_DATA_Fall_2016.xlsx');
if testing
    xlswrite(file,subj_num,'times',['A' num2str(row_num)]);
    xlswrite(file,subj_num,'hit count',['A' num2str(row_num)]);
    xlswrite(file,subj_num,'trial attempts',['A' num2str(row_num)]);

    xlswrite(file,age,'times',['C' num2str(row_num)]);
    xlswrite(file,age,'hit count',['C' num2str(row_num)]);
    xlswrite(file,age,'trial attempts',['C' num2str(row_num)]);

    xlswrite(file,gender,'times',['B' num2str(row_num)]);
    xlswrite(file,gender,'hit count',['B' num2str(row_num)]);
    xlswrite(file,gender,'trial attempts',['B' num2str(row_num)]);
end

% delay in ms
```

```
delay_ms = xlsread(file,1,'D2:R2');

% read number of delay slots as array, one column is one test case
delay_number = delay_ms./50;
delay_number(16) = 0;

trial_number = 1;
training_number = 1;

csv_file= strcat('joystick_data_', num2str(subj_num), '.csv');

%% LOAD VIRTUAL WORLD
%--------------------------------------------------------------------------

% open virtual world from satellite.wrl - previosly test.wrl
w = vrworld('satellite3.wrl', 'new');
open(w);

% create the vrfigure showing the virtual scene
fig = vrfigure(w);
ref = vrnode(w, 'reference');

% go to a viewpoint suitable for user navigation
set(fig, 'Viewpoint', 'Effector Camera');

% get the manipulated effector node
effector = vrnode(w, 'Effector');

% read plane initial translation and rotation
initialTranslation = [30 0 0];
originalTranslation = effector.translation + initialTranslation;
originalRotation = effector.rotation;

% LATCH POSITION
latchPosition = [-0.75 75 189];
err_x = 0;
err_y = 0;
err_z = 0;
dist_to_target = 0;

% set the HUD display text
offset = vrnode(w, 'HUDOffset');
offset.translation = offset.translation + [-0.15 1.9 0];

hudtext = vrnode(w, 'HUDText1');
hudstr = sprintf(strcat(...
    '-- TRAINING --\n', ...
    'Press ''3'' to reset position\n', ...
    'Press ''4'' to end training\n' ...
));
hudstr = textscan(hudstr, '%s', 'delimiter', '\n');
hudtext.string = hudstr{1};
```

```
%TEST COLOR CHANGE
bottom_box = vrnode(w, 'bottom_box');
bottom_box = getfield(bottom_box, 'children');
top_box = vrnode(w, 'top_box');
top_box = getfield(top_box, 'children');
left_box = vrnode(w, 'left_box');
left_box = getfield(left_box, 'children');
right_box = vrnode(w, 'right_box');
right_box = getfield(right_box, 'children');

%% VARIABLE INIT
%--------------------------------------------------------------------------

% rows for generating delay
buffer_len = 80;

% whether the user has given up or proceeded to next rial
skip_trial = 0;
begin_trial = 0;

% velocity and buffer values
v = zeros(1,7);
buffer = zeros(buffer_len+1,7);
bounce_max = 20;
bounce = 0;

% joystick bounce
bounce_dir = 0;
joy_axis = 0;
joy_force = 0;

% trial vars
sim_state = 'wait_training';
trial_attempts = 1;
hit_count = 0;
block_signal = 1;

times = zeros(1,15);
reached_target = 0;
timer_trial = tic;

% latch vars
latch_max = 30;
latch_loop = latch_max;
latch_velocities = [0 0 0];

joy_data = [0 0 0 0 0 0 0];
n = 1;

% dlmwrite(csv_file,joy_data,'delimiter',',','-append');
joy_data_abs = 0;
joy_data_interval = 0;
tJoyData = tic;
% csvwrit
% joy_data(n, 1) = 'delay';
```

```
% joy_data(n, 2) = 'time';
% joy_data(n, 3) = 'x';
% joy_data(n, 4) = 'y';
% joy_data(n, 5) = 'yaw';
% joy_data(n, 6) = 'throttle';
% joy_data(n, 7) = 'z';

%% SIMULATION LOOP
%--------------------------------------------------------------------------

roll_d_old = 0;
roll_old = 0;
pitch_d_old = 0;
pitch_old = 0;
yaw_d_old = 0;
yaw_old = 0;

while isvalid(fig)
    tSimLoop = tic; % per kerchunk iteration time (for delay)
    joy_data_interval = toc(tJoyData);
    tJoyData = tic;
    % tAbsLoop = tic; % testing


