Software, Hardware, and AI Visual Resume

Michael Fouche
Updated August 25, 2021

Table of Contents

Introduction

This document just covers the projects that can be discussed in public (those that were not performed in a classified environment) – some were performed in a professional environment (at work) and others were home projects or consulting projects.

I work on software projects at home in my free time and started a personal tech blog in December of 2019 - https://mikescodeprojects.com/. The blog articles cover past and current projects. Some of these articles are discussed in another part of the document.

Below is a list of the articles with the associated web links.

1. What are Super Nets? - https://mikescodeprojects.com/2021/03/19/what-are-super-nets/

2. A Software Profiler is Your Best Friend - https://mikescodeprojects.com/2021/03/03/a-software-profiler-is-your-best-friend/

3. Good Data Means Fast Neural Network Training Times - https://mikescodeprojects.com/2021/02/27/good-data-means-fast-neural-network-training-times/

4. Neural Network Performance Shaping Preview - https://mikescodeprojects.com/2021/02/07/neural-network-performance-shaping-preview/

5. Keep It Simple - https://mikescodeprojects.com/2021/01/31/keep-it-simple/

6. New Patreon Site for Learning to Apply AI - https://mikescodeprojects.com/2021/01/30/new-patreon-site-for-learning-to-apply-ai/

7. Pyrenn Levenberg-Marquardt (LM) Neural Network Training Algorithm as an Alternative to Matlab’s LM Training Algorithm - https://mikescodeprojects.com/2020/01/12/pyrenn-vs-matlab/

8. A Lesson in Perseverance: Development of a Prototype AI Neural Network Helicopter Control System - https://mikescodeprojects.com/2020/01/05/ai-helicopter-control/

9. Hands-On Introduction and Tutorial for Setting up and Running NASA’s First-Class Java WorldWind Earth Model Simulation - https://mikescodeprojects.com/2019/12/29/introduction-to-java-worldwind/

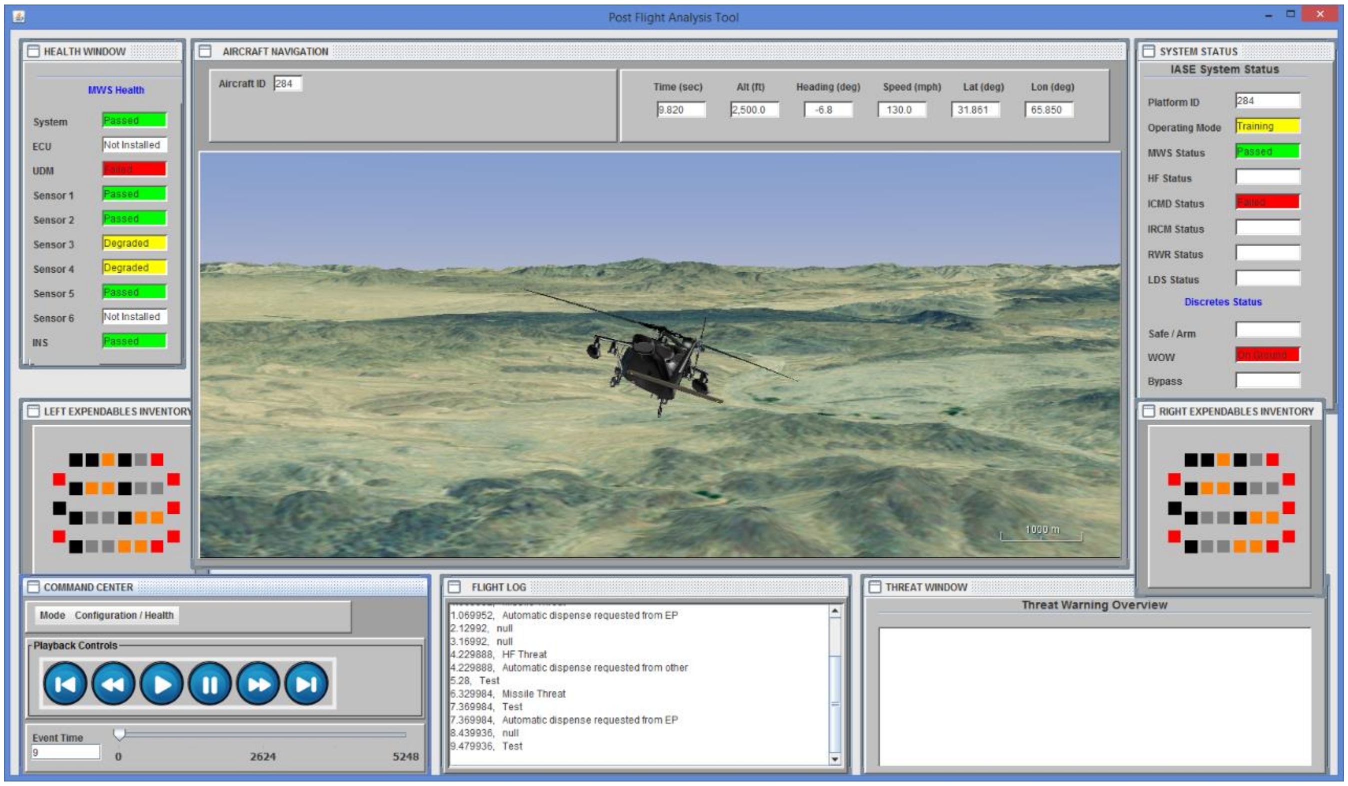
10. Using Integrators in Matlab to Simulate the Motion of an Object - https://mikescodeprojects.com/2019/12/15/integrators-in-matlab/

Software Development - Java

I have been coding in Java (to learn the Object-Oriented approach to coding) since 2013 (taught myself at home – began applying it on the job).

Java Swing – Helicopter System Monitoring

For this work project, my efforts focused more on functionality as opposed to the “user experience”. This application was designed with Java Swing – it had 11 independent threads, interacted with a MySQL database, and had an integrated NASA WorldWind Earth model API to provide visual tracking of the helicopter flight path.



MWS Health

System	Passed
ECU	Not Installed
UDM	Failed
Sensor 1	Passed
Sensor 2	Passed
Sensor 3	Degraded
Sensor 4	Degraded
Sensor 5	Passed
Sensor 6	Not Installed
INS	Passed

Aircraft ID 284

Time (sec)	Alt (ft)	Heading (deg)	Speed (mph)	Lat (deg)	Lon (deg)
9.820	2,500.0	-6.8	130.0	31.861	65.850

IASE System Status

Platform ID	284
Operating Mode	Training
MWS Status	Passed
HF Status	
ICMD Status	Failed
IRCM Status	
RWR Status	
LDS Status	

Discretes Status

Safe / Arm	
WOW	On Ground
Bypass	

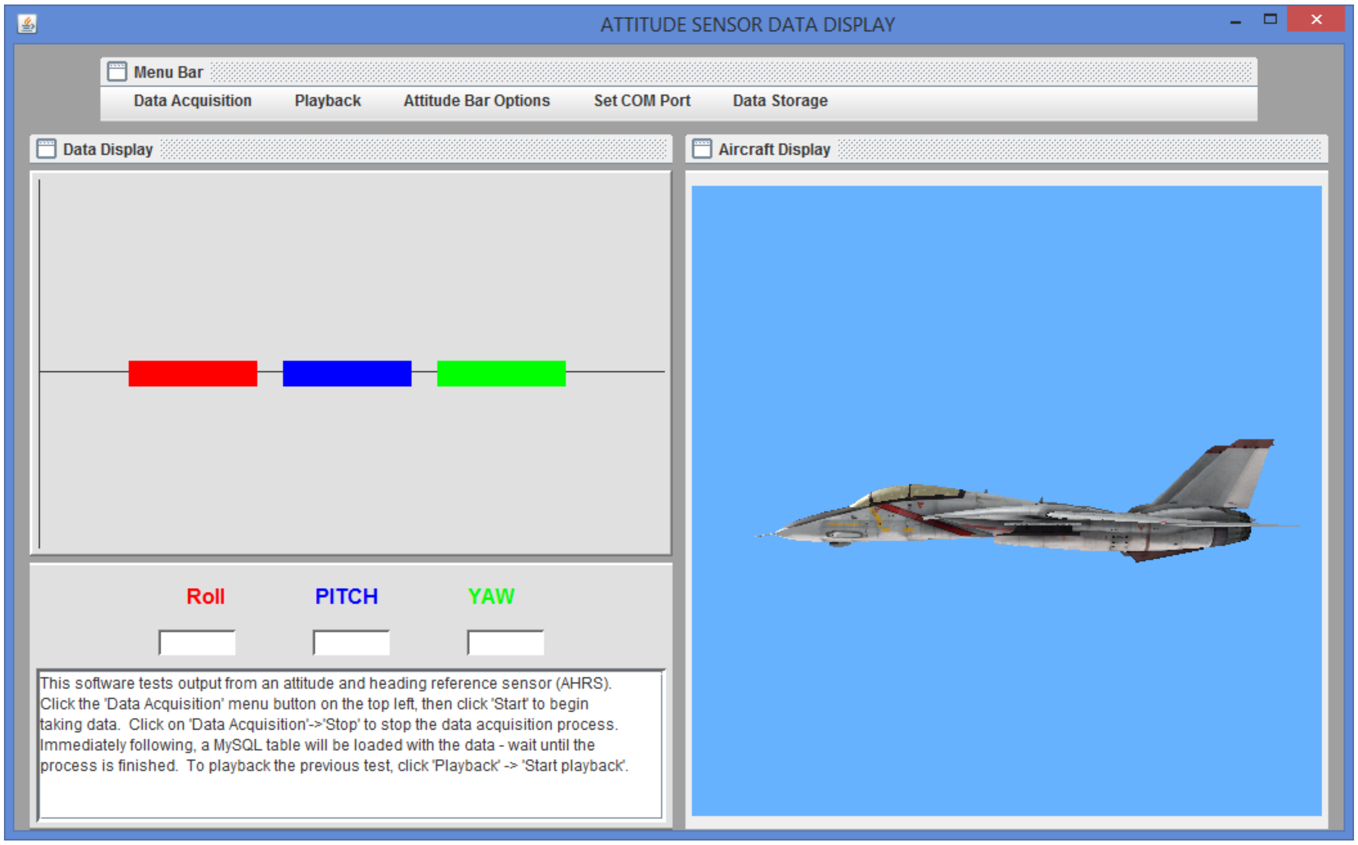
Mode Configuration / Health

Event Time 9

```
1.069952,  Automatic dispense requested from EP
2.12992,  null
3.16992,  null
4.229888,  HF Threat
4.229888,  Automatic dispense requested from other
5.28,  Test
6.329984,  Missile Threat
7.369984,  Test
7.369984,  Automatic dispense requested from EP
8.439936,  null
9.479936,  Test
```

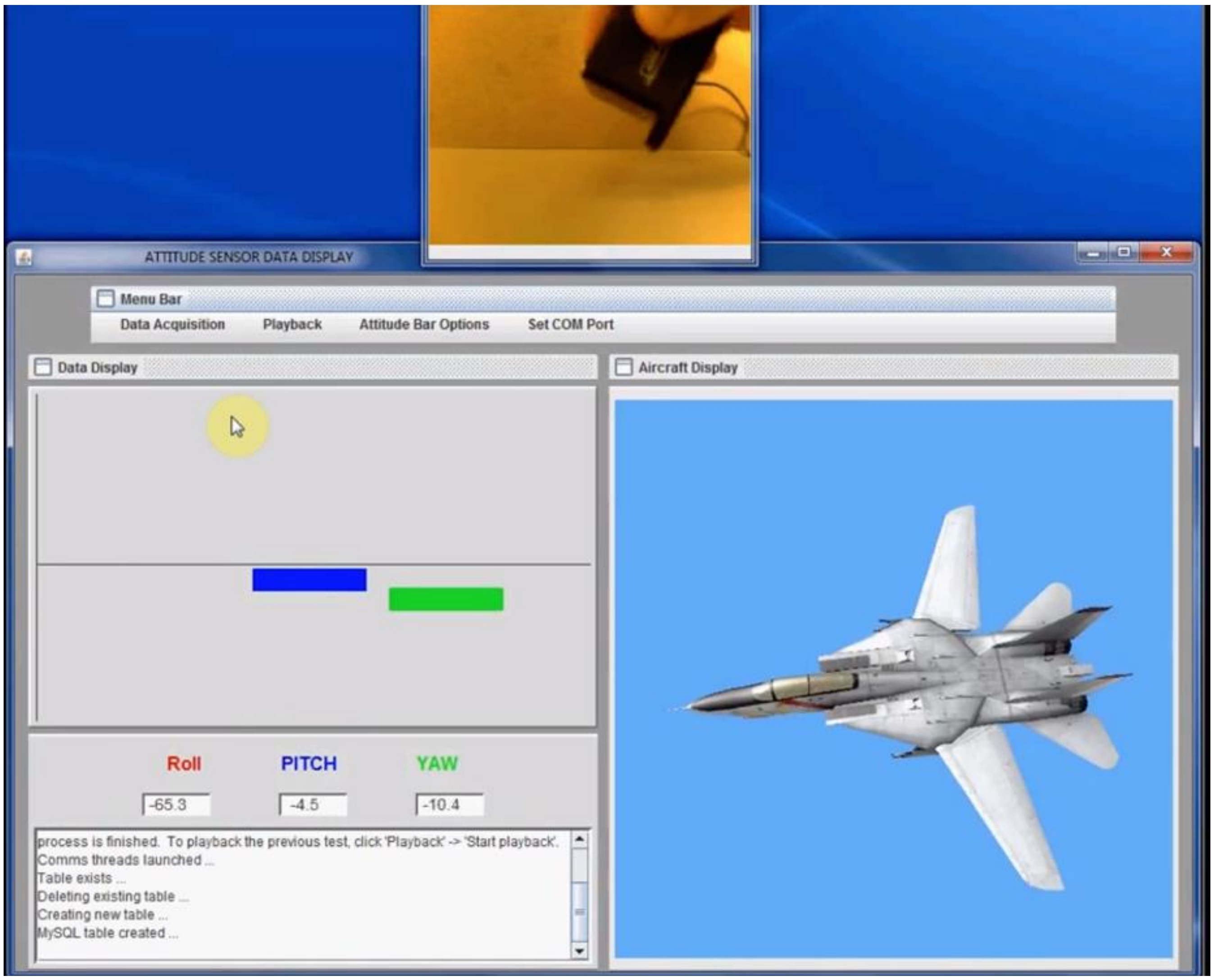
Java Swing – Inertial Sensor (AHRS) Attitude Display

The following was home project that was designed with Java Swing and was connected to a MySQL database. It processes data from an inertial sensor (stabilized roll, pitch, and heading) via an RS-232 port and displays the attitude information digitally as well as graphically with the F-14 aircraft model. The software has two independent threads, interacts with a MySQL database to allow "playback", etc.



This software tests output from an attitude and heading reference sensor (AHRS). Click the 'Data Acquisition' menu button on the top left, then click 'Start' to begin taking data. Click on 'Data Acquisition'->'Stop' to stop the data acquisition process. Immediately following, a MySQL table will be loaded with the data - wait until the process is finished. To playback the previous test, click 'Playback' -> 'Start playback'.

A demonstration in Windows can be seen here - https://youtu.be/NR0lmkqU4Qc and a demonstration in Linux can be seen here - https://youtu.be/2DDSxk_vrj0. Below is a screenshot from the Windows video – the aircraft model follows the attitude of the inertial sensor being manipulated by my hand.

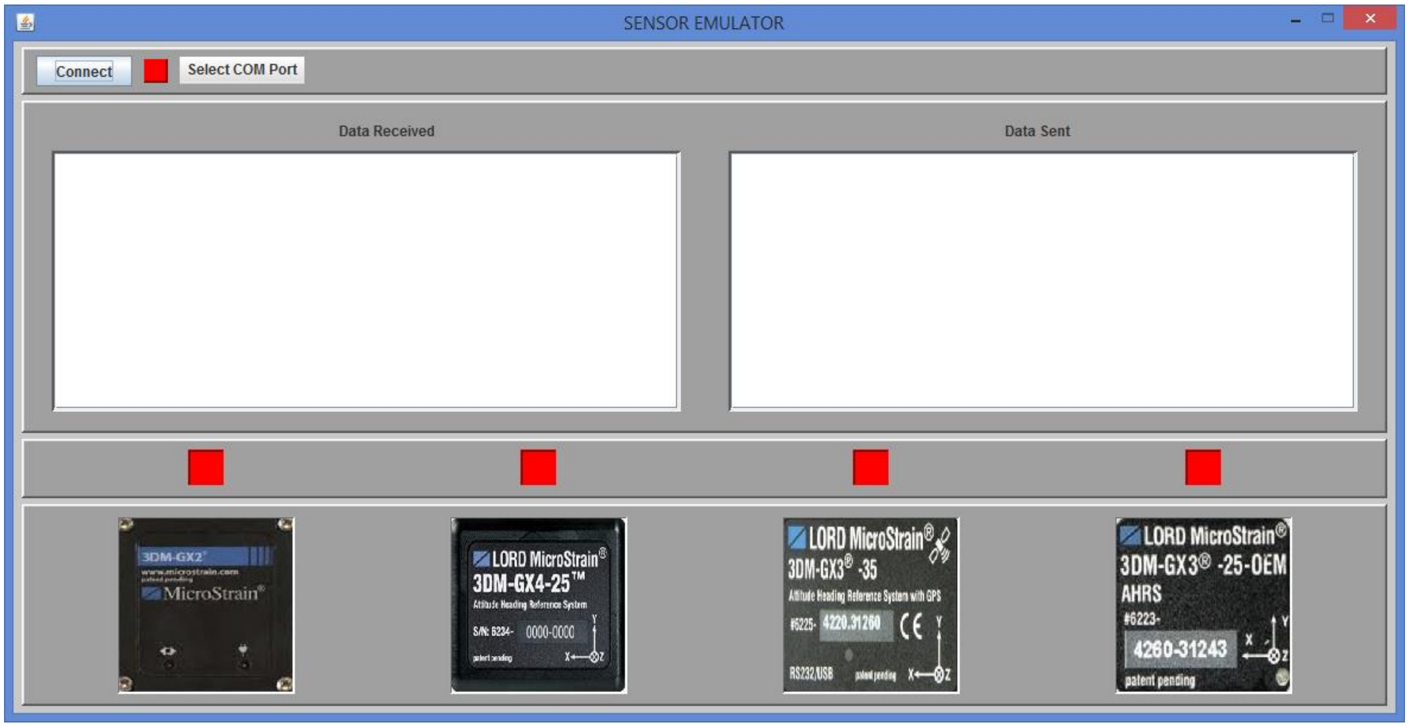


-65.3   -4.5   -10.4

```
process is finished.  To playback the previous test, click 'Playback' -> 'Start playback'.
Comms threads launched ...
Table exists ...
Deleting existing table ...
Creating new table ...
MySQL table created ...
```

Java Swing – Inertial Sensor (AHRS) Emulator

As part of some consulting work, I built a Java Swing application that was complementary to the previous Java Swing sensor project. This application was designed to replace the real inertial sensor with a "sensor emulator" that mimics the data coming from the actual sensor. I was able to complete the communications protocols for two of the sensor units (the two black box areas shown in the left bottom panel) but did not complete the protocols for the other two sensors (as they were not needed). The Swing GUI was simple but functional. The user selects a COM port and knows that the RS-232 connection is good when the light, next to the "Connect" button, turns from red to green. The received data is displayed in the left text window while the sent data is displayed in the right text window. The light above the image of the sensor changes from red to green when that sensor is being emulated.



Java NASA WorldWind Tutorial from Personal Tech Blog

I wrote an article on my personal tech blog - https://mikescodeprojects.com/2019/12/29/introduction-to-java-worldwind/, that discusses, in detail, the fundamentals of getting started with NASA's Java WorldWind Earth model simulation. I wrote all of the code on my own time at home - the article includes videos of code walk-throughs and downloadable source code.

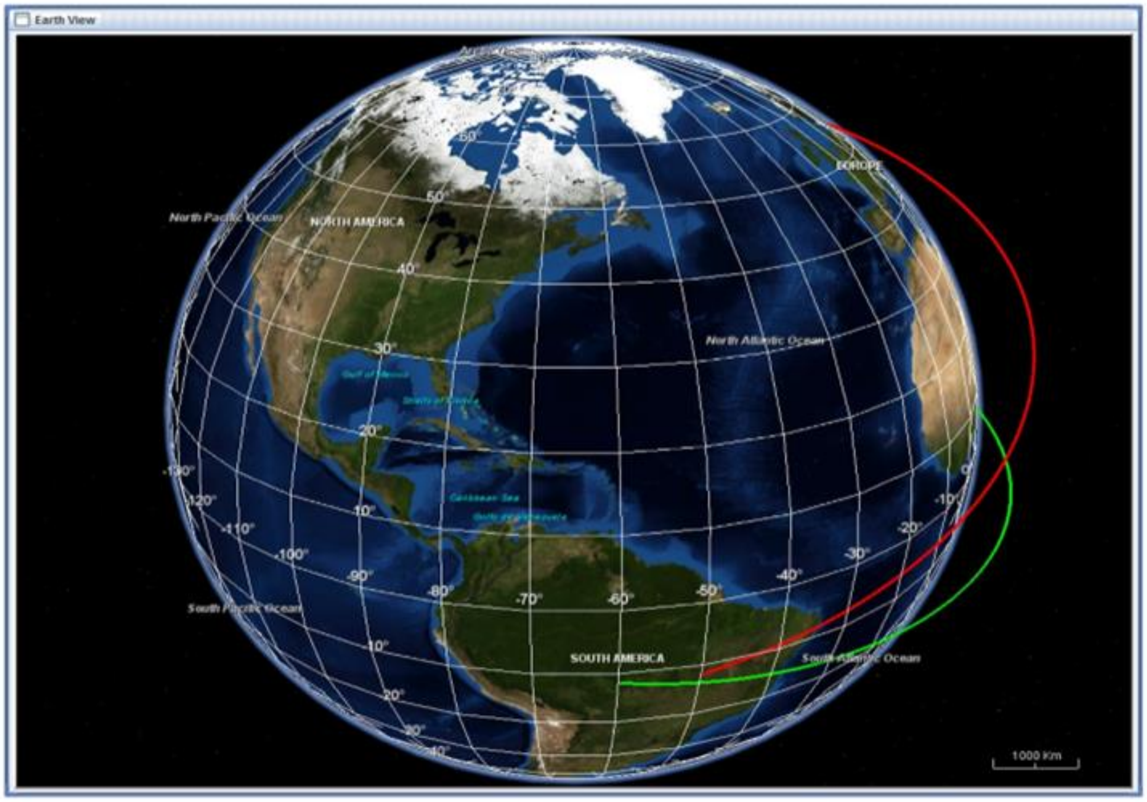
Mike's Coding Blog

This is a place for me to rant / rejoice as I build various new software projects and write about old ones.

Hands-On Introduction and Tutorial for Setting up and Running NASA's First-Class Java WorldWind Earth Model Simulation

Joys and Sorrows of Coding December 29, 2019 Uncategorized

aircraft, earth, earth model, java, model, NASA, NetBeans, radar, simulation, software development, spacecraft, trajectories, world wind, worldwind



What Is It?

What is WorldWind? Well let me quote NASA's site directly:

WorldWind is an open source virtual globe API. WorldWind allows developers to quickly and easily create interactive visualizations of 3D globe, map and geographical information. Organizations around the world use WorldWind to monitor weather patterns, visualize cities and terrain, track vehicle movement, analyze geospatial data and educate humanity about the Earth.

Graphical Ballistic Missile Engagement Simulation (MDA – GMD)

This was a classified program and thus no graphics are available for display.

Java Swing – Ballistic Missile / Countermeasures Scenario Display

I developed a Java (with NASA WorldWind Earth model API) GUI-driven ballistic missile / interceptor flight analysis software application for Verification and Validation (V&V) of Ground Based Missile (GMD) systems and subsystems. The application could process Monte Carlo runs from a trajectory generator for validation of statistical measures and presented a full view of potential engagements.

Java Servlets

As a side project, I developed a specialized Java Servlet package using Apache Tomcat, JavaScript & Ajax/jQuery, MySQL, and HTML to support development of a co-worker's distributed analysis tool. This was a specialized case where an Apache Tomcat CORS filter was set to allow JavaScript code to communicate with client files (bypassing normal security protocols which was acceptable since it was a secure, closed environment). I did the whole setup myself – built up the backend (Servlet and other Java parts) as well as the front end testing code.

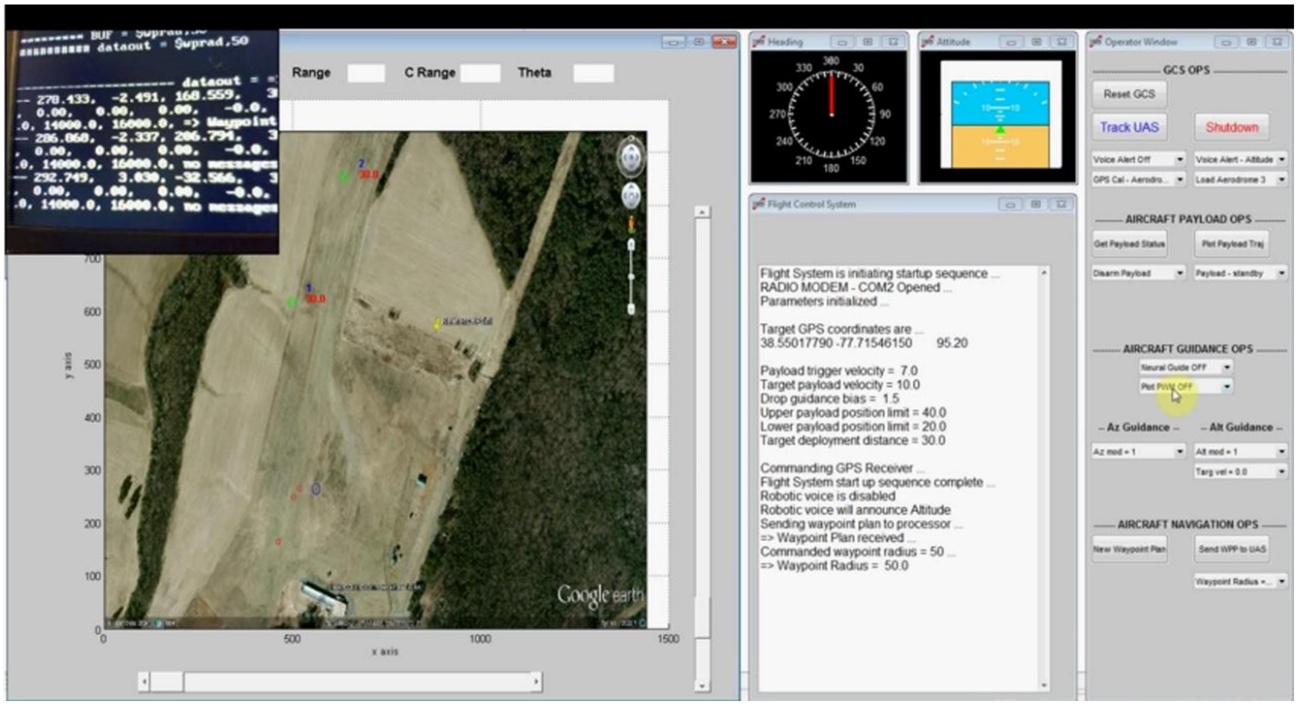
Java / Matlab Client-Server

As a side project related to this effort, I developed a Matlab server package to service Matlab and Java clients (TCP, stand-alone executables), using the Instrument Control toolbox.

Software Development - Matlab

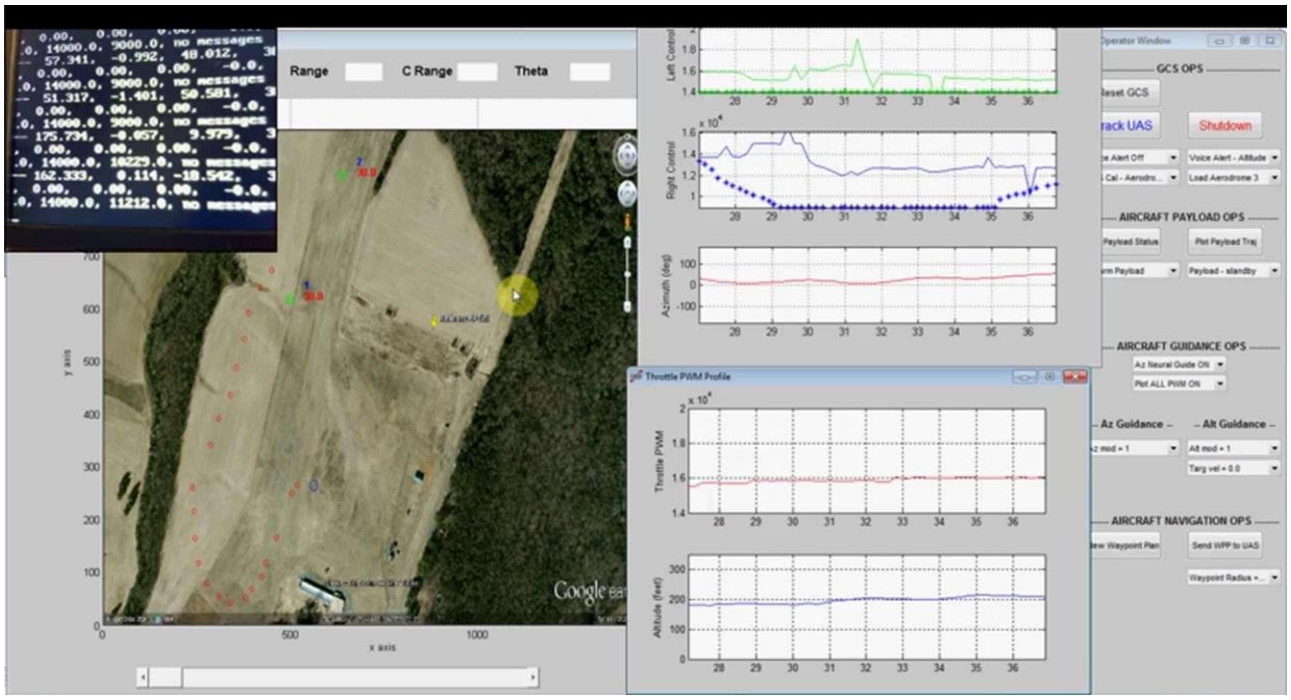
Real-Time UAV Ground Control Station

The following two screenshots are from a Matlab Unmanned Aerial Vehicle (UAV) ground control station (GCS) application that I designed / coded (consulting work) and used for a UAV project in Northern Virginia (was used for real-time flight telemetry & command). A video desktop demonstration can be seen here - https://youtu.be/mjqOFOr5b8c.



```
========= BUF = $wprad,50
=========== dataout = $wprad,50

------------------------- dataout = =
-- 278.433,  -2.491, 168.559,   3
,  0.00,    0.00,    0.00,    -0.0,
.0, 14000.0, 16000.0, => Waypoint
-- 286.068,  -2.337, 206.794,   3
,  0.00,    0.00,    0.00,    -0.0,
.0, 14000.0, 16000.0, no messages
-- 292.749,   3.030, -32.566,   3
,  0.00,    0.00,    0.00,    -0.0,
.0, 14000.0, 16000.0, no messages
```

```
Flight System is initiating startup sequence ...
RADIO MODEM - COM2 Opened ...
Parameters initialized ...

Target GPS coordinates are ...
38.55017790 -77.71546150      95.20

Payload trigger velocity =  7.0
Target payload velocity = 10.0
Drop guidance bias =  1.5
Upper payload position limit = 40.0
Lower payload position limit = 20.0
Target deployment distance = 30.0

Commanding GPS Receiver ...
Flight System start up sequence complete ...
Robotic voice is disabled
Robotic voice will announce Altitude
Sending waypoint plan to processor ...
=> Waypoint Plan received ...
Commanded waypoint radius = 50 ...
=> Waypoint Radius =  50.0
```

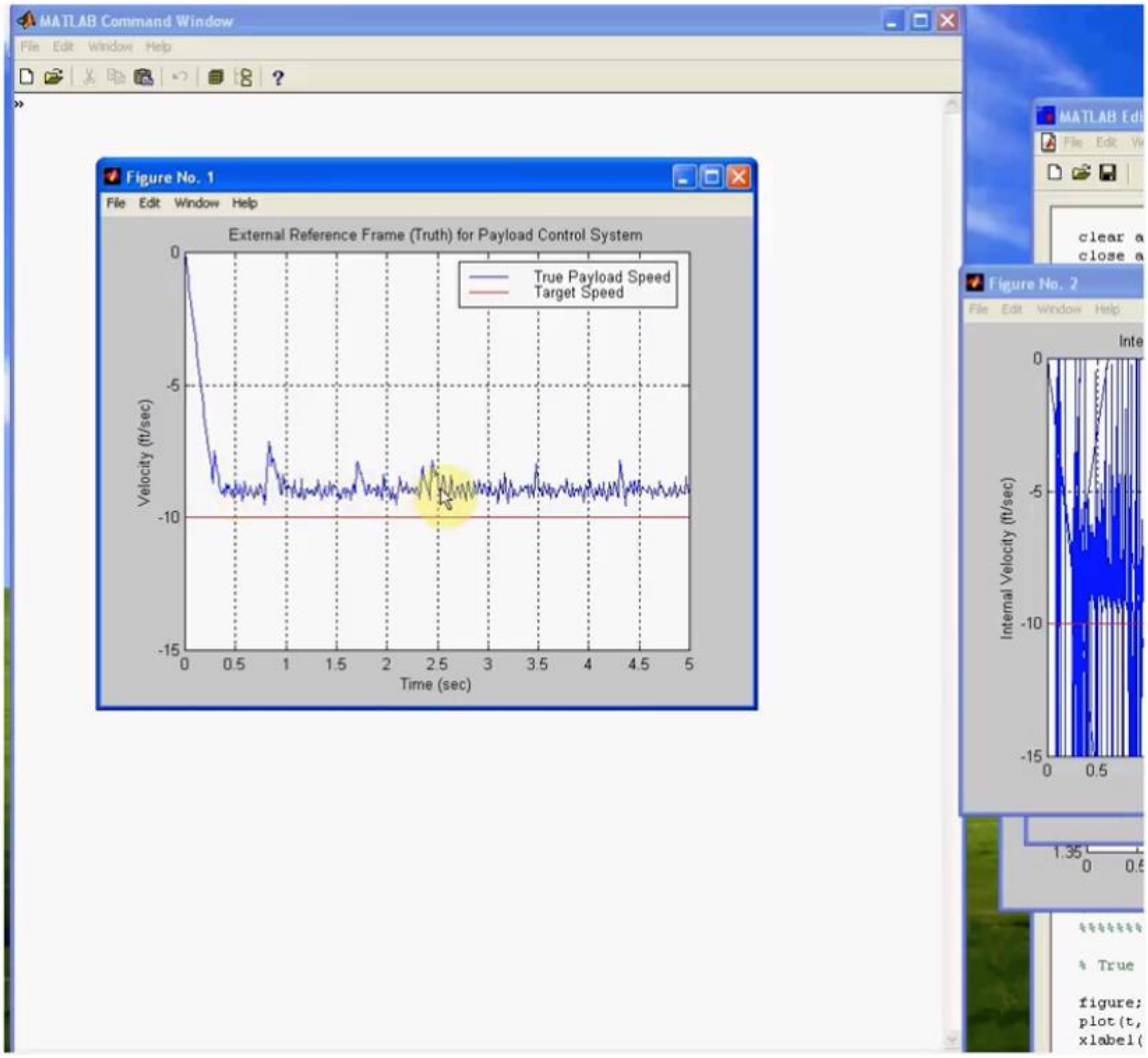


```
, 0.00,  0.00,  0.00,
.0, 14000.0, 9000.0, no messages
—  57.341,  -0.992,  48.012,   3
,  0.00,  0.00,  0.00,   -0.0,
.0, 14000.0, 9000.0, no messages
—  51.317,  -1.401,  50.581,   3
,  0.00,  0.00,  0.00,   -0.0,
.0, 14000.0, 9000.0, no messages
— 175.734,  -0.057,   9.979,   3
,  0.00,  0.00,  0.00,   -0.0,
.0, 14000.0, 10229.0, no messages
— 162.333,   0.114, -18.542,   3
,  0.00,  0.00,  0.00,   -0.0,
.0, 14000.0, 11212.0, no messages
```

Payload Deployment Controller Modeling

The following is a Matlab model that I developed (consulting work) of a payload controller (which used a simple Proportional-Derivative control law) used in the DARPA-funded powered-parafoil UAV project. The purpose was to try and understand some anomalies that we were seeing with different brake components (which controlled the descent rate of the payload).

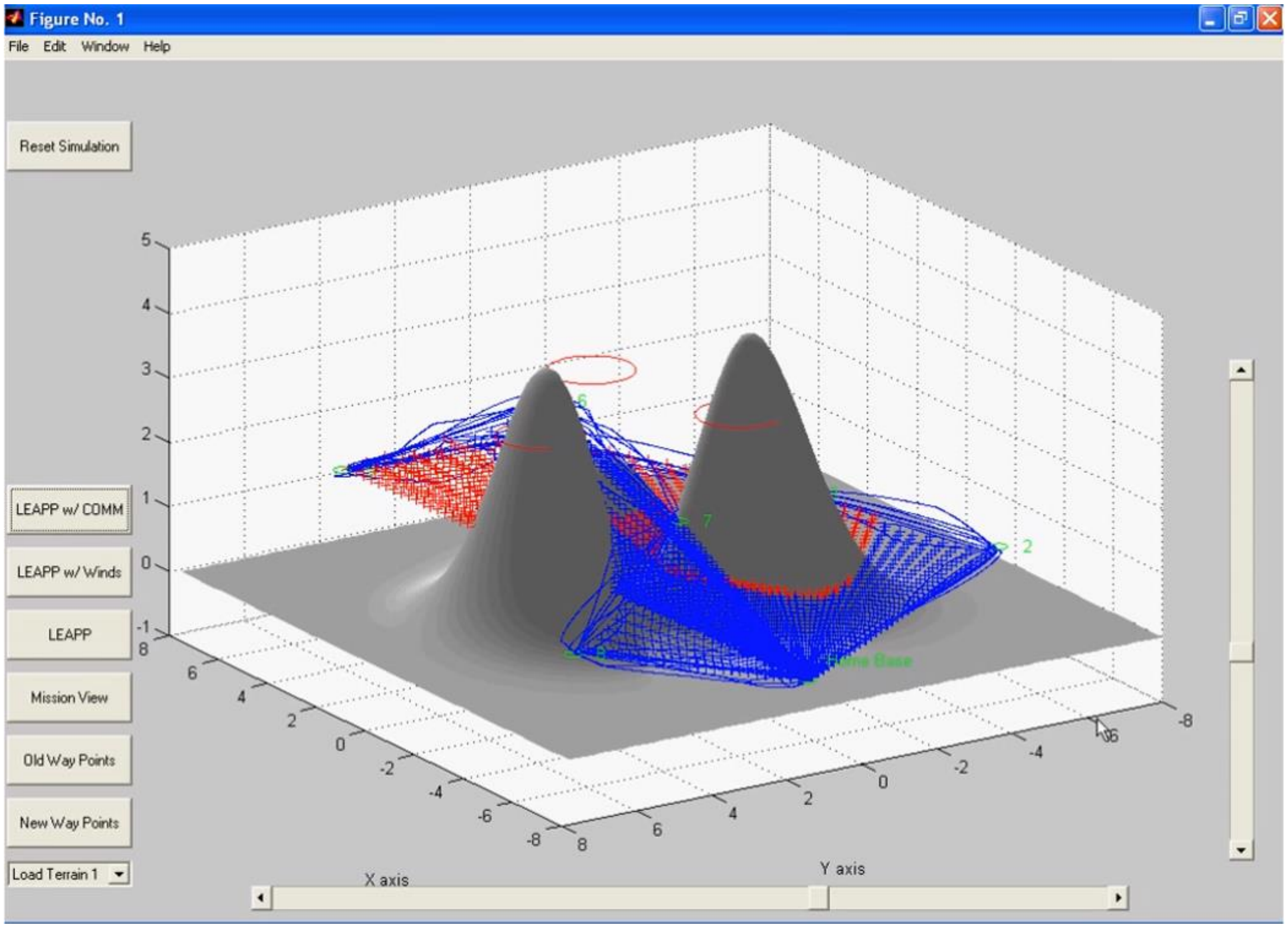
The video can be viewed here - https://youtu.be/BpsH2wC0pIc.



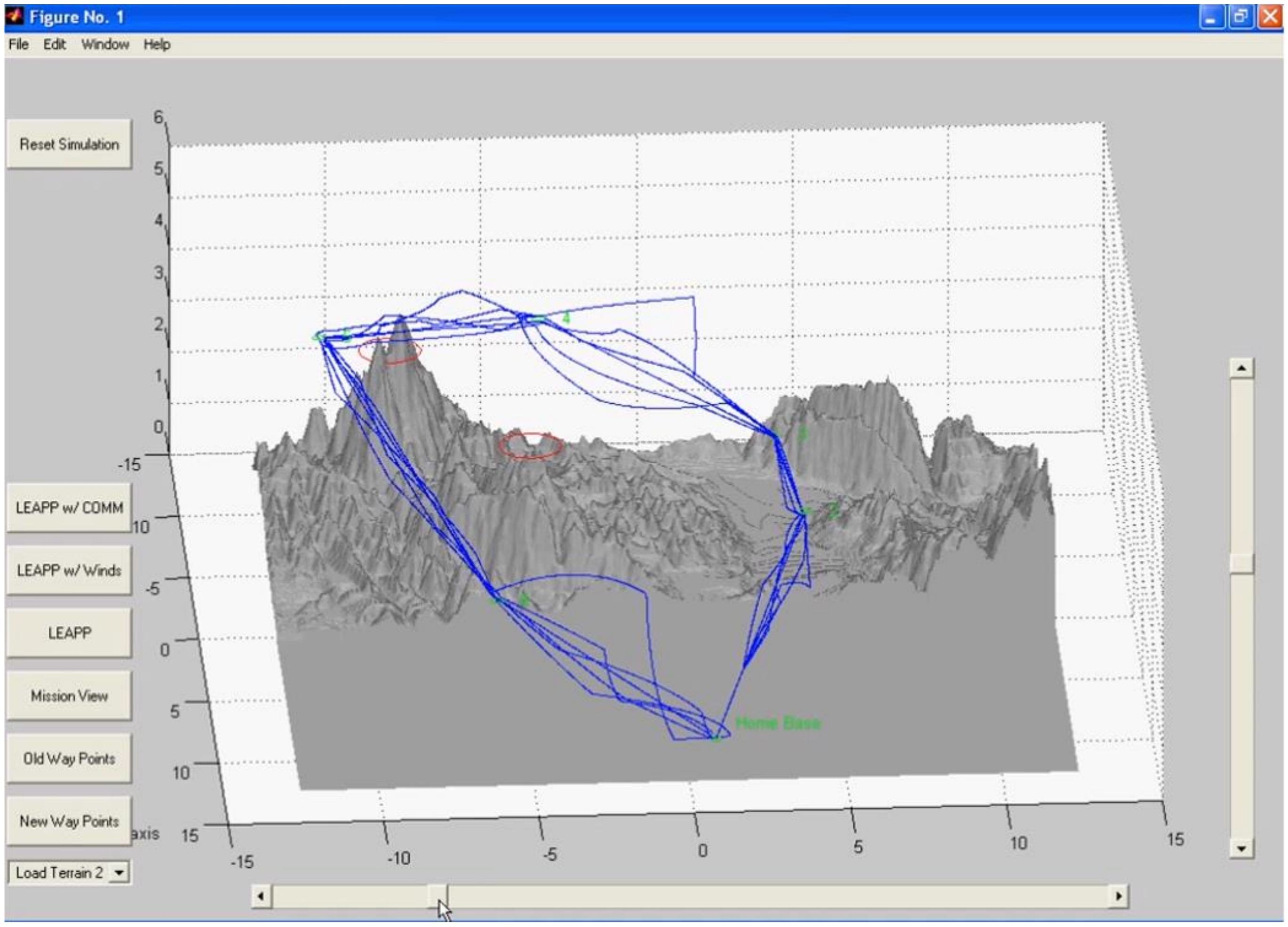
Simulated UAV Trajectory and Communications Modelling

I designed / coded a “rough draft” Matlab application (consulting work) that models UAV trajectories with user-provided waypoints and “avoidance points”. It models the basic trajectories as well as performs Monte Carlo simulations to measure trajectory dispersions as a result of random wind profiles. In addition, basic line-of-sight communications is modeled as well. A demonstration video can be viewed here https://youtu.be/xnn3n5mwPd8.