%% READ JOYSTICK
%--------------------------------------------------------------------------

    % read joystick vals
    [axes, buttons, cap] = read(joy);

    roll = axes(1);
    pitch = axes(2);
    yaw = axes(3);
    throttle = -axes(4);

    roll_d = roll_d_old+0.0167*roll_old;
    pitch_d = pitch_d_old+0.0167*pitch_old;
    yaw_d = yaw_d_old+0.0167*yaw_old;

    cap_up_down = 0;
    cap_right_left = 0;
    front_back = 0;

    if(cap == 0), cap_up_down = 1;
    elseif (cap == 180), cap_up_down = -1;
    elseif (cap == 90), cap_right_left = 1;
    elseif (cap == 270), cap_right_left = -1;
    end

    if (buttons(1) == 1 && buttons(2) == 1), front_back = 0;
    elseif (buttons(1) == 1), front_back = -1;
    elseif (buttons(2) == 1), front_back = 1;
    end
```

```
    sens_mult = .51 + (throttle * .5);

    sens_x = 1 * sens_mult;
    sens_y = 1 * sens_mult;
    sensy_z = .6 * sens_mult;

    % joystick readings - velocities
    % V = [right_left up_down front_back pitch yaw roll];
    v_x = roll * sens_x;
    v_y = pitch * sens_y;
    v_z = front_back * sensy_z;

    roll_d_old = roll_d;
    roll_old = roll;
    pitch_d_old = pitch_d;
    pitch_old = pitch;
    yaw_d_old = yaw_d;
    yaw_old = yaw;

    % velocity values
    v = [v_x v_y v_z 0 0 0 0];

    % timestamp speed samples
    s_time = toc(timer_trial);

%% DELAY
%-------------------------------------------------------------------------
% mouse values are sampled at 40Hz and loaded into a buffer at position 1
% delay is chosen by sampling from a later row in the buffer
% specifically delay_number(trial_number)+1

    % shift mouse samples to next index in buffer
    if delay_number(trial_number) ~= 0
        %fprintf('delay');
        for j = buffer_len:-1:1
            buffer(j+1,:) = buffer(j,:);
        end
    end

    % load current samples into buffer at row 1
    buffer(1,:) = v;

    % grab later input value from the buffer to achieve delay
    v = buffer(delay_number(trial_number)+1,:);

    % add .025s delay to each iteration (e.g. trial_number of 4 selects from
4th row providing .1s delay)
    % pause(0.025);

%% SIMULATION FLOW
%-------------------------------------------------------------------------
    switch sim_state
        case 'training'
            block_signal = 0;
            %n = n + 1;
```

```
            % HUD
            hudstr = sprintf('-- TRAINING --\nPress ''3'' to reset
position\nPress ''4'' to end training\nDist: %d', int64(dist_to_target));
            hudstr = textscan(hudstr, '%s', 'delimiter', '\n');
            hudtext.string = hudstr{1};

            % keep user on trial with no delay
            if (reached_target)
                %save timer to beginning of array, so I don't have to make
                %a new var
                times(1) = toc(timer_trial);

                % get distance to latch center
                latch_velocities = latchPosition - effector.translation;
                latch_velocities = latch_velocities ./ latch_max;
                latch_loop = latch_max;

                sim_state = 'training_latch';
                fprintf('Passed training exercise %d\n', trial_attempts);
                trial_attempts = trial_attempts + 1;
                training_number = training_number + 1;
            end

            % press 3 to reset
            if (button(joy, 3) == 1)
                effector.translation = originalTranslation;
                effector.rotation = originalRotation;
            end

            % press 4 to begin testing
            if (button(joy, 4) == 1)
                trial_number = 1;
                sim_state = 'wait_trial';
                disp('Training Ended');
            end

        case 'training_latch'
            block_signal = 1;
            hudstr = sprintf('>> LATCHED <<\nPress ''3'' to start new
trial\n');
            hudstr = textscan(hudstr, '%s', 'delimiter', '\n');
            hudtext.string = hudstr{1};

            % latch to center
            % effector.translation = latchPosition;
            if(latch_loop > 0)
                effector.translation = effector.translation +
latch_velocities;
                latch_loop = latch_loop - 1;
            end

            bottom_box.appearance.material.diffuseColor = [0 1 0];
            top_box.appearance.material.diffuseColor = [0 1 0];
            left_box.appearance.material.diffuseColor = [0 1 0];
```