The screenshot below shows light-of-sight visuals for a series of trajectories.

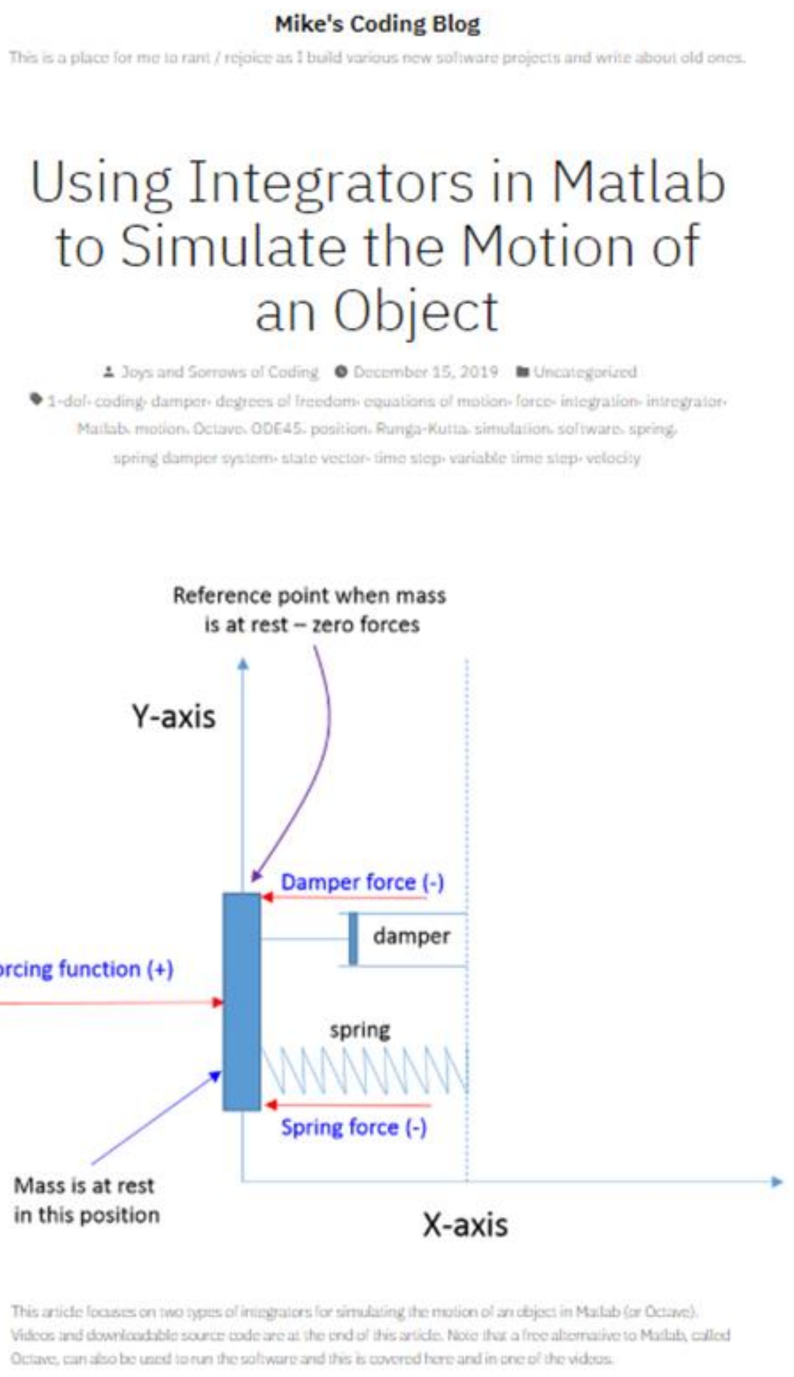


The screenshot below shows a small Monte-Carlo dispersion analysis simulation (to observe the effects of unexpected turbulent winds).



Matlab Motion Simulation Tutorial from Personal Tech Blog

I wrote an article on my personal tech blog - https://mikescodeprojects.com/2019/12/15/integrators-in-matlab/, that discusses, in detail, the fundamentals of integrating the equations of motion for a 2-D spring-damper system in Matlab. The application was coded on my own time - the blog article main page is shown below.



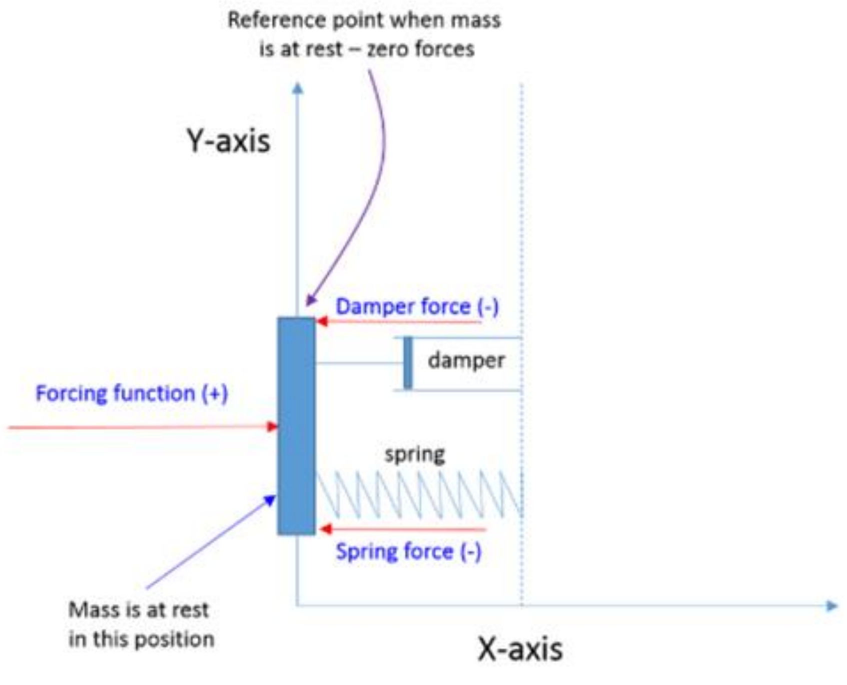
Mike's Coding Blog

This is a place for me to rant / rejoice as I build various new software projects and write about old ones.

Using Integrators in Matlab to Simulate the Motion of an Object

Joys and Sorrows of Coding December 15, 2019 Uncategorized

1-dof, coding, damper, degrees of freedom, equations of motion, force, integration, intregrator, Matlab, motion, Octave, ODE45, position, Runga-Kutta, simulation, software, spring, spring damper system, state vector, time step, variable time step, velocity

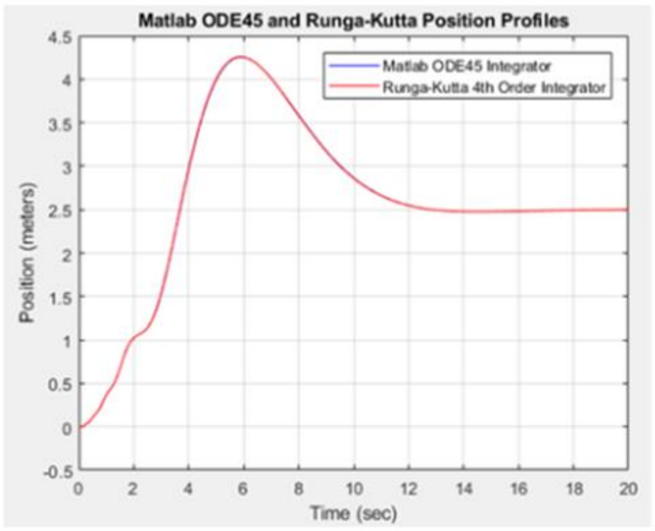

Reference point when mass is at rest – zero forces
Y-axis
Damper force (-)
damper
Forcing function (+)
spring
Spring force (-)
Mass is at rest in this position
X-axis

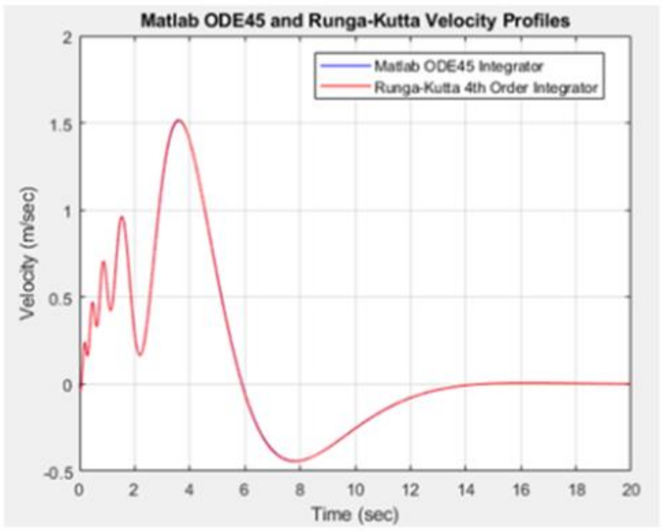
This article focuses on two types of integrators for simulating the motion of an object in Matlab (or Octave). Videos and downloadable source code are at the end of this article. Note that a free alternative to Matlab, called Octave, can also be used to run the software and this is covered here and in one of the videos.

This screenshot below shows the section that discusses the test results for the comparison of a hand-coded Runge-Kutta integration function against Matlab's ODE45 variable step-size integrator function. Note that I incorrectly spelled "Runge-Kutta" in the graphics plots so I'll be making that correction in the Matlab code.

Position and Velocity Profiles

Given that ODE45 data is plotted in blue and the Runga-Kutta data is plotted in red, it's hard to see any differences in the trajectories – they are basically identical.





Software Development - C

Real-Time Closed-Loop Autonomous Flight Control System for Helicopter UAV

I developed (home and work effort) an autonomous flight control system (using Neural Networks for the inner and outer loop control modules) for small Radio-Control (RC) sized helicopters (gas, electric, and jet power-plants). The flight software was coded in C running on the Dos 6.2 operating system. The screenshot below was from an onboard camera that swiveled back such that part of the helicopter could be viewed as well as the parking lot area. The autonomous flight can be observed here - https://youtu.be/YhMmkRHn-14.

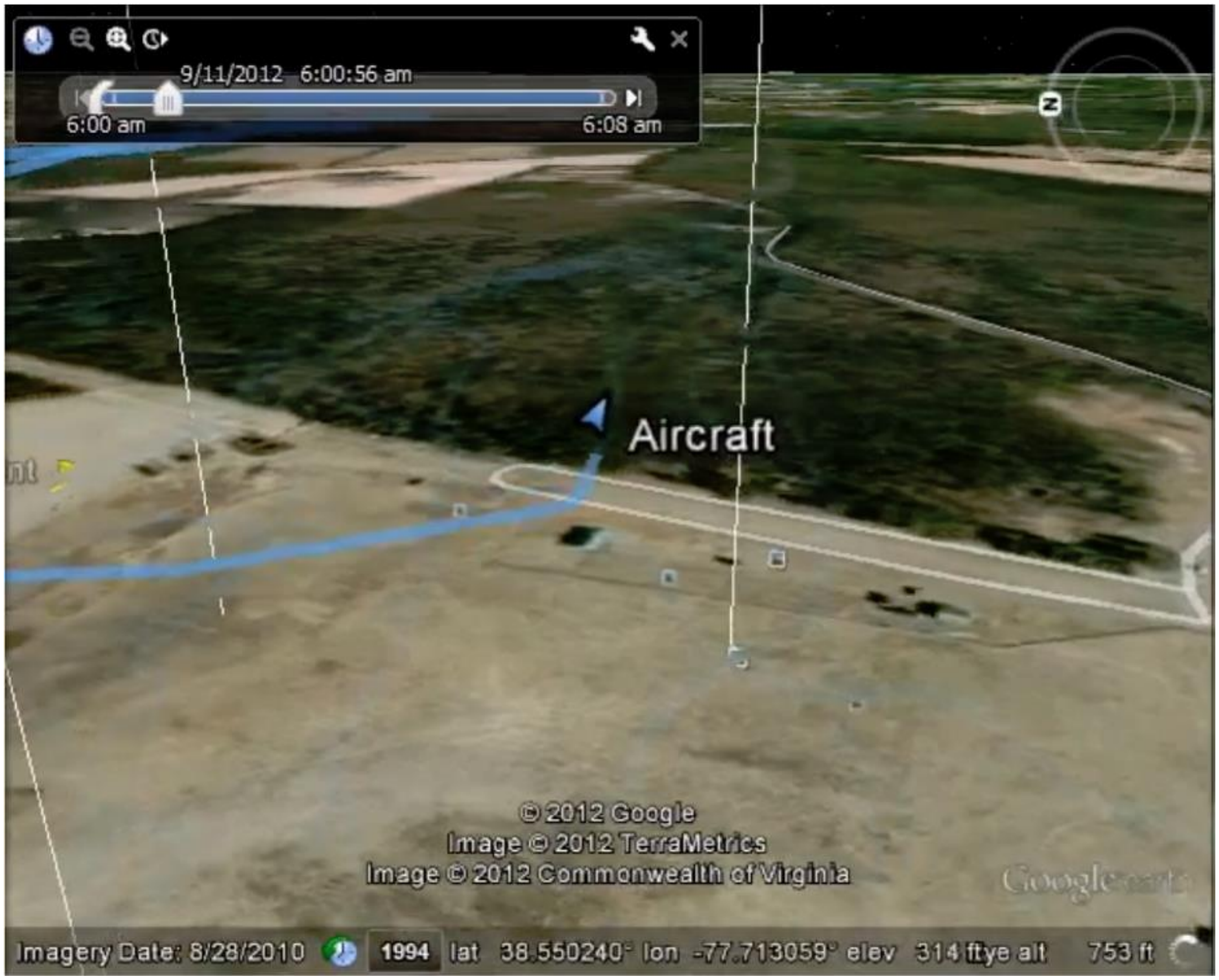
I wrote a couple of articles about this experience (each includes original flight test videos):

Keep It Simple - https://mikescodeprojects.com/2021/01/31/keep-it-simple/

A Lesson in Perseverance - https://mikescodeprojects.com/2020/01/05/ai-helicopter-control/

Autonomous Guidance System for DARPA-Funded Powered-Parafoil UAV

I developed (consulting work) an autonomous flight control system (using Neural Networks for guidance modules) for a DARPA-funded powered-parafoil UAV. The flight software was coded in C running on a Linux operating system. The screenshot below was from a flight test with 4 waypoints – the GPS flight data was processed through Google Earth as a virtual fly-through for this video. The screenshot below shows the UAV having just tagged one waypoint (the vertical white lines) and is on its way to the next waypoint. The flight test can be observed here - https://youtu.be/bhBnOthL9Mw.


9/11/2012 6:00:56 am
6:00 am
6:08 am
Aircraft
© 2012 Google
Image © 2012 TerraMetrics
Image © 2012 Commonwealth of Virginia
Google earth
Imagery Date: 8/28/2010
1994
lat 38.550240° lon -77.713059° elev 314 ft eye alt 753 ft

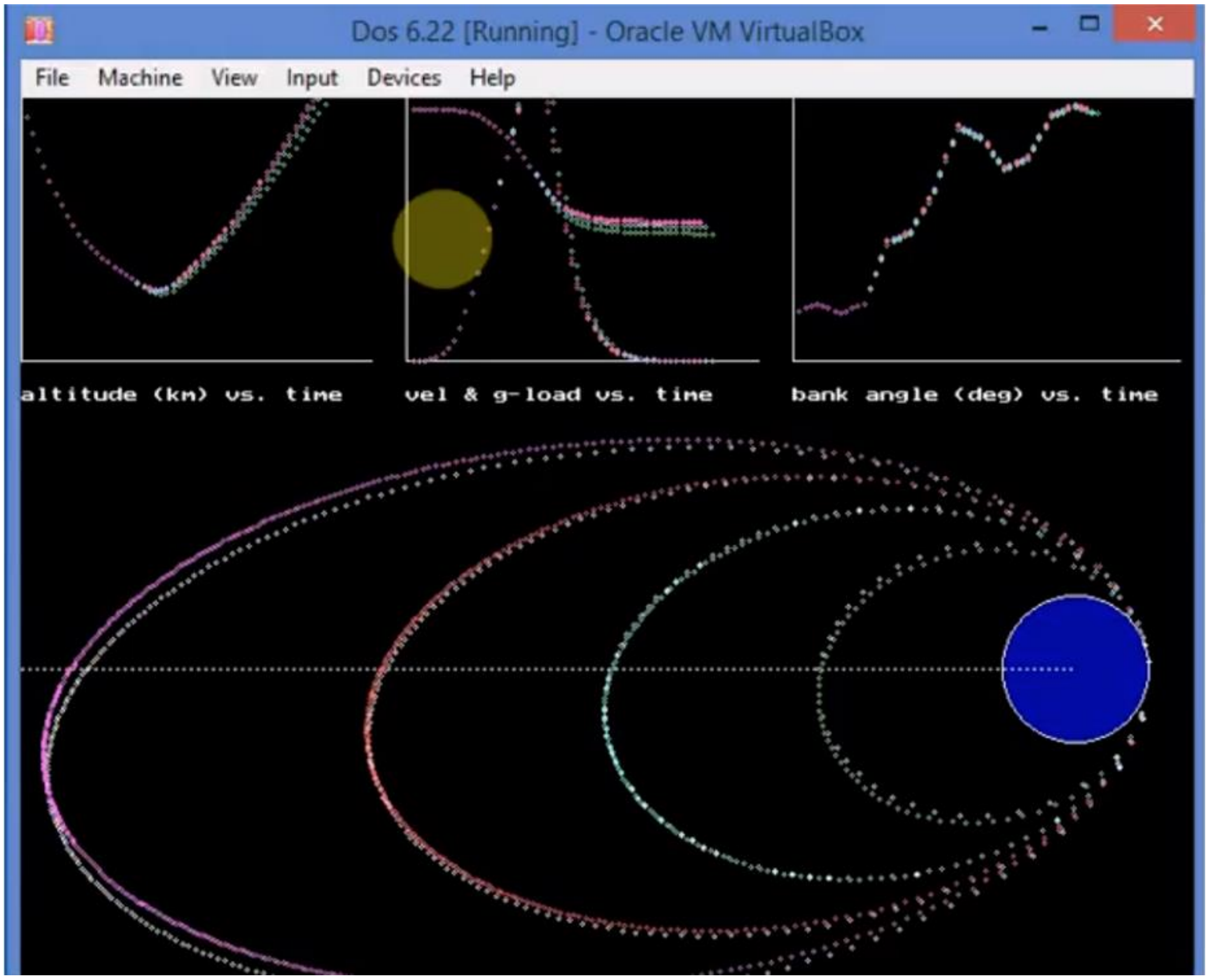
Simulated Closed-Loop Autonomous Flight Control of Aerobraking Vehicle

I developed (at work) a simulated closed-loop aerocapture (at Mars) Neural Network guidance algorithm (coded in C running on Dos 6.2) – the guidance system successfully achieved targeted orbit apogee and inclination elements via modulation of the lift vector during the pass through the atmosphere.

The guidance system performance was tested with worst-case disturbances / system changes – in all cases the Neural Network guidance algorithm performance far exceeded the performance of Boeing and Draper Labs predictor-corrector algorithms. Dispersion testing included:

- Increased and decreased atmosphere density profile values.
- Introduced latency in the steering vector response.
- Introduced errors in the aerodynamic model coefficients.

The simulation can be observed here - https://youtu.be/3bKmHycpnhE.


Dos 6.22 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
altitude (km) vs. time
vel & g-load vs. time
bank angle (deg) vs. time

Software Development - C++

U.S. Army Aircraft Survivability Equipment Laboratory

I developed (at work) an unclassified multi-threaded C++ simulation tool, using NetBeans/Cygwin Windows and Red Hat Linux environments, to: 1) generate flight trajectory data, 2) encode 1553 data packets and put them on a 1553 bus (as an RT), and 3) act as a server to exchange data, via TCP/IP protocol, with a Java client (Client / Server configuration).

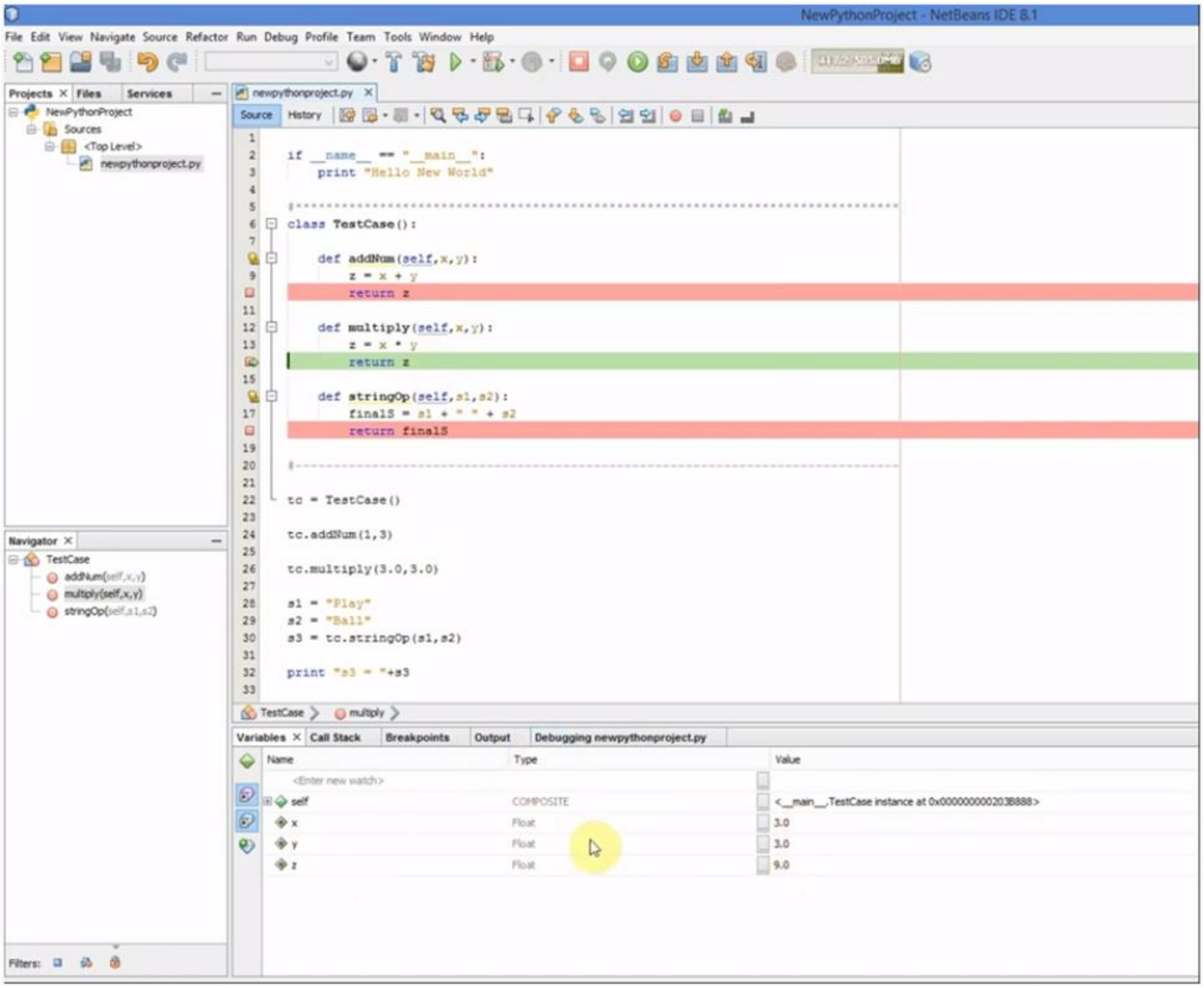
NASA SLS Avionics Data Processing

I developed (at work) a software application in C++ (using Visual Studio 2015) for a multi-threaded framework application that sends data from multiple avionics models via TCP/IP to a 1553 data bus. At its core, it is a basic Client / Server architecture that is designed for speed given the tight timing requirements.

Software Development - Python

NetBeans Python Setup Tutorial

This was a simple video tutorial that I put together at home for a colleague - https://youtu.be/yihB0-D0Esg.
The purpose was to demonstrate the setup of Python in the NetBeans 8.2 IDE.

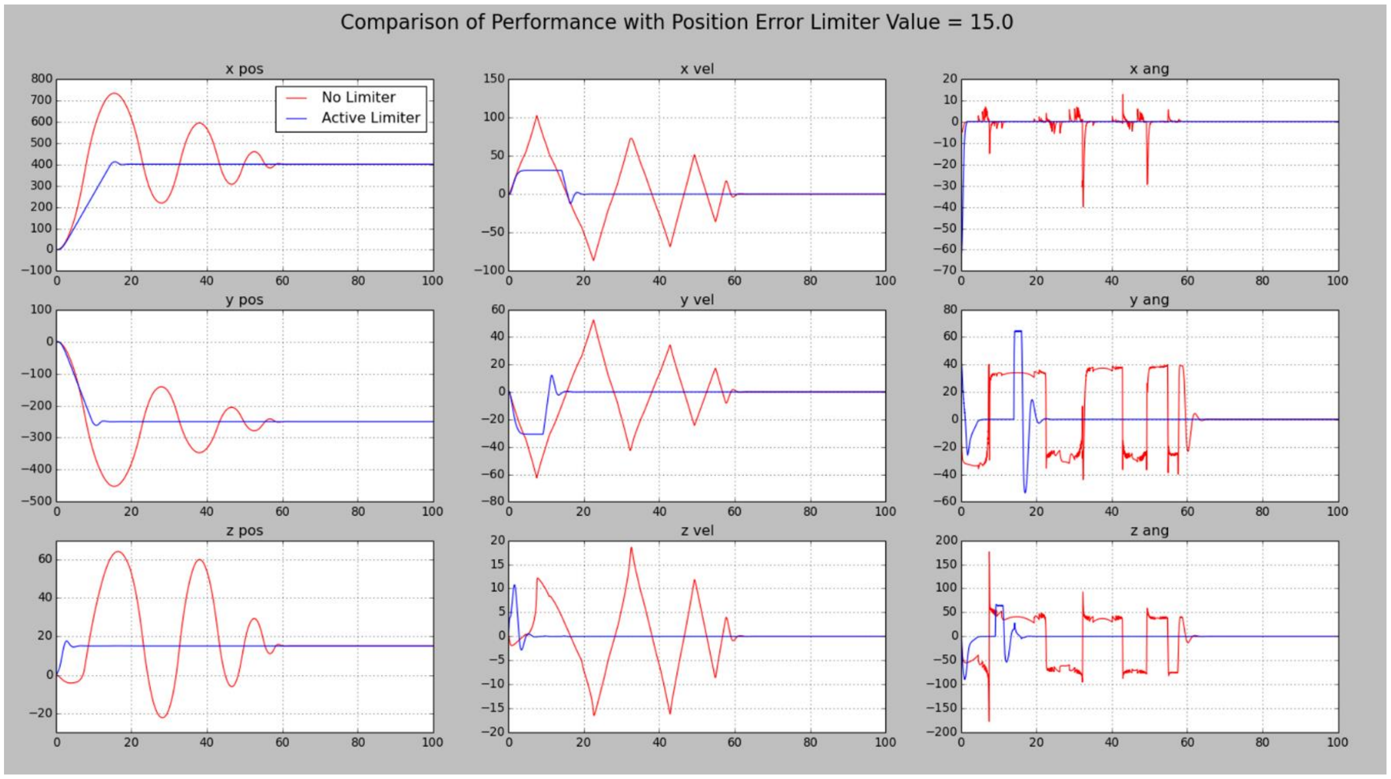


```

if __name__ == "__main__":
    print "Hello New World"

#*****************************************************************************
class TestCase():

    def addNum(self,x,y):
        z = x + y
        return z

    def multiply(self,x,y):
        z = x * y
        return z

    def stringOp(self,s1,s2):
        finalS = s1 + " " + s2
        return finalS

#-----------------------------------------------------------------------------

tc = TestCase()

tc.addNum(1,3)

tc.multiply(3.0,3.0)

s1 = "Play"
s2 = "Ball"
s3 = tc.stringOp(s1,s2)

print "s3 = "+s3

```

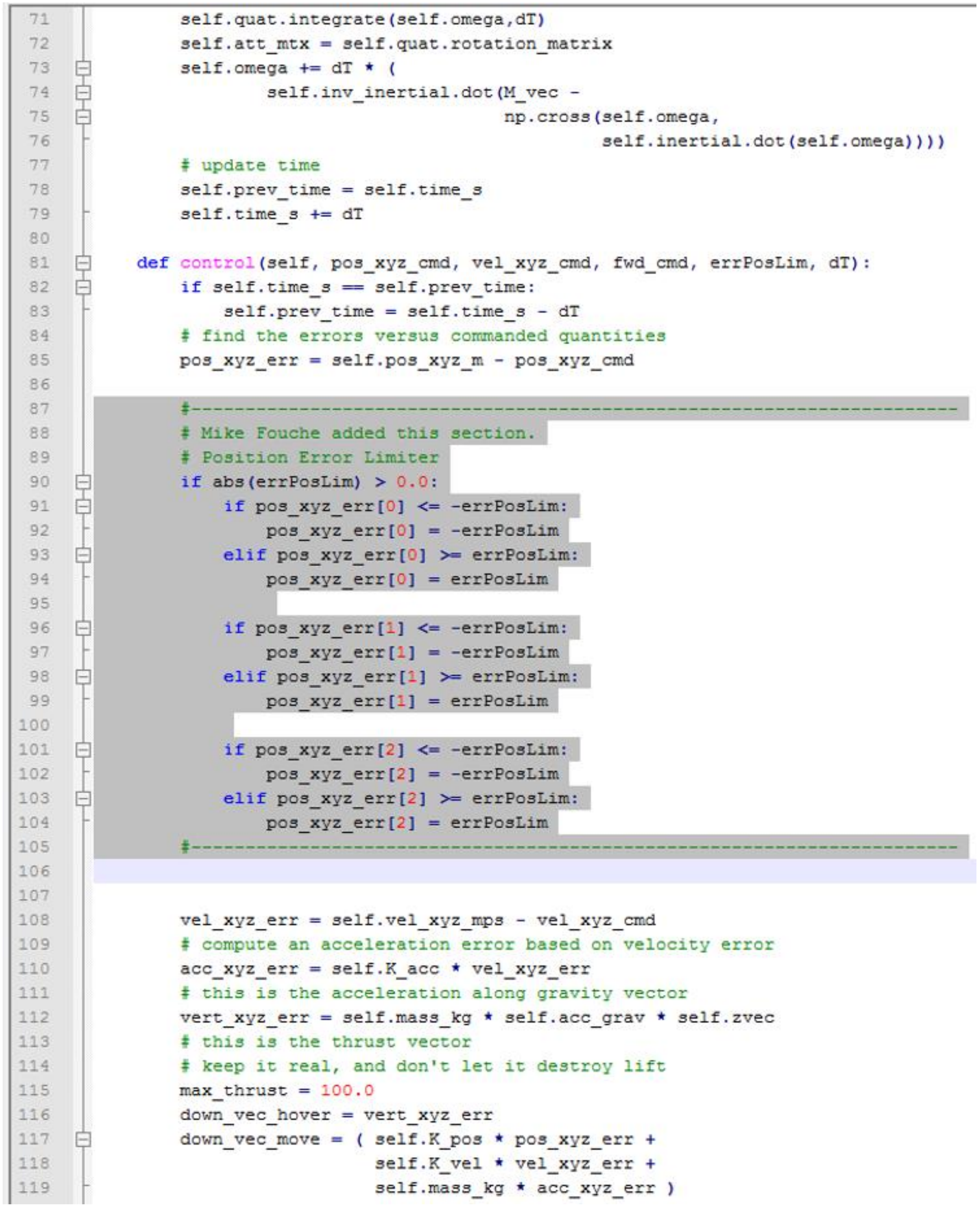
Name	Type	Value
<Enter new watch>		
self	COMPOSITE	<__main__.TestCase instance at 0x000000000203B888>
x	Float	3.0
y	Float	3.0
z	Float	9.0

Simulated UAV Flight Control Stability Resolution

On a consulting basis, I was asked to resolve a flight control instability issue for a UAV simulation that was coded in Python running on Linux. I'd never worked with Python in the past but was able to get acclimated quickly using NetBeans as the Python IDE running on Ubuntu Linux. Within a short time, I determined that controller input limiters were needed and this indeed resolved the problem. Below is a set of plots that show the UAV flight performance (when attempting to achieve a target position / velocity) differences between the system without the controller input limiters (red) and the system with the controller input limiters (blue). The improvement in performance was dramatic but not unexpected.



The gray area in the Python code screenshot below is the actual fix for the problem.

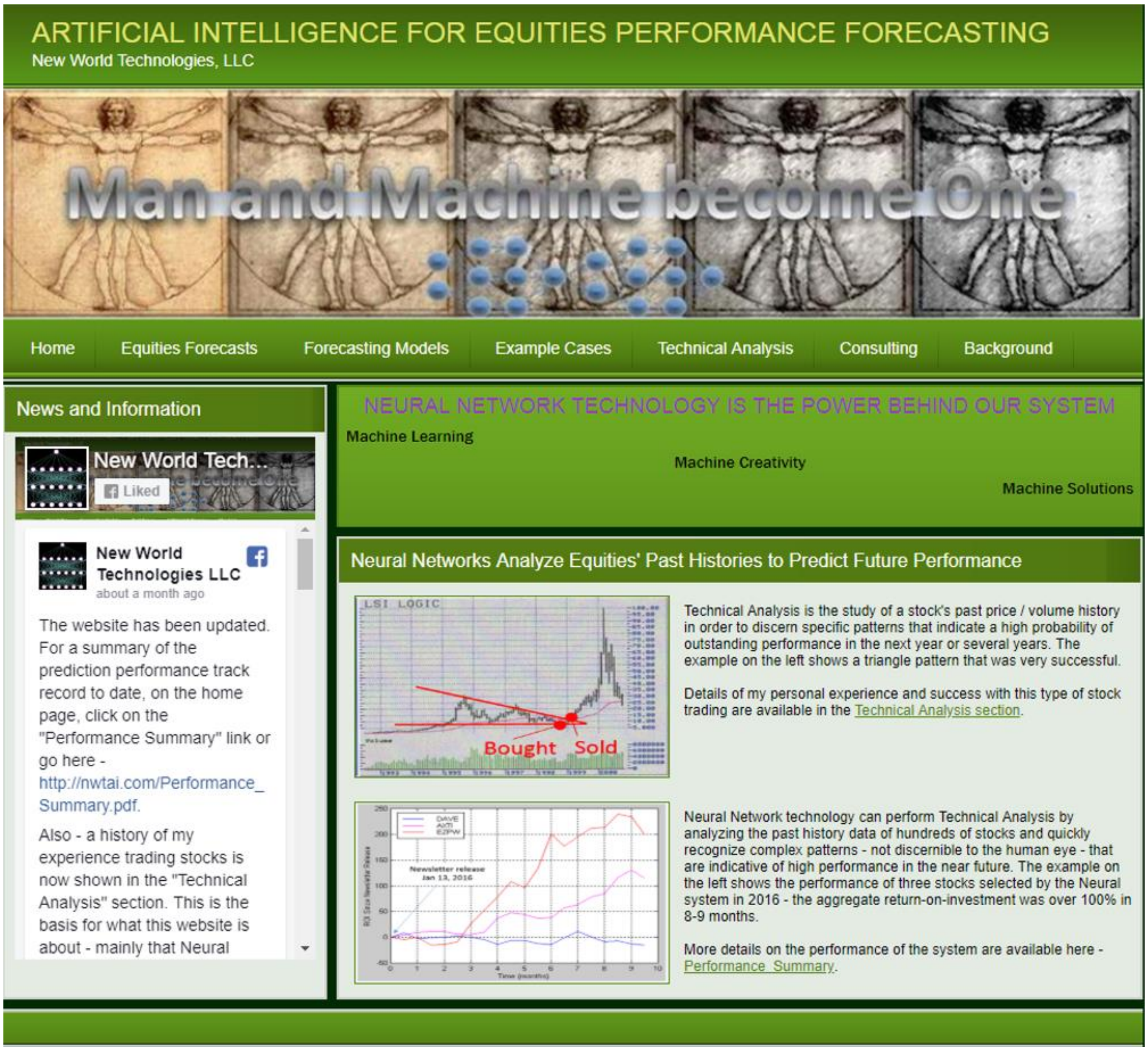


```
        self.quat.integrate(self.omega,dT)
        self.att_mtx = self.quat.rotation_matrix
        self.omega += dT * (
                self.inv_inertial.dot(M_vec -
                                      np.cross(self.omega,
                                               self.inertial.dot(self.omega))))
        # update time
        self.prev_time = self.time_s
        self.time_s += dT

    def control(self, pos_xyz_cmd, vel_xyz_cmd, fwd_cmd, errPosLim, dT):
        if self.time_s == self.prev_time:
            self.prev_time = self.time_s - dT
        # find the errors versus commanded quantities
        pos_xyz_err = self.pos_xyz_m - pos_xyz_cmd

        #-----------------------------------------------------------------------
        # Mike Fouche added this section.
        # Position Error Limiter
        if abs(errPosLim) > 0.0:
            if pos_xyz_err[0] <= -errPosLim:
                pos_xyz_err[0] = -errPosLim
            elif pos_xyz_err[0] >= errPosLim:
                pos_xyz_err[0] = errPosLim

            if pos_xyz_err[1] <= -errPosLim:
                pos_xyz_err[1] = -errPosLim
            elif pos_xyz_err[1] >= errPosLim:
                pos_xyz_err[1] = errPosLim

            if pos_xyz_err[2] <= -errPosLim:
                pos_xyz_err[2] = -errPosLim
            elif pos_xyz_err[2] >= errPosLim:
                pos_xyz_err[2] = errPosLim
        #-----------------------------------------------------------------------


        vel_xyz_err = self.vel_xyz_mps - vel_xyz_cmd
        # compute an acceleration error based on velocity error
        acc_xyz_err = self.K_acc * vel_xyz_err
        # this is the acceleration along gravity vector
        vert_xyz_err = self.mass_kg * self.acc_grav * self.zvec
        # this is the thrust vector
        # keep it real, and don't let it destroy lift
        max_thrust = 100.0
        down_vec_hover = vert_xyz_err
        down_vec_move = ( self.K_pos * pos_xyz_err +
                          self.K_vel * vel_xyz_err +
                          self.mass_kg * acc_xyz_err )
```

Software Development - HTML / JavaScript / CSS

Personal Business Website

I built my own website using HTML and CSS – www.nwtai.com, and maintain and update it when needed. The Facebook feed was an original idea that was implemented to make it easy to update the news section without having to update any of the HTML pages.



ARTIFICIAL INTELLIGENCE FOR EQUITIES PERFORMANCE FORECASTING

New World Technologies, LLC

News and Information

New World Tech...

New World Technologies LLC

about a month ago

The website has been updated. For a summary of the prediction performance track record to date, on the home page, click on the "Performance Summary" link or go here - http://nwtai.com/Performance_Summary.pdf.

Also - a history of my experience trading stocks is now shown in the "Technical Analysis" section. This is the basis for what this website is about - mainly that Neural

NEURAL NETWORK TECHNOLOGY IS THE POWER BEHIND OUR SYSTEM

Machine Learning

Machine Creativity

Machine Solutions

Neural Networks Analyze Equities' Past Histories to Predict Future Performance

Technical Analysis is the study of a stock's past price / volume history in order to discern specific patterns that indicate a high probability of outstanding performance in the next year or several years. The example on the left shows a triangle pattern that was very successful.

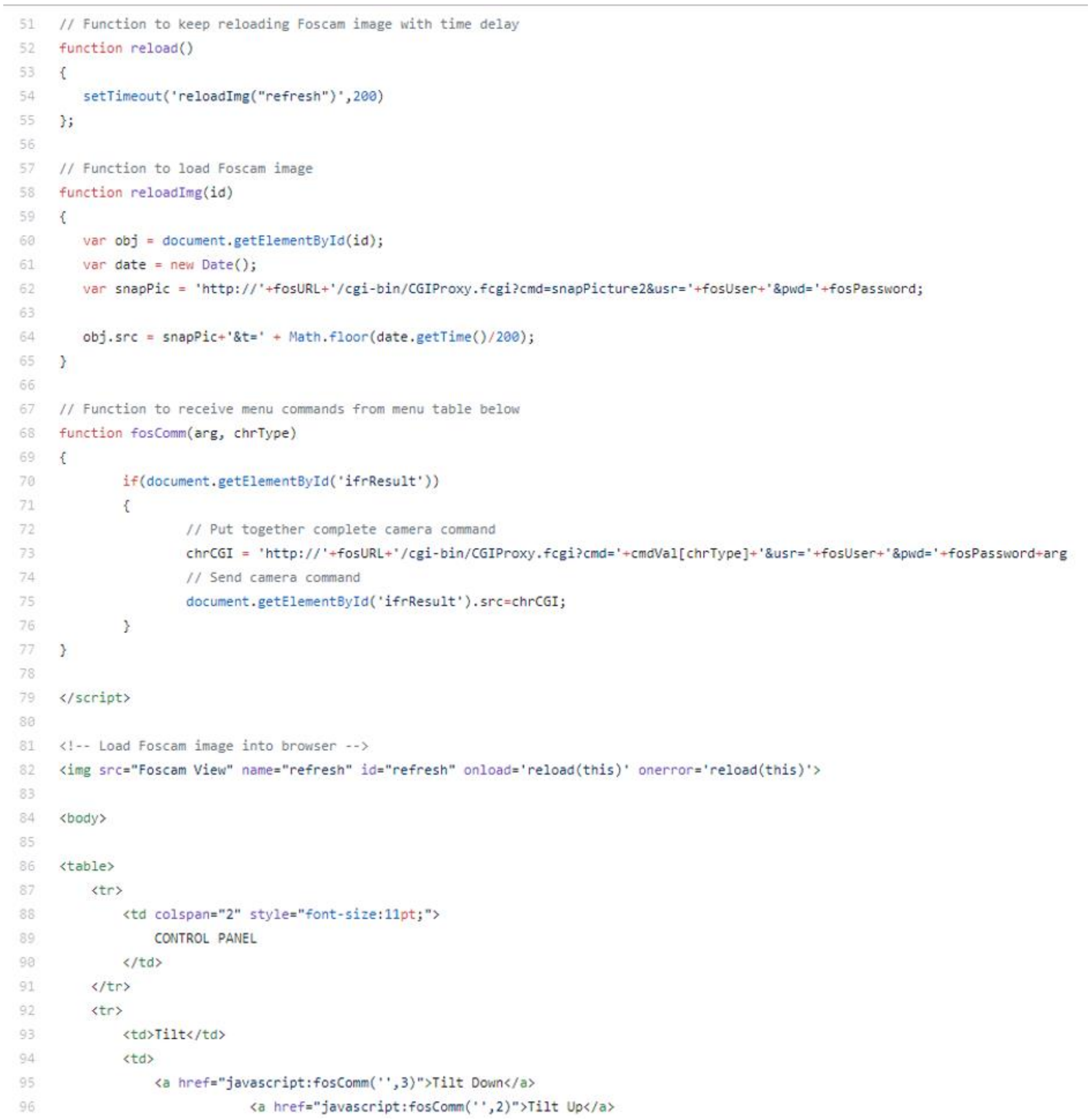
Details of my personal experience and success with this type of stock trading are available in the Technical Analysis section.