```
            right_box.appearance.material.diffuseColor = [0 1 0];

            if (button(joy, 3) == 1)
                % record testing data
                if testing
                    % calculate column
                    if training_number <= 22
                        excol1 = char(0);
                        excol2 = char(training_number+67);
                    else
                        decm = floor(training_number/22);
                        excol1 = char(decm+64);
                        excol2 = char(training_number-(decm*22)+64);
                    end
                    excol = [excol1 excol2 num2str(row_num)];

                    xlswrite(file, times(1), 'training', excol);

                    %CSV write
                    % fprintf('offset - %d\tn - %d\', csv_offset, n);
                    % dlmwrite(csv_file,joy_data,'delimiter',',','-append');
                end

                training_number = training_number + 1;

                while(button(joy, 3)), end
                sim_state = 'wait_training';
                continue;
            end

        case 'wait_training'
            block_signal = 1;

            %CSV
            % n = 1;
            % joy_data = [0 0 0 0 0 0 0];

            effector.translation = originalTranslation;
            effector.rotation = originalRotation;

            bottom_box.appearance.material.diffuseColor = [1 0 0];
            top_box.appearance.material.diffuseColor = [1 0 0];
            left_box.appearance.material.diffuseColor = [1 0 0];
            right_box.appearance.material.diffuseColor = [1 0 0];

            hudstr = sprintf('-- WAITING --\nPress ''3'' to start
training\n');
            hudstr = textscan(hudstr, '%s', 'delimiter', '\n');
            hudtext.string = hudstr{1};

            % press 3 to start trial
            if (button(joy, 3) == 1)
                while(button(joy, 3)), end
                sim_state = 'training';
                timer_trial = tic;
```

```
            end

        case 'wait_trial'
            block_signal = 1;

            if trial_number > 15
                sim_state = 'testing_complete';
                continue
            end

            % hold position at start
            effector.translation = originalTranslation;
            effector.rotation = originalRotation;

            hit_count = 0;
            trial_attempts = 1;

            % reset box color
            bottom_box.appearance.material.diffuseColor = [1 0 0];
            top_box.appearance.material.diffuseColor = [1 0 0];
            left_box.appearance.material.diffuseColor = [1 0 0];
            right_box.appearance.material.diffuseColor = [1 0 0];

            hudstr = sprintf('-- WAITING - TRIAL %d --\nPress ''3'' to start
trial\n', trial_number);
            hudstr = textscan(hudstr, '%s', 'delimiter', '\n');
            hudtext.string = hudstr{1};

            % press 3 to start trial
            if (button(joy, 3) == 1)
                while(button(joy, 3)), end
                sim_state = 'start_trial';
                delay_to_use = delay_ms(trial_number);
                fprintf('Beginning trial %d with delay of %dms\n',
trial_number, delay_to_use);
                timer_trial = tic;
            end

        case 'start_trial'
            block_signal = 0;

            %CSV
            n = 1;
            joy_data_abs = 0;
            joy_data = [0 0 0 0 0 0 0];

            % calculate excel rows
            if trial_number <= 22
                excol1 = char(0);
                excol2 = char(trial_number+67);
            else
                decm = floor(trial_number/22);
                excol1 = char(decm+64);
                excol2 = char(trial_number-(decm*22)+64);
            end
```

```
            excol = [excol1 excol2 num2str(row_num)];
            %reset position and zero inputs
            effector.translation = originalTranslation;
            effector.rotation = originalRotation;
            v = zeros(1, 7);
            buffer(:,:)=0;

            hudstr = sprintf('-- TRIAL %d --\nPress ''3'' to restart
trial\n', trial_number);
            hudstr = textscan(hudstr, '%s', 'delimiter', '\n');
            hudtext.string = hudstr{1};

            % begin execution loop
            sim_state = 'execute_trial';
            continue;

        case 'execute_trial'
            block_signal = 0;
            n = n + 1;
            joy_data_abs = joy_data_abs + joy_data_interval;
            % wait for user to reach target

            if(reached_target)
                % get distance to latch center
                latch_velocities = latchPosition - effector.translation;
                latch_velocities = latch_velocities ./ latch_max;
                latch_loop = latch_max;

                sim_state = 'pass_trial';
                continue;
            end

            % HUD
            hudstr = sprintf('-- TRIAL %d --\nPress ''3'' to restart
trial\nDist: %d', trial_number, int64(dist_to_target));
            hudstr = textscan(hudstr, '%s', 'delimiter', '\n');
            hudtext.string = hudstr{1};

            % press 3 to restart trial
            if (button(joy, 3) == 1)
                while(button(joy, 3)), end
                sim_state = 'start_trial';
                fprintf('Retry trial %d\n', trial_number);

                % restart trial and timer
                trial_attempts = trial_attempts + 1;
                timer_trial = tic;
            end

        case 'pass_trial'
            block_signal = 1;
            % stop timer
            times(trial_number) = toc(timer_trial);
```