Neural Network technology can perform Technical Analysis by analyzing the past history data of hundreds of stocks and quickly recognize complex patterns - not discernible to the human eye - that are indicative of high performance in the near future. The example on the left shows the performance of three stocks selected by the Neural system in 2016 - the aggregate return-on-investment was over 100% in 8-9 months.

More details on the performance of the system are available here - Performance_Summary.

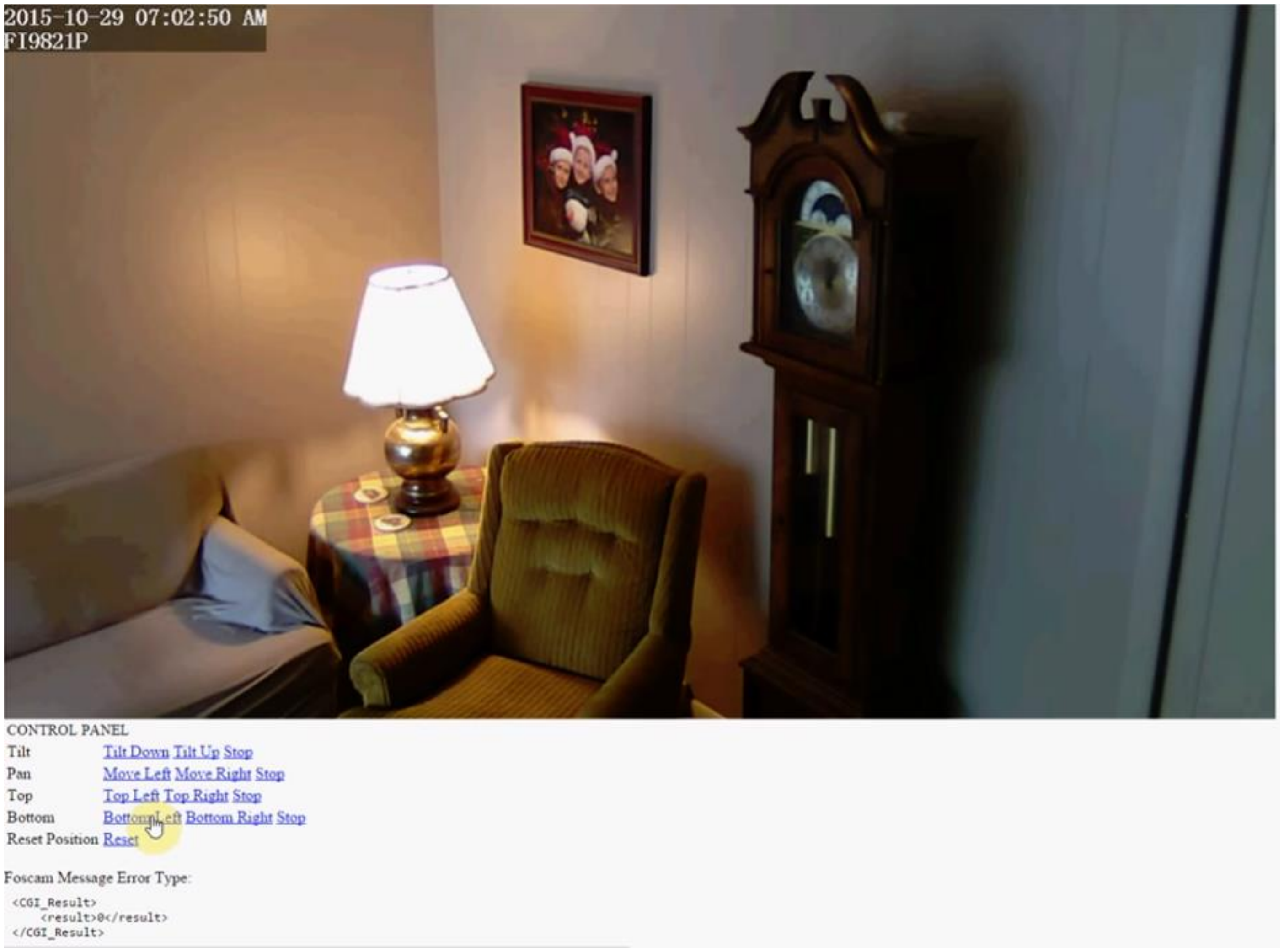
Web Page for Controlling Home Video Camera

I put together a basic HTML page with JavaScript functions to allow the user to control a Foscam video camera by simply clicking on command links from a web browser. A Java element was also part of the system. A portion of the HTML and JavaScript code is shown below. The full code base (just one file) can be obtained from https://github.com/mikesjavacode/Foscam-Videocam-JavascriptHtml-Controller.



```
// Function to keep reloading Foscam image with time delay
function reload()
{
    setTimeout('reloadImg("refresh")',200)
};

// Function to load Foscam image
function reloadImg(id)
{
    var obj = document.getElementById(id);
    var date = new Date();
    var snapPic = 'http://'+fosURL+'/cgi-bin/CGIProxy.fcgi?cmd=snapPicture2&usr='+fosUser+'&pwd='+fosPassword;

    obj.src = snapPic+'&t=' + Math.floor(date.getTime()/200);
}

// Function to receive menu commands from menu table below
function fosComm(arg, chrType)
{
        if(document.getElementById('ifrResult'))
        {
                // Put together complete camera command
                chrCGI = 'http://'+fosURL+'/cgi-bin/CGIProxy.fcgi?cmd='+cmdVal[chrType]+'&usr='+fosUser+'&pwd='+fosPassword+arg
                // Send camera command
                document.getElementById('ifrResult').src=chrCGI;
        }
}

</script>

<!-- Load Foscam image into browser -->
<img src="Foscam View" name="refresh" id="refresh" onload='reload(this)' onerror='reload(this)'>

<body>

<table>
    <tr>
        <td colspan="2" style="font-size:11pt;">
            CONTROL PANEL
        </td>
    </tr>
    <tr>
        <td>Tilt</td>
        <td>
            <a href="javascript:fosComm('',3)">Tilt Down</a>
                        <a href="javascript:fosComm('',2)">Tilt Up</a>
```

An example of what the user would see is shown below. The video from the camera is displayed in the upper portion of the browser while the control panel for controlling the camera is on the lower left side of the browser. There was an error message section that I left in place for debugging purposes. The video demonstration can be seen at https://youtu.be/KVRhQfLe9U8.



CONTROL PANEL

Tilt	Tilt Down Tilt Up Stop
Pan	Move Left Move Right Stop
Top	Top Left Top Right Stop
Bottom	Bottom Left Bottom Right Stop
Reset Position	Reset

Foscam Message Error Type:

```
<CGI_Result>
    <result>0</result>
</CGI_Result>
```

Software Development - Relational Databases

MySQL

I have used MySQL for a variety of projects, both at work and at home, including projects already discussed previously in this document.

The following is an example of an application that I did at home that used MySQL – https://youtu.be/NR0lmkqU4Qc, that uses MySQL to store and retrieve flight data. In this case the attitude profile can be replayed with data retrieved from the MySQL database.

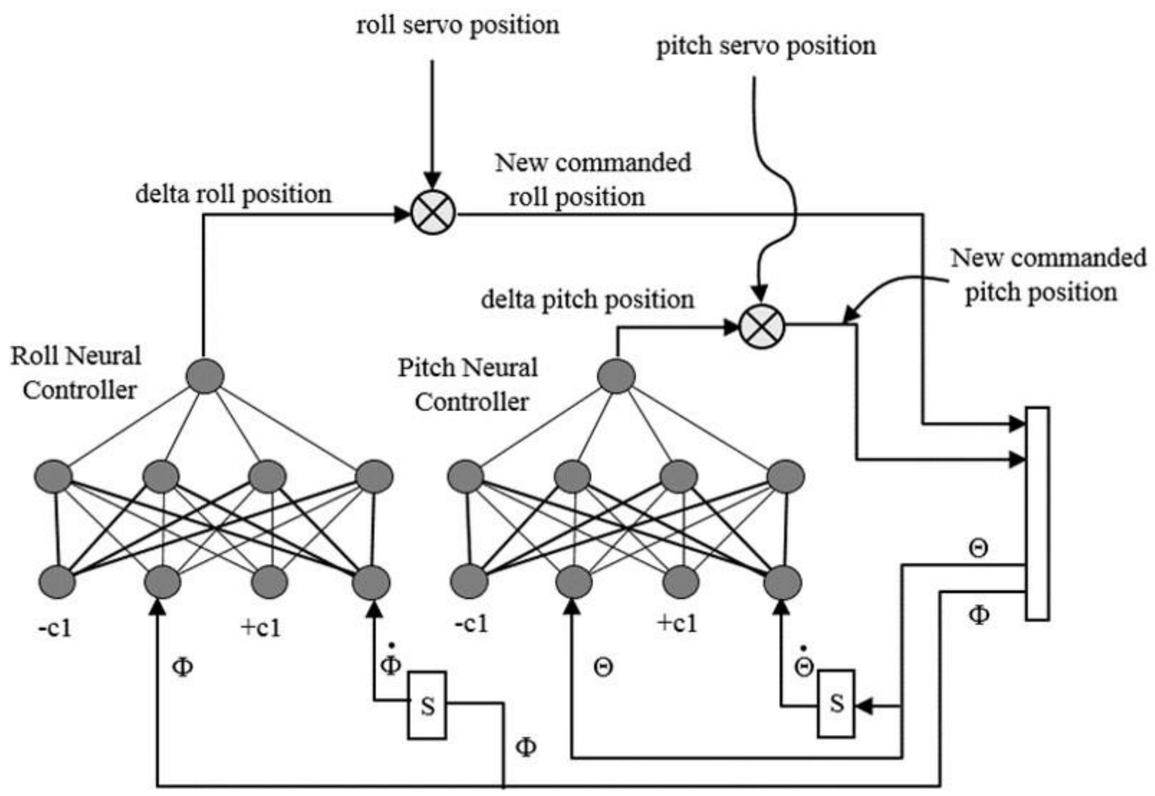
Artificial Intelligence (AI) – Neural Networks

I have developed artificial intelligence applications (specifically using feed-forward Neural Networks) off and on since the early 90s – these include (but are not limited to) autonomous flight control, image perception (object recognition / classification), equities (stocks) performance prediction, etc.

My tools include both the Matlab Levenberg-Marquardt (LM) optimization training algorithm and the Pyrenn LM optimization training algorithm (which can be obtained from here in Matlab and Python formats - https://pyrenn.readthedocs.io/en/latest/) and are used for the training of the Neural Networks.

Real-Time Autonomous Neural Network UAV Flight Control

I developed a Neural Network flight control system – inner-loop (attitude control) and outer-loop (velocity control), for a Radio-Control (RC) helicopter. A basic control system diagram is shown below.



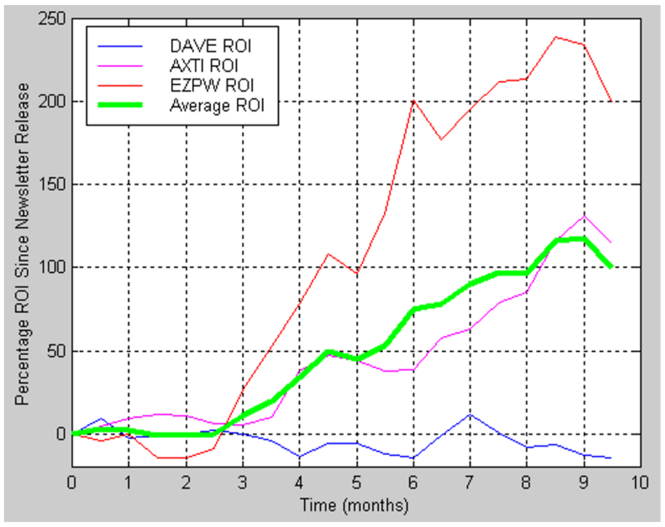
The full article (with several flight videos) is available to read here - https://mikescodeprojects.com/2020/01/05/ai-helicopter-control/.

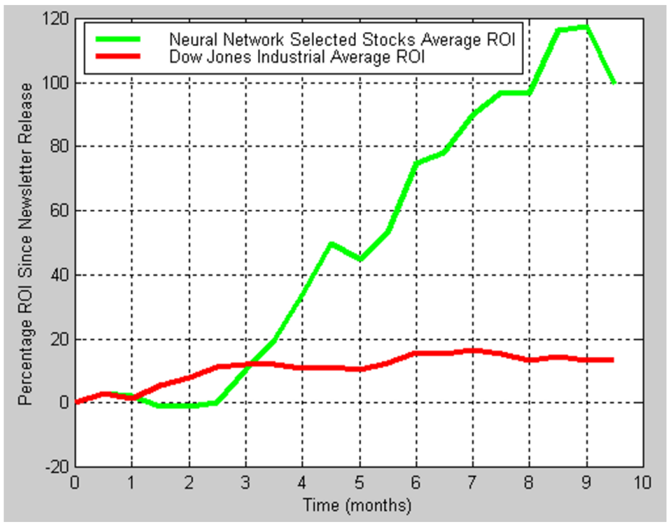
Neural Network Stock Price Predictor

I developed, on my own time, a Neural Network based equities analysis software tool code in Matlab. The algorithm analyzes previous stock history and predicts a percentage price increase or decrease for the following year (the conventional name for this approach is “Technical Analysis”).

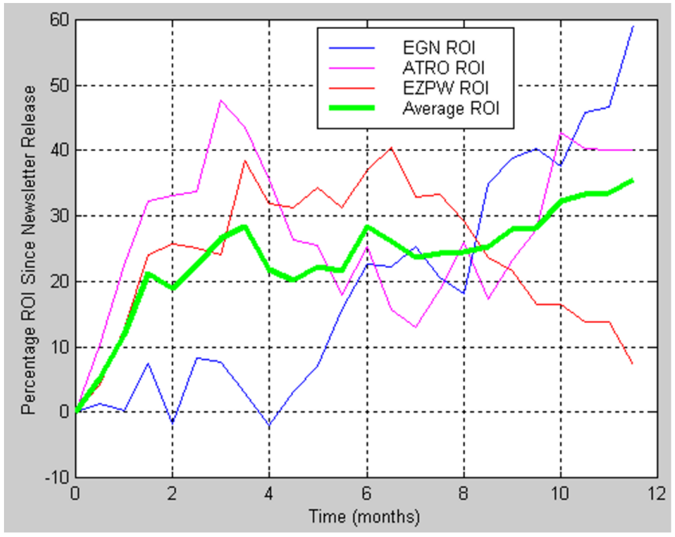
The performance thus far is promising and consistent - results may reviewed at www.nwtai.com.

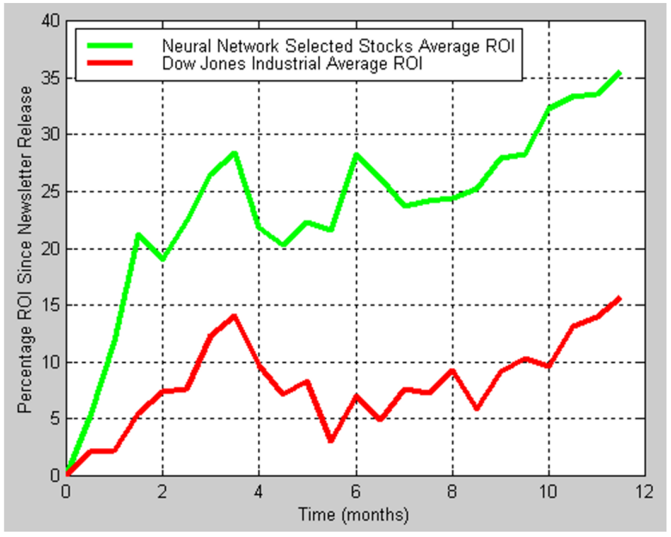
The average Return-on-Investment (ROI) for the three stocks – shown below, on one of the newsletter predictions (from the website) was over 100% in a 9 month whereas the Dow Jones Industrial Average had an ROI of about 17% for the same time period.





The stocks selected in another newsletter (shown below) had an average ROI of approximately 35% whereas the Dow Jones Industrial average had an ROI of 15% for the same time period.





Neural Network Video Tutorials

The following are YouTube videos that I put together a couple of years ago for beginners that are interested in the technology:

1. Neural Network Application Tutorial - https://youtu.be/dnazn9xnTCA

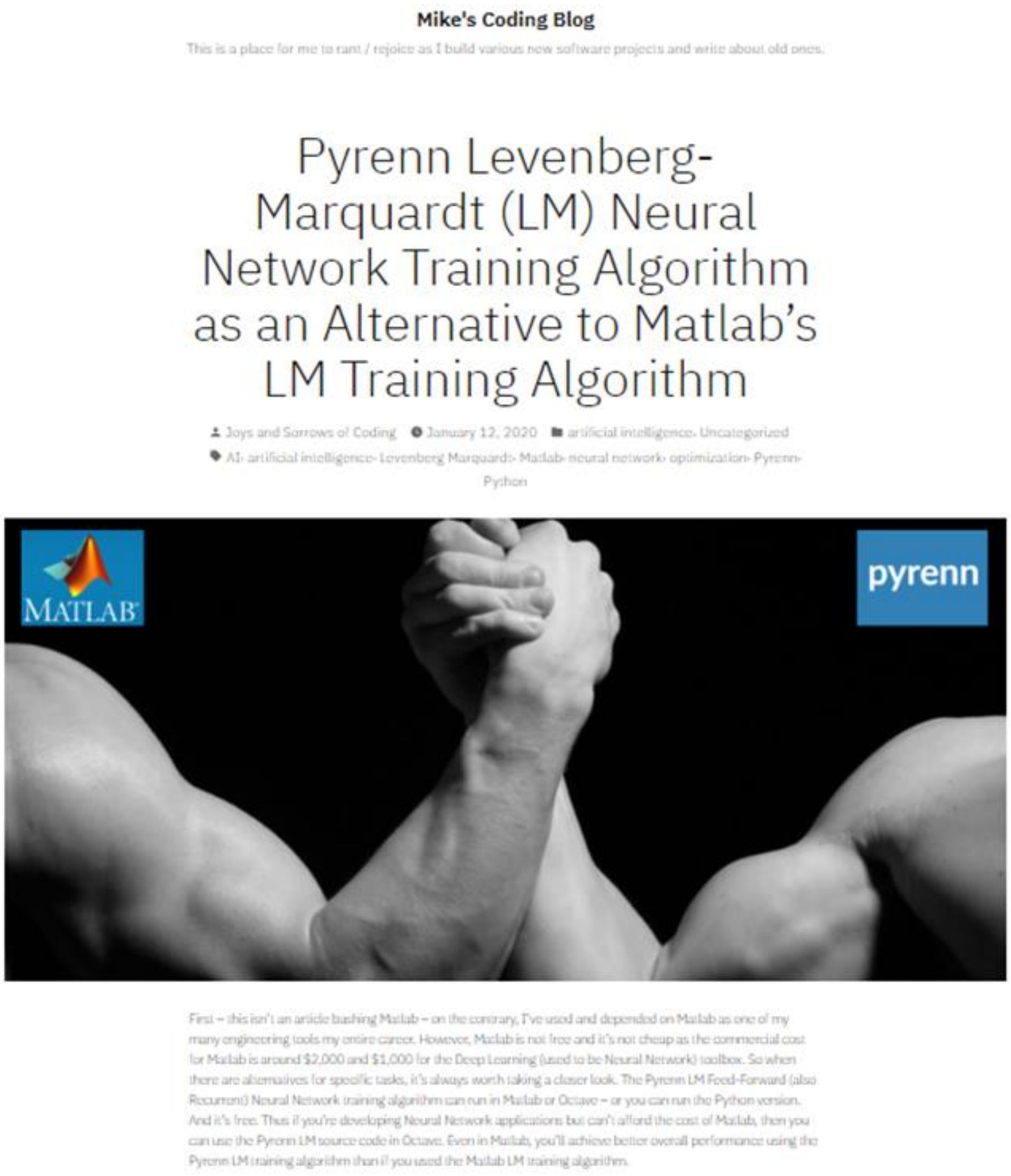
2. How to run the Matlab software for this tutorial - https://youtu.be/Sis-xTcP76w

3. Video of actually running the Matlab software - https://youtu.be/o_HM1J27WOg

4. Example of how Neural Networks far outperform their training sets - https://youtu.be/VpdkIN4kAFo

Comparison of Neural Network Training Algorithms from Personal Tech Blog

I wrote an article on my personal tech blog - https://mikescodeprojects.com/2020/01/12/pyrenn-vs-matlab/, that discussed, in detail, the comparison of Matlab’s Neural Network Levenberg-Marquardt training algorithm vs Pyrenn’s Neural Network Levenberg-Marquardt training algorithm.



Mike's Coding Blog

This is a place for me to rant / rejoice as I build various new software projects and write about old ones.

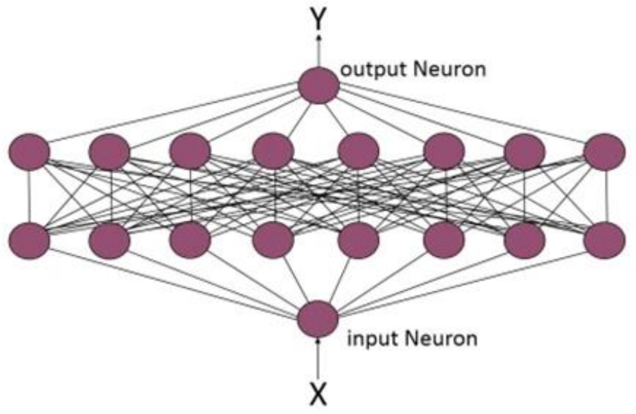
Pyrenn Levenberg-Marquardt (LM) Neural Network Training Algorithm as an Alternative to Matlab’s LM Training Algorithm

Joys and Sorrows of Coding January 12, 2020 artificial intelligence, Uncategorized

AI, artificial intelligence, Levenberg Marquardt, Matlab, neural network, optimization, Pyrenn, Python

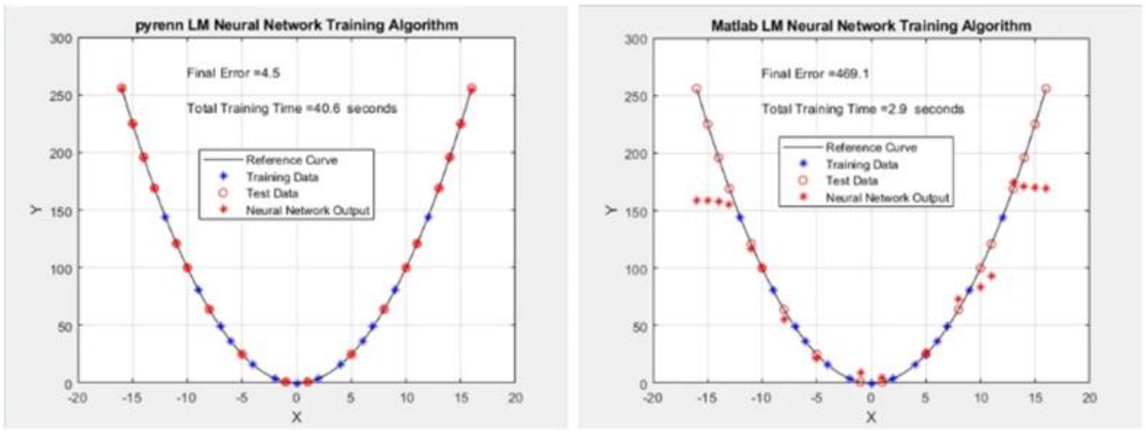
First – this isn’t an article bashing Matlab – on the contrary, I’ve used and depended on Matlab as one of my many engineering tools my entire career. However, Matlab is not free and it’s not cheap as the commercial cost for Matlab is around $2,000 and $1,000 for the Deep Learning (used to be Neural Network) toolbox. So when there are alternatives for specific tasks, it’s always worth taking a closer look. The Pyrenn LM Feed-Forward (also Recurrent) Neural Network training algorithm can run in Matlab or Octave – or you can run the Python version. And it’s free. Thus if you’re developing Neural Network applications but can’t afford the cost of Matlab, then you can use the Pyrenn LM source code in Octave. Even in Matlab, you’ll achieve better overall performance using the Pyrenn LM training algorithm than if you used the Matlab LM training algorithm.

The screenshot below is lifted from a section in the article that reviews the performance of the two LM training algorithms.



Neural Network Architecture – 2 Middle Layers with 8 Neurons

The performance of the Matlab LM-trained Neural Networks continued to deteriorate while the Pyrenn LM-trained Neural Networks maintained good performance. Each of the two plots represent the best performing Neural Network, out of a total of 10 – that is, the best one out of 10 generated by the Pyrenn LM algorithm, and the best one out of 10 generated by the Matlab LM algorithm.



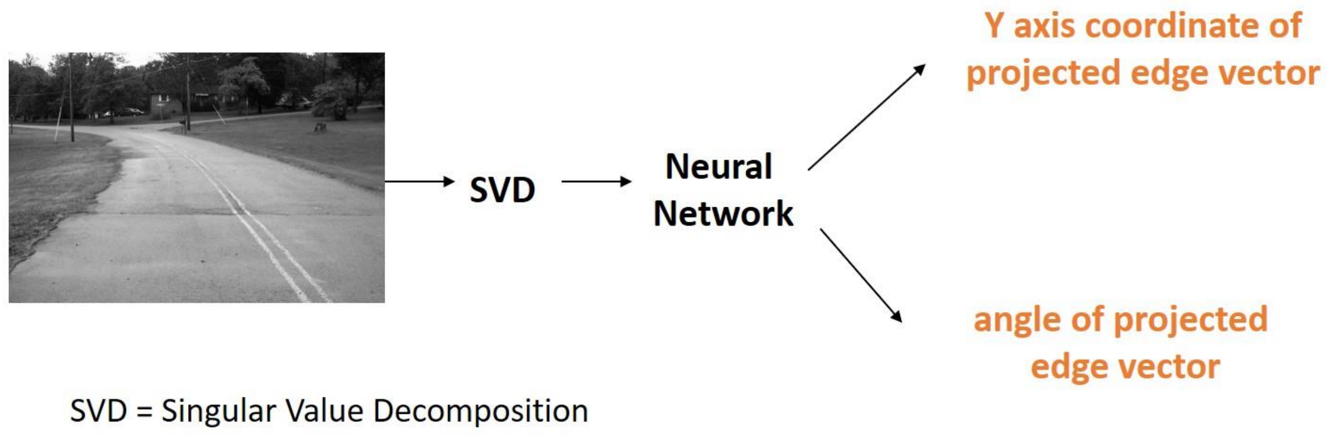
Comparison of Performance between Pyrenn and Matlab

As before, there was a significant difference between the performances of the Neural Networks trained by the Pyrenn LM algorithm and those trained by the Matlab LM algorithm. Note that the errors were sorted from lowest to highest.

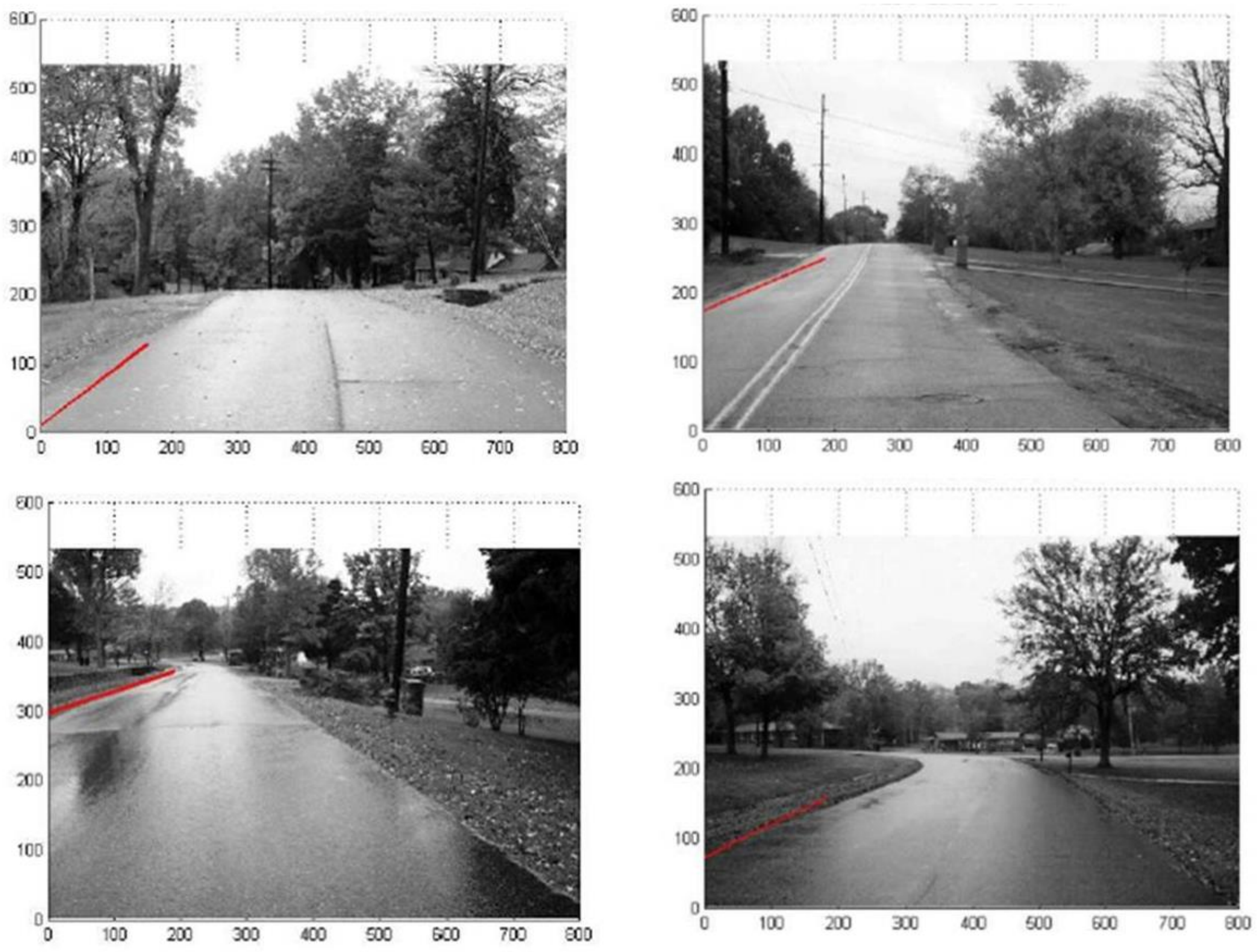
Neural Network Road Edge Detection

This project's focus was to see if a Neural Network could "view" an image of a road and determine the location of the left edge of the road, for the purposes of autonomous road navigation.

The basic process was the following: 1) reduce the resolution, 2) use Singular Value Decomposition (SVD) to extract the key features, 3) train a Neural Network with the primary singular values as inputs, and 4) generate the following two outputs: a) the Y-axis coordinate of the projected edge vector, and b) the angle of the projected vector.


SVD
Neural Network
Y axis coordinate of projected edge vector
angle of projected edge vector
SVD = Singular Value Decomposition

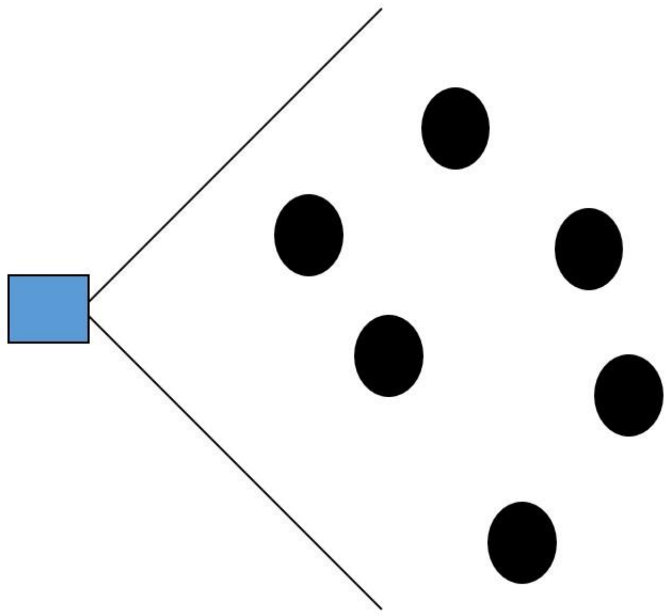
The Neural Network system was reasonably able to discern the left road edges in different images, as shown in the images below. The red line represents the Neural Network's estimate of the left road edge.



Neural Network Object Ranging

This project’s focus was to see if a Neural Network could “view” an image of the same-size balls (but at different distances and angles from the camera centerline) and determine the range and angle of the closest ball (ignoring the other balls in the camera’s field-of-view).

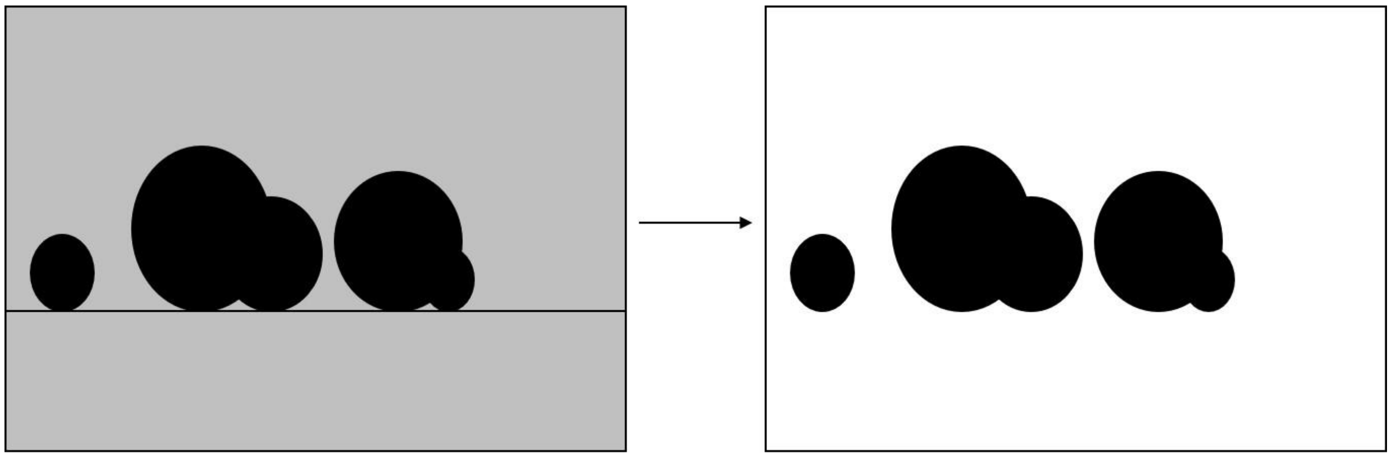
Objective: using a single video camera (256x256 gray-scale), looking at balls (same size) at various distances, determine the range/angle of the closest ball



Perspective is looking down – balls are sitting a floor

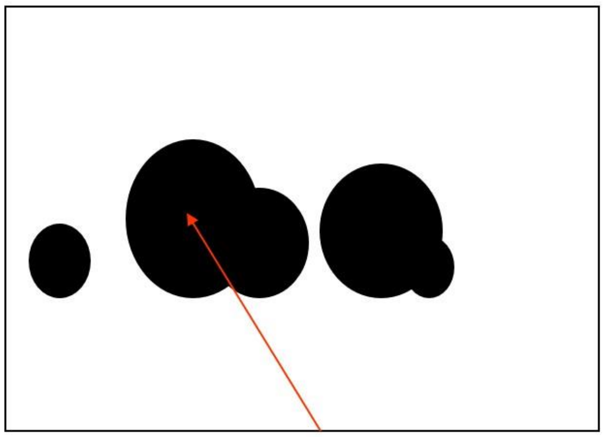
The basic process was the following: 1) threshold the image, 2) reduce the resolution, 3) use Singular Value Decomposition (SVD) to extract the key features, 4) train a Neural Network with the primary singular values as inputs, and the range and angle of the closest ball in the image, as the outputs.

The Neural Network system very accurately was able to determine the range and angle of the closest ball no matter how the balls were arranged.



Perform thresholding on primary image

- Take snapshot of scene
- Threshold image
- Reduce resolution
- Process through neural network
- Neural network outputs range and angle of ball that is closest to the video camera

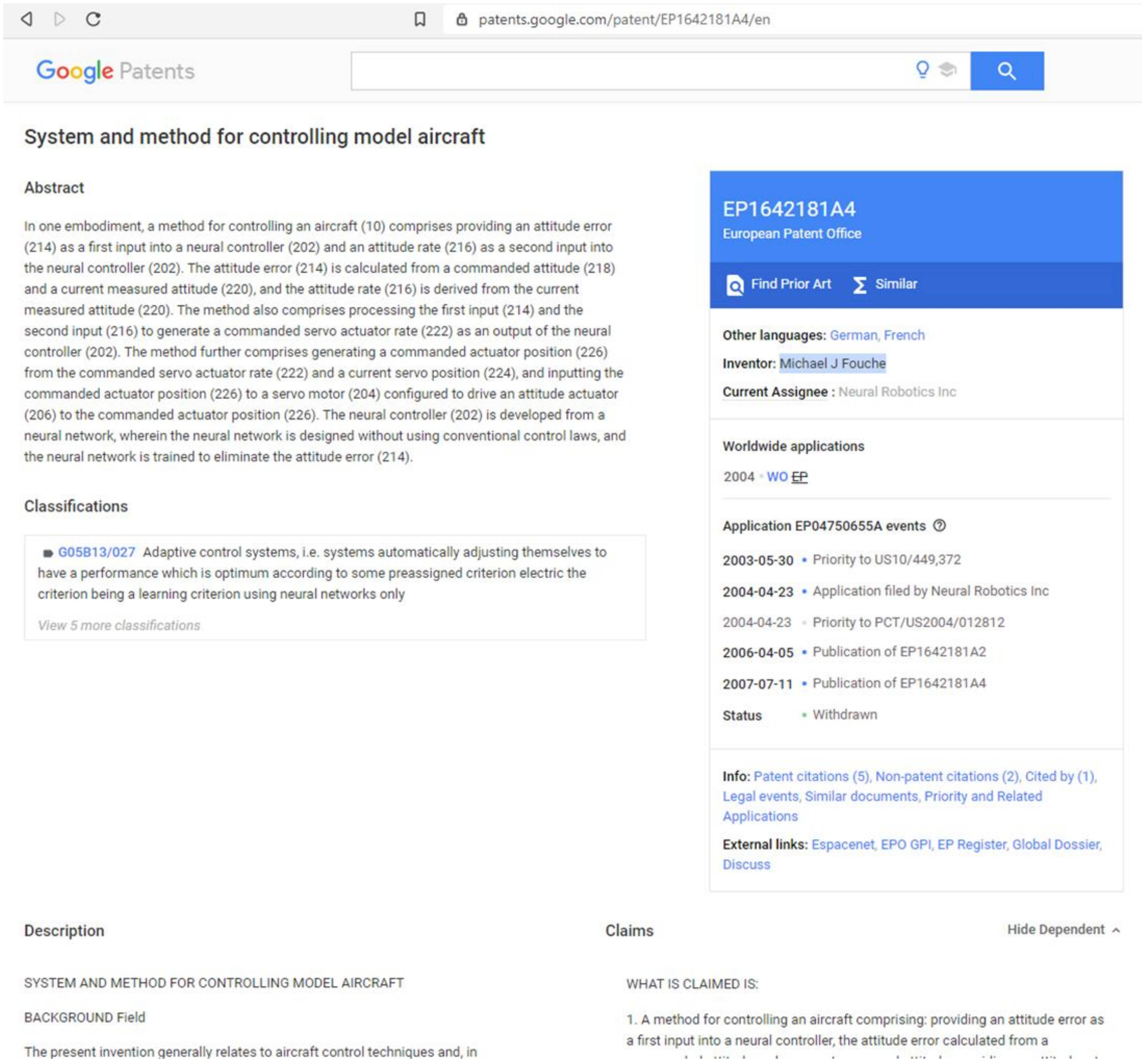


Range = X inches
Angel = Theta degrees
(measured from center)

Patent – "System and Method for Controlling Model Aircraft"

I filed for this patent in late 2002 and it was granted by the U.S. Patent Office in 2004. The patent description can be accessed at https://www.google.com/patents/EP1642181A4?cl=en.

The patent covers the approach for building a Neural Network attitude control system for a model aircraft.



System and method for controlling model aircraft

Abstract

In one embodiment, a method for controlling an aircraft (10) comprises providing an attitude error (214) as a first input into a neural controller (202) and an attitude rate (216) as a second input into the neural controller (202). The attitude error (214) is calculated from a commanded attitude (218) and a current measured attitude (220), and the attitude rate (216) is derived from the current measured attitude (220). The method also comprises processing the first input (214) and the second input (216) to generate a commanded servo actuator rate (222) as an output of the neural controller (202). The method further comprises generating a commanded actuator position (226) from the commanded servo actuator rate (222) and a current servo position (224), and inputting the commanded actuator position (226) to a servo motor (204) configured to drive an attitude actuator (206) to the commanded actuator position (226). The neural controller (202) is developed from a neural network, wherein the neural network is designed without using conventional control laws, and the neural network is trained to eliminate the attitude error (214).

Classifications

G05B13/027 Adaptive control systems, i.e. systems automatically adjusting themselves to have a performance which is optimum according to some preassigned criterion electric the criterion being a learning criterion using neural networks only

View 5 more classifications

EP1642181A4
European Patent Office

Find Prior Art Similar

Other languages: German, French

Inventor: Michael J Fouche

Current Assignee : Neural Robotics Inc

Worldwide applications

2004 WO EP

Application EP04750655A events

2003-05-30	Priority to US10/449,372
2004-04-23	Application filed by Neural Robotics Inc
2004-04-23	Priority to PCT/US2004/012812
2006-04-05	Publication of EP1642181A2
2007-07-11	Publication of EP1642181A4
Status	Withdrawn

Info: Patent citations (5), Non-patent citations (2), Cited by (1), Legal events, Similar documents, Priority and Related Applications

External links: Espacenet, EPO GPI, EP Register, Global Dossier, Discuss

Description

SYSTEM AND METHOD FOR CONTROLLING MODEL AIRCRAFT

BACKGROUND Field

The present invention generally relates to aircraft control techniques and, in

Claims

Hide Dependent

WHAT IS CLAIMED IS:

1. A method for controlling an aircraft comprising: providing an attitude error as a first input into a neural controller, the attitude error calculated from a

Hardware – Avionics for Autonomous Flight Systems

While I'm not trained as an electrical engineer (my degree is Aerospace Engineering), I've spent a fair amount of time integrating avionics hardware systems over the years. These efforts entailed gaining a good understanding of the basic design process and requirements (sometimes this knowledge was learned in a painful manner).

There are several requirements when putting together an avionics box (as part of the overall avionics system). The following are not all-inclusive but are key requirements:

1. Temperature management
2. Elimination of RF interference between avionics boards
3. Assurance of proper cable stress relief
4. Designing accessibility to internal components
5. Power management
6. Robust component integration framework

Temperature Management: Internal temperatures need to be controlled such that the maximum temperature limits of the components are not exceeded. Temperature management can be passive with component and / or component heat sinks or with filtered air ducts in the avionics box to allow free air passage. Active temperature management can be accomplished with the use of internal fans which force outside air to pass through the avionics box.

RF Interference: This issue doesn't arise many times until the components are already integrated. It's critical to try and mock up the component assembly in order to test for RF interference problems ahead of avionics box design.

Cable Stress Relief: This aspect of careful avionics design is one that, if overlooked, will manifest in the long term with sudden intermittent loss of signal, power outages, etc.

Accessibility to Internal Components: What this means is that there is the ability to access the component electronically to perform unit testing and debugging. This access can be either through the main system cabling (if designed that way) or via an external communications port (TCP, RS-232, etc.). If this is not designed into the avionics system, then whenever troubleshooting takes place, components will have to be removed and tested individually.