```
            % write success
            fprintf('Passed trial %d after %f seconds\n', trial_number,
times(trial_number));

            % move to next trial

            sim_state = 'latch';
            continue;

        case 'latch'
            block_signal = 1;
            hudstr = sprintf('>> LATCHED <<\nPress ''3'' to start new
trial\n');
            hudstr = textscan(hudstr, '%s', 'delimiter', '\n');
            hudtext.string = hudstr{1};

            % latch to center
            % effector.translation = latchPosition;
            if(latch_loop > 0)
                effector.translation = effector.translation +
latch_velocities;
                latch_loop = latch_loop - 1;
            end

            bottom_box.appearance.material.diffuseColor = [0 1 0];
            top_box.appearance.material.diffuseColor = [0 1 0];
            left_box.appearance.material.diffuseColor = [0 1 0];
            right_box.appearance.material.diffuseColor = [0 1 0];

            if (button(joy, 3) == 1)
                % mark time to completion in Excel file
                if testing
                    xlswrite(file, times(trial_number), 'times', excol);
                    xlswrite(file, hit_count, 'hit count', excol);
                    xlswrite(file, trial_attempts, 'trial attempts', excol);
                    % xlswrite(file, 1, 4, excol);

                    % CSV
                    dlmwrite(csv_file,joy_data,'delimiter',',','-append');
                end

                trial_number = trial_number + 1;

                while(button(joy, 3)), end
                    if trial_number < 16
                        sim_state = 'wait_trial';
                    else
                        sim_state = 'testing_complete';
                    end
                continue;
            end

        case 'testing_complete'
            fprintf('Testing complete!\n');
```

```
            hudstr = sprintf('TESTING COMPLETE!\n');
            hudstr = textscan(hudstr, '%s', 'delimiter', '\n');
            hudtext.string = hudstr{1};
            input('press enter to exit');
            sim_state = 'end';

        case 'end'
            close(fig);

        otherwise
            disp('Unknown state %s!\n', sim_state);
    end

    % add .025s delay to each iteration (e.g. trial_number of 4 selects from
4th row providing .1s delay)
    tSimLoopElapsed = toc(tSimLoop);
    if (tSimLoopElapsed < 0.025)
        pause (0.025 - tSimLoopElapsed);
    else
        % don't need to add delay

        %fprintf('loop was %f seconds\n', tSimLoopElapsed);
    end
    % pause(0.025);

%% COLLISION CHECKING?
%--------------------------------------------------------------------------
% check if the effector has reached the pink box

% TL  0.35   76.12  187.78
% BR  -1.80   73.77  187.78

    pos = effector.translation;
    % pos = [left/right up/down front/back]
    if ((pos(1) >= -1.80) && (pos(1) <= 0.35)) && ((pos(2) >= 73.77) &&
(pos(2) <= 76.12)) && ((pos(3) >= 187.9) && (pos(3) <= 191.7))
        reached_target = 1;
        force(joy, 2, 1);

    else
        % hit satellite box
        if ((pos(1) >= -5.36) && (pos(1) <= 3.67)) && ((pos(2) >= 65.32) &&
(pos(2) <= 82.2)) && ((pos(3) >= 187.9) && (pos(3) <= 197.9))
            % front collision
            if (pos(3) >= 187.9) && (pos(3) <= 188.9)
                force(joy, 2, -1);
                fprintf('front collision\n');
                bounce_dir = 3;

            % back collision
            elseif (pos(3) >= 191.7) && (pos(3) <= 190.7)
                force(joy, 2, 1);
                fprintf('back collision\n');
                bounce_dir = 3;
```

```
            % left collision
            elseif (pos(1) >= 2.67) && (pos(1) <= 3.67)
                force(joy, 1, 1);
                fprintf('left collision\n');
                bounce_dir = 1;

            % right collision
            elseif (pos(1) >= -5.36) && (pos(1) <= -4.36)
                force(joy, 1, -1);
                fprintf('right collision\n');
                bounce_dir = 1;

            % top collision
            elseif (pos(2) >= 81.2) && (pos(1) <= 82.2)
                force(joy, 2, -1);
                fprintf('top collision\n');
                bounce_dir = 2;

            % bottom collision
            elseif (pos(2) >= 65.32) && (pos(1) <= 66.32)
                force(joy, 2, 1);
                fprintf('bottom collision\n');
                bounce_dir = 2;

            % really bad driving
            else

            end

            %force(joy, joy_axis, joy_force);
            hit_count = hit_count + 1;
            %fprintf('Collisions: %d\n', hit_count);
            bounce = bounce_max;