Power Management: It is critical that all power coming from outside the avionics box is regulated – each component must have access to the correct voltage and required current (just because a power board provides 12 volts doesn't mean that it has an unlimited amount of current). Thus, if there is a power surge, the power regulation system will continue to provide steady-state voltage and current levels to each component.

In addition, it is helpful if power monitoring is available for each component. Thus, if there is a failure during testing (or in normal operation), the voltage and current levels would have been logged and can be analyzed. Note that not all electrical components simply shut down if the supplied power drops below the required level. In some cases, the component continues to operate but the circuits become unstable yet output is still provided – even though it is incorrect. Thus an "under power" condition could explain faulty data in some situations.

Component Integration Framework: There are several aspects to this task: 1) actual mounting of each component and 2) the ability of the mounting system to withstand shock and vibration (for the full spectrum of frequencies for which the system may experience in its application). Even little things like how the avionics board is mounted to the standoffs can be critical – in some cases, metal-on-metal can induce RF noise during vibration and thus plastic-to-metal is the only solution.

Avionics for DARPA-funded Prototype Autonomous Powered-Parafoil Aircraft

I was tasked to lead the effort to put together a system to perform flight control for an autonomous powered parafoil. My responsibilities included: 1) designing the avionics suite (generate the requirements, put together the hardware diagrams including cable assemblies, etc.), 2) design and code the ground control station software, and 3) design and code the flight control software.

Based on previous experience, I decided to go with a PC-104 stack because of the reliability and ruggedness of the design (weight was not a mitigating issue). The final prototype system is shown below in Figure 1.

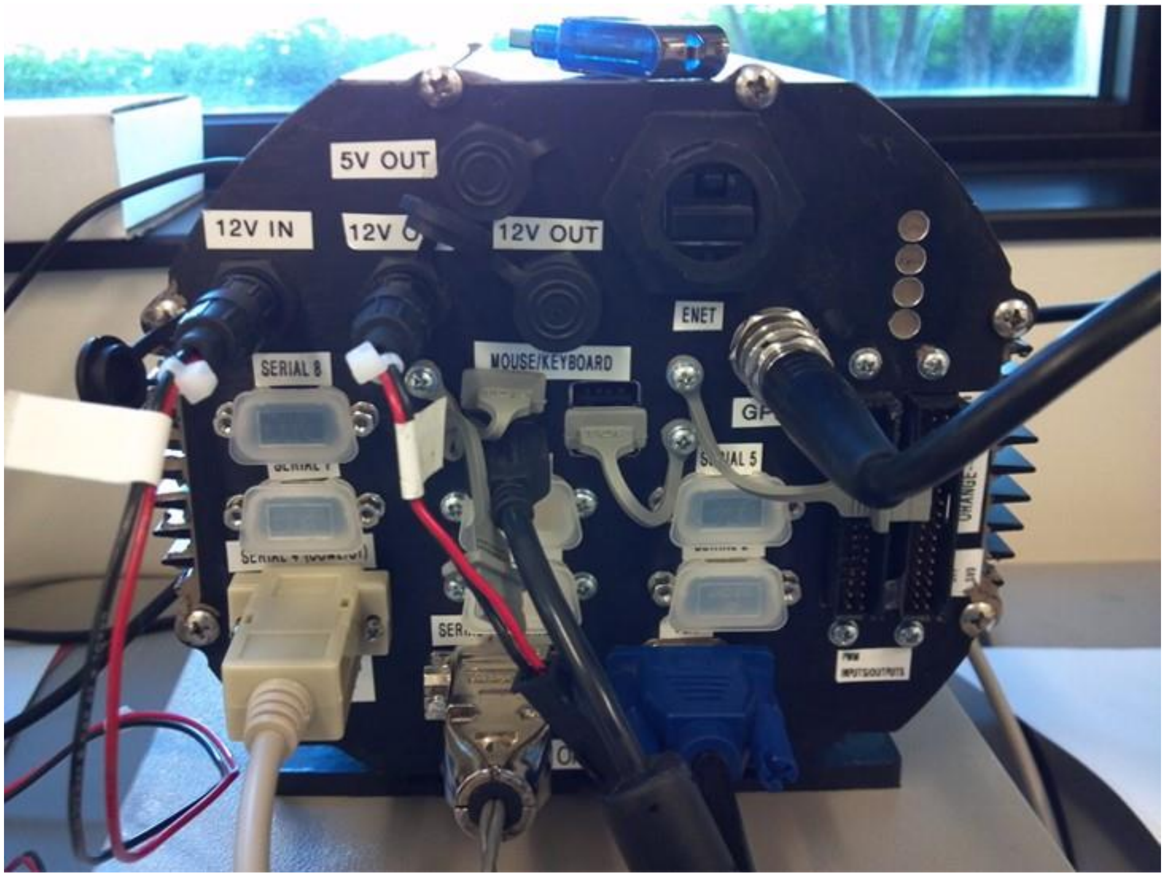


Figure 1 – Core Avionics System

The PC-104 stack (CPU board, power board, VGA board, etc.), GPS receiver board, and IO boards were integrated inside the black avionics box. Additional avionics components were distributed throughout the vehicle (GPS antenna, communications radio modem, and Attitude and Heading Reference System – referred to as AHRS) as shown below in Figure 2.

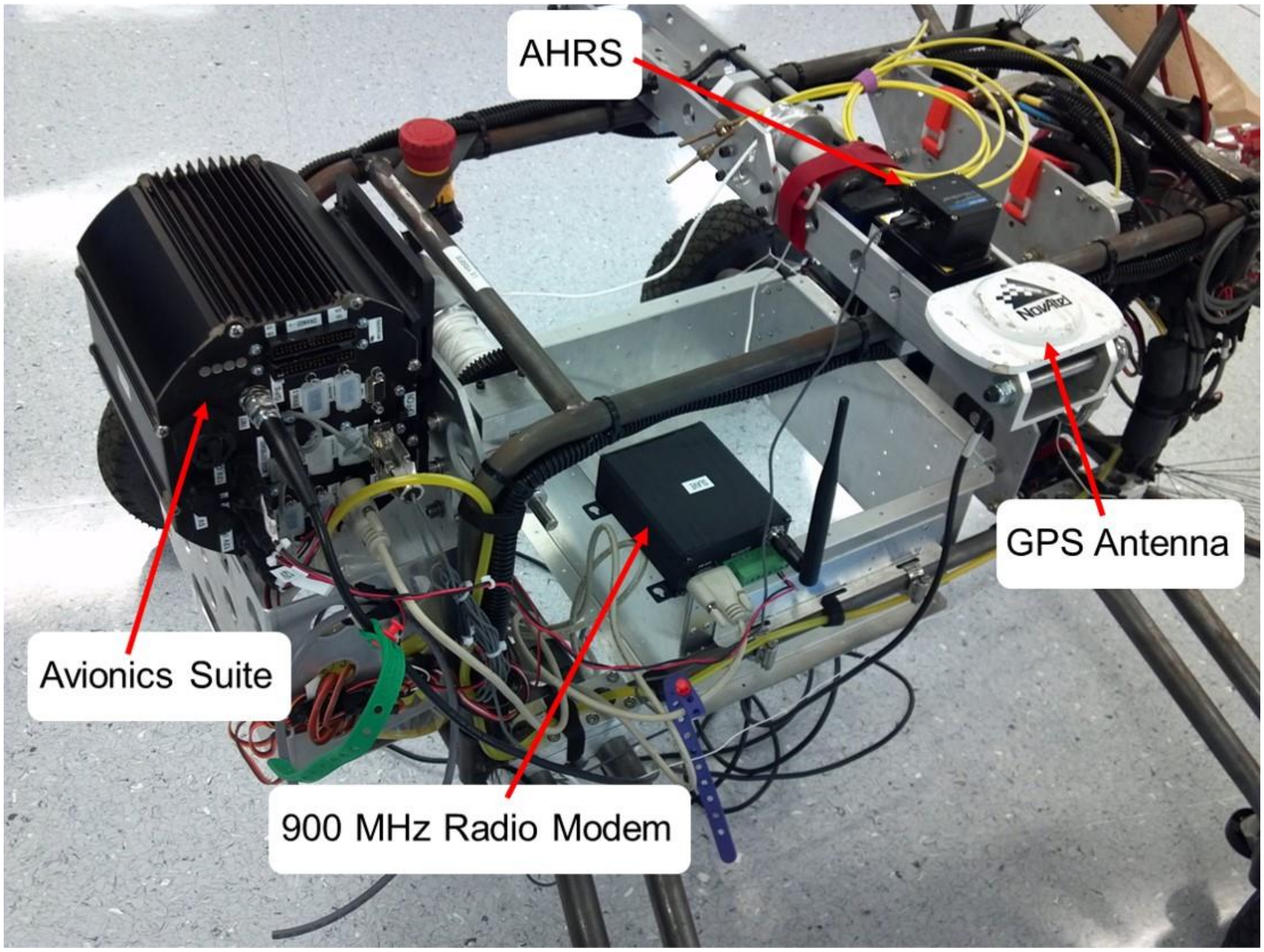


Figure 2 – Autonomous Powered-Parafoil Avionics Layout

The autonomous powered-parafoil aircraft, with integrated avionics, is shown below in Figure 3.

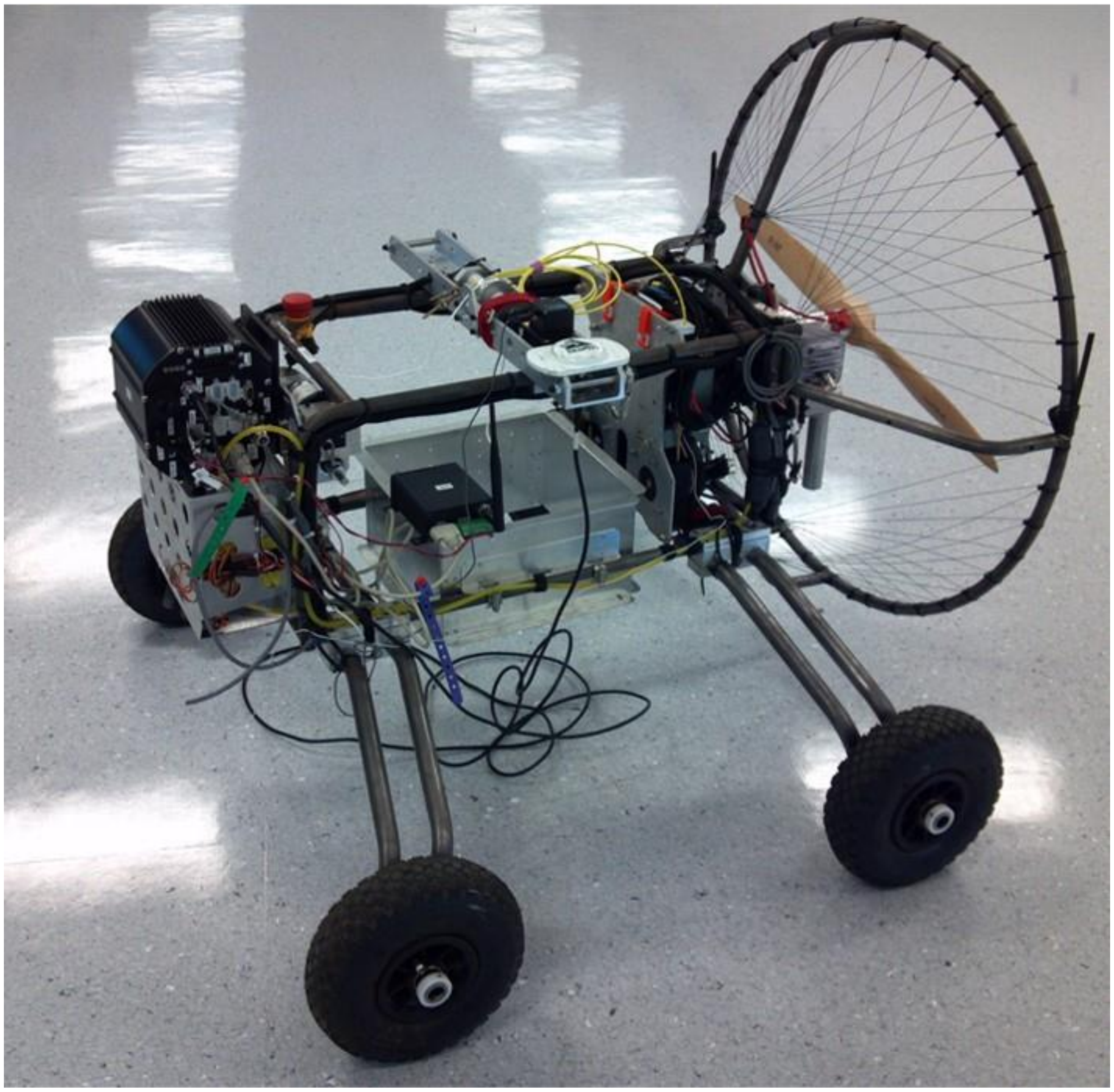


Figure 3 – Autonomous Powered-Parafoil Aircraft with Integrated Avionics

Avionics for Autonomous Helicopter Neural Network Flight Control System

For an autonomous helicopter system, I had to build basic diagrams of the avionics components in order to understand the power requirements and for later providing detailed schematics to the technicians that would perform the cable assemblies.

Not all of the avionics components operated with the same voltage – in this case, the voltage requirements were 5 volts, 6 volts, and 12 volts. The main power board is shown below on the top right in Figure 1. The actuator controller board is shown below in the top left. The GPS board is shown below on the bottom left.

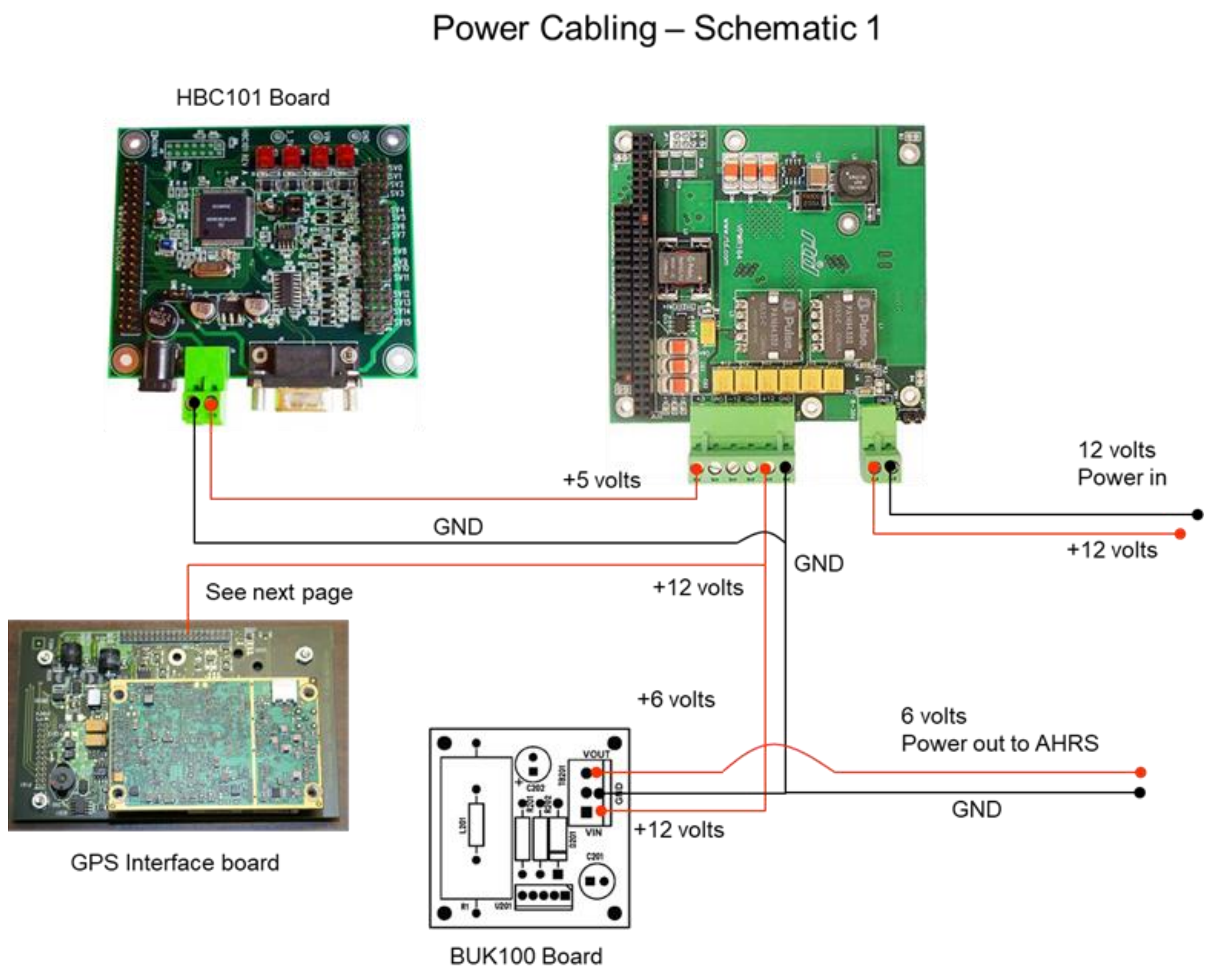


Figure 1 – Draft Partial Avionics Components Power Schematic

While the actuator control board's primary function was to send commands to the actuators, a smaller function was to power three LED lights that were attached to the external part of the avionics box. These LED lights were used to provide information to the operator regarding the helicopter's state (e.g., "ready to fly", "problem", etc.). The basic layout is shown below in Figure 2.

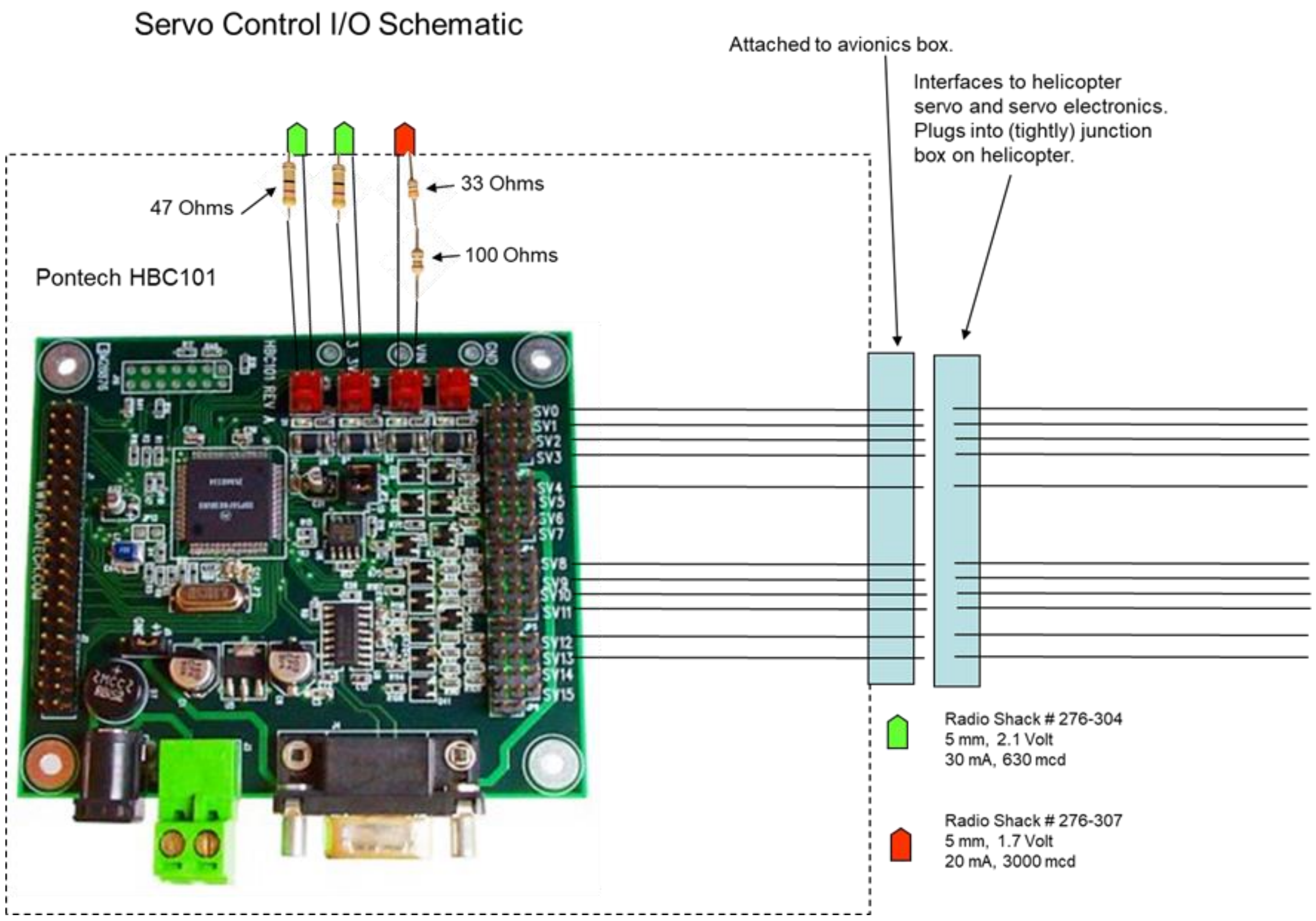


Figure 2 – Component Schematic for Actuator Control Board

Some first-stage diagrams contained both basic power and communications diagrams – as shown below in Figure 3.