            % enforce minimum bounce so we don't stay in satellite
            if (v(bounce_dir) < .01) && (v(bounce_dir) >= 0)
                v_bounce = -.01;
            elseif (v(bounce_dir) > -.01) && (v(bounce_dir) <= 0)
                v_bounce = .01;
            else
                v_bounce = -v(bounce_dir);
            end

        else
            force(joy, 2, 0);
        end

        reached_target = 0;
    end

 %% DEBUGGING
%--------------------------------------------------------------------------

    % Position
    if (button(joy, 9))
```

```
        disp(pos);
    end

    % Velocity
    if (button(joy, 10))
        disp(v);
    end

    %% DISTANCE CALC
%--------------------------------------------------------------------------
    err_x = latchPosition(1) - effector.translation(1);
    err_y = latchPosition(2) - effector.translation(2);
    err_z = latchPosition(3) - effector.translation(3);

    dist_to_target = sqrt(err_x ^ 2 + err_y ^ 2 + err_z ^ 2);

%% CALCULATE NEW POSITION
%--------------------------------------------------------------------------

    if bounce > 0
        bounce = bounce - 1;

        v(bounce_dir) = v_bounce; % .*((bounce_max - bounce)./bounce_max);
    end

    if (~block_signal) %~(strcmp(sim_state, 'latch') == 1 ||
strcmp(sim_state, 'training_latch') == 1)
        % set new positions by adding the velocity vectors
        effector.translation = effector.translation + [-1 1 -1].*v(1:3);
        effector.rotation = [-1 1 -1 1].*v(4:7);
    end

    % tSimLoopElapsed = toc(tSimLoop);
    % disp(toc(tAbsLoop)); % time per loop (with controlled delay)
    % disp(tSimLoopElapsed);

    % redraw the virtual scene
    vrdrawnow;

    %% WRITING JOYSTICK TO FILE
%--------------------------------------------------------------------------
    joy_data(n, 1) = delay_number(trial_number) * 40;
    joy_data(n, 2) = joy_data_abs; % n * .025;
    joy_data(n, 3) = roll;
    joy_data(n, 4) = pitch;
    joy_data(n, 5) = yaw;
    joy_data(n, 6) = throttle;
    joy_data(n, 7) = front_back;

% end simulation loop
end

%% CLEAR WORKSPACE
%--------------------------------------------------------------------------
```

```
% close the vrfigure
% close(fig);

% close the vrworld
close(w);

% clear all used variables
% clear all;
disp('The simulation has ended');
```

APPENDIX III: MATLAB POWER SPECTRAL DENSITY SOURCE CODE

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function calculates the single-sided Power Specral Density (PSD) of
% a real signal and the frequency of the PSD
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% the code is partially adopted from Porat, B., "A Course in Digital Signal
% Processing", Wiley, ISBN 0471149616, 1997
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Syntax: [P,F] = smPSD(Input,Time)
% Inputs:
% 1.     Input == Input signal
% 2.     Time  == Time vector OR a sampling frequency in Hz
% Outputs:
% 1.     P == PSD estimate
% 2.     F == Frequency vector (Hz)


% PROGRAMMER: Kirill Zaychik
%
% $ STATUS:  beta testing     $
% $ Revision: 1.0             $
% $ Date: 2011/07/06 13:36:41 $


function [P, f] = smPSD(x,FS)

len     = length(x);
if length(FS) > 1
    Fs = 1/(FS(2) - FS(1));
    time = FS;
else
    Fs = FS;
    time    = 0:1/Fs:(len - 1)/Fs;
end

% zeropad the signal to the length of FFT
xz=[x' zeros(1, 2^nextpow2(length(x))-len)];

if ~rem(len,2), % determine the length of the window, it must be odd
    lwin=len-1;
else
    lwin=len;
end

itime   = time(1:lwin);
w       = .54-.46*cos(2*pi*itime/time(end)); % Hamming window
% w       = sqrt(2/3)*(1-cos(2*pi*itime/time(end)));   % Hann window
% w       = .42-.5*cos(2*pi*itime/time(end))+.08*cos(4*pi*itime/time(end)); %
Blackman window

xz      = reshape(xz,1,length(xz));
kappa   = (1/length(xz))*conv(xz,fliplr(xz));
```

```
n       = 0.5*(length(kappa)-length(w));
s       = fft([zeros(1,n),w,zeros(1,n)].*kappa);
P       = abs(s(1:length(xz))); % PSD

df = Fs/length(P)/2;
f = 0:df:Fs/2-df; % frequency vector
```