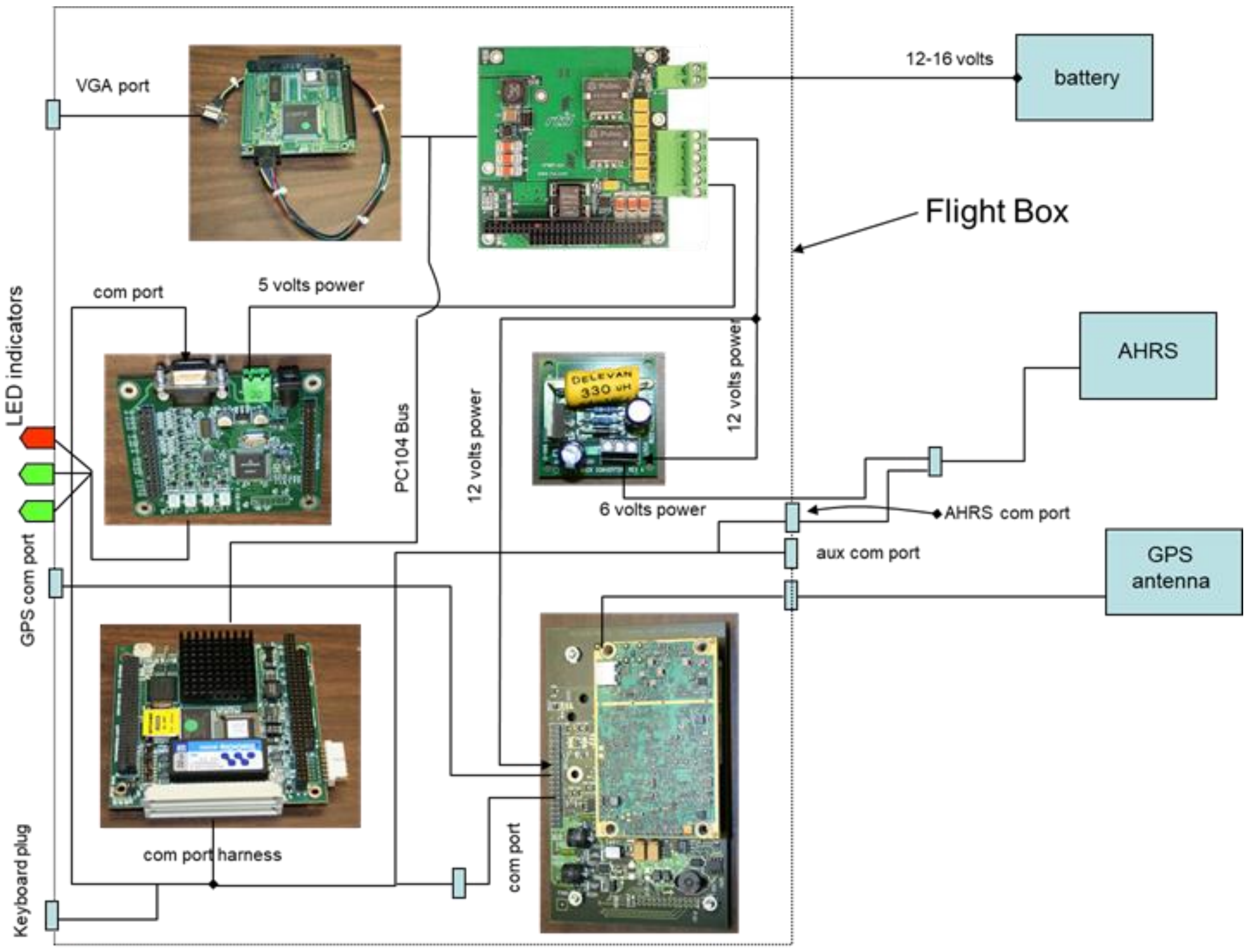


Figure 3 – Early-Stage Avionics Diagram – Power and Connectivity

Figure 4 shows a completed early-stage avionics box for an unmanned autonomous helicopter (an RC helicopter converted to autonomous capability with the avionics system).

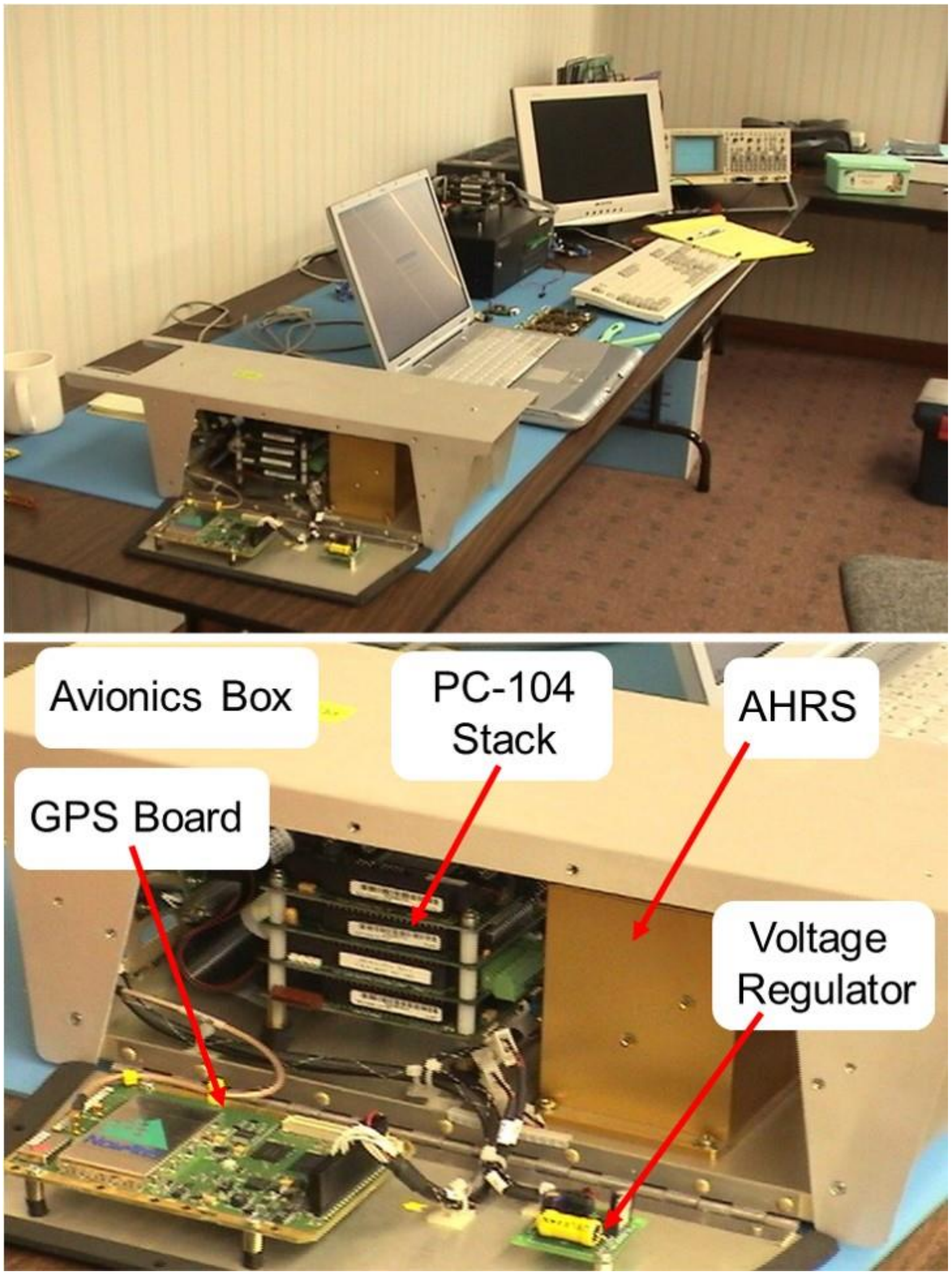


Figure 4 – Early-Stage Avionics System

For military customers, more rigorous requirements meant that the avionics box design had to follow certain military standards for assembly and integration. One of those requirements was an external military-grade connector system – as shown below in Figure 5. This system was assembled / integrated by another company that was a provider to the military for electronic assembly – thus they had the expertise of implementing the required military specifications. We provided the components, wiring diagrams, avionics component diagrams, etc.

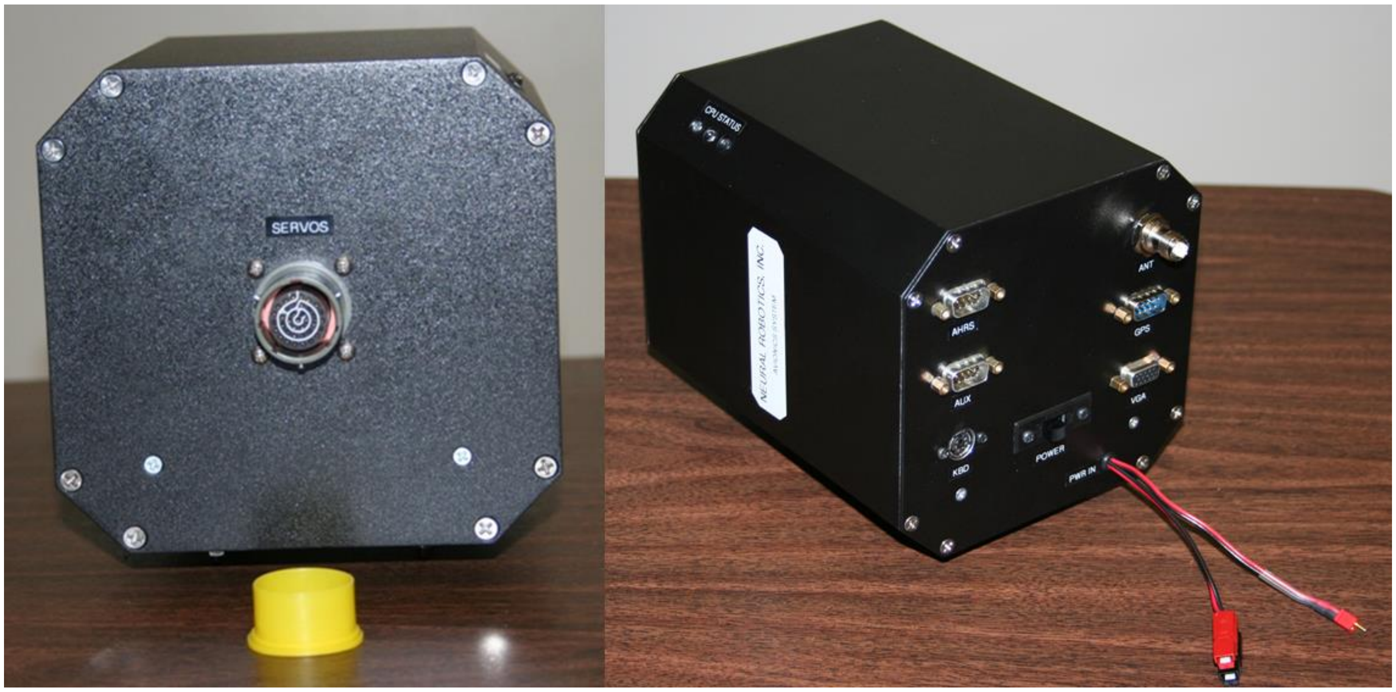


Figure 5 – Ruggedized Avionics for Military Customers

Early Phase Avionics for Neural Network Autonomous Control System Development

During this period, I was building a prototype autonomous helicopter flight control system to demonstrate the ability to hover using a Neural Network flight controller. Given that I was using my own money and working on my own time, I was careful with expenditures (although quality – and costly – components were used where needed).

Figure 1 shows the basic “avionics” for this demonstrator helicopter system.

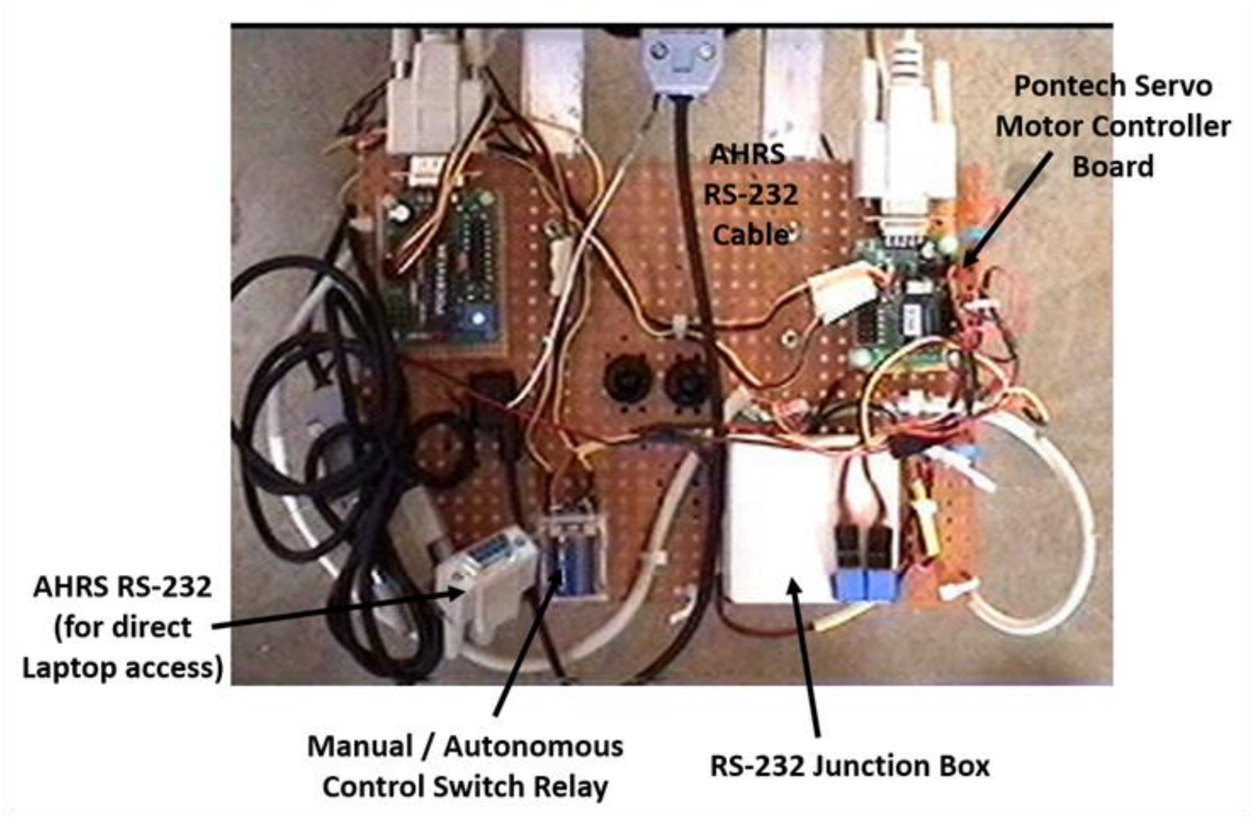

Pontech Servo Motor Controller Board
AHRS RS-232 Cable
AHRS RS-232 (for direct Laptop access)
Manual / Autonomous Control Switch Relay
RS-232 Junction Box

Figure 1 – Garage-style Avionics for Autonomous Helicopter Demonstration System

The most expensive component in this system was the Crossbow Attitude and Heading Reference System ($4,000 at the time), as shown below in Figure 2.

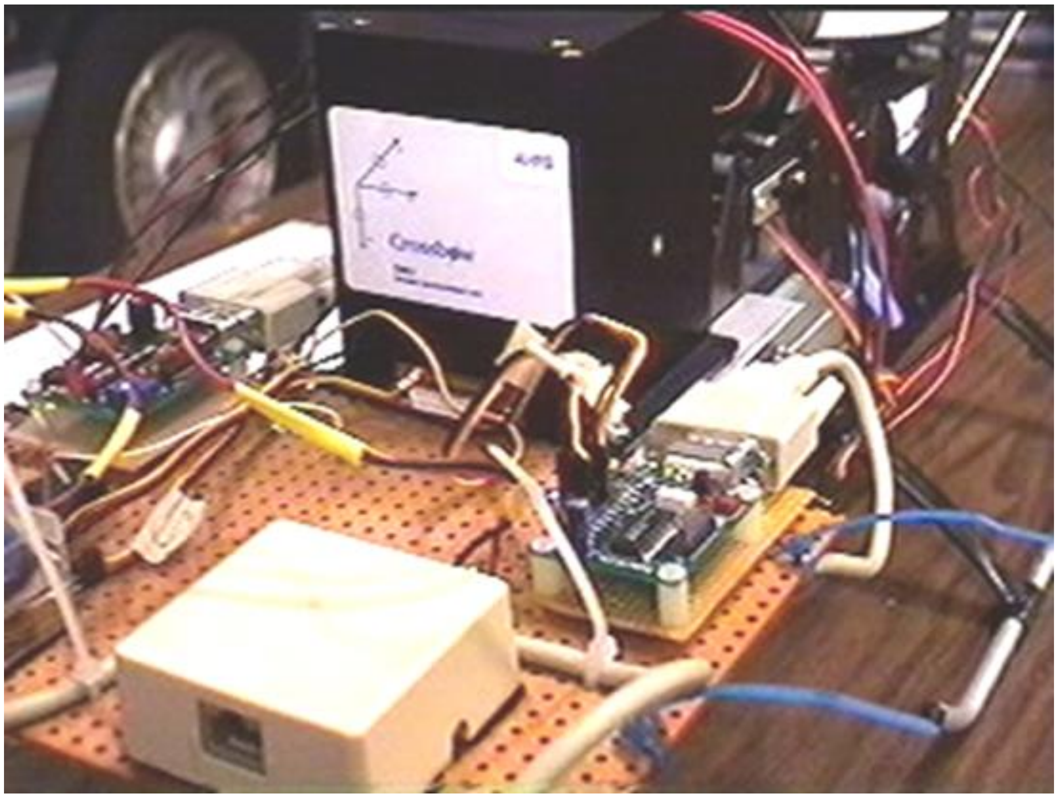
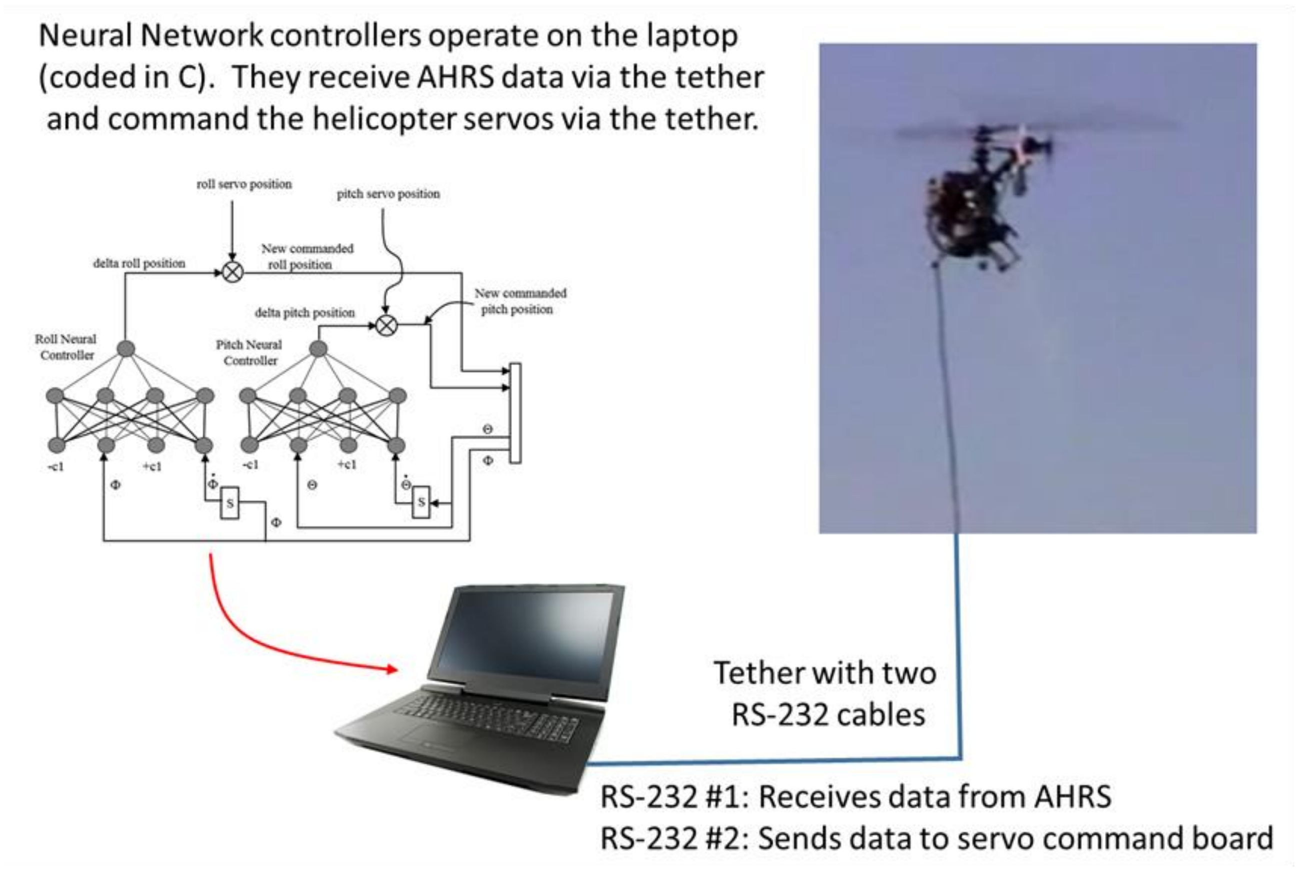


Figure 2 – “Garage-style” Avionics with Crossbow AHRS

The actual flight computer was a laptop which communicated with the helicopter onboard-avionics via a 50-foot RS-232 tether – as shown below in Figure 3. The software was coded in C, running in DOS 6.22.



Neural Network controllers operate on the laptop (coded in C). They receive AHRS data via the tether and command the helicopter servos via the tether.

Tether with two RS-232 cables

RS-232 #1: Receives data from AHRS

RS-232 #2: Sends data to servo command board

Figure 3 – High-level Flight Control System Diagram

Early Phase Avionics for Neural Network Prototype Control System Design & Test

During this time period I was trying to demonstrate the feasibility of demonstrating the ability of Neural Networks to perform real-time flight control for specific modes – in this case, for an RC (Radio-Control) helicopter. The two areas of control that I focused on were: 1) altitude control, and 2) heading control. In each case I set up a lab environment with the necessary test rigging to test each specific concept.

Accelerometer for Altitude Control System Development

A flight test rig was customized for constraining the helicopter to only move vertically – the horizontal portion of the test stand was held in place by two vertical planks, as shown below in Figure 1. An accelerometer - also called "tilt sensor", was mounted to the horizontal aluminum beam to provide the measurement of altitude. The sine of the angle was used to compute the altitude.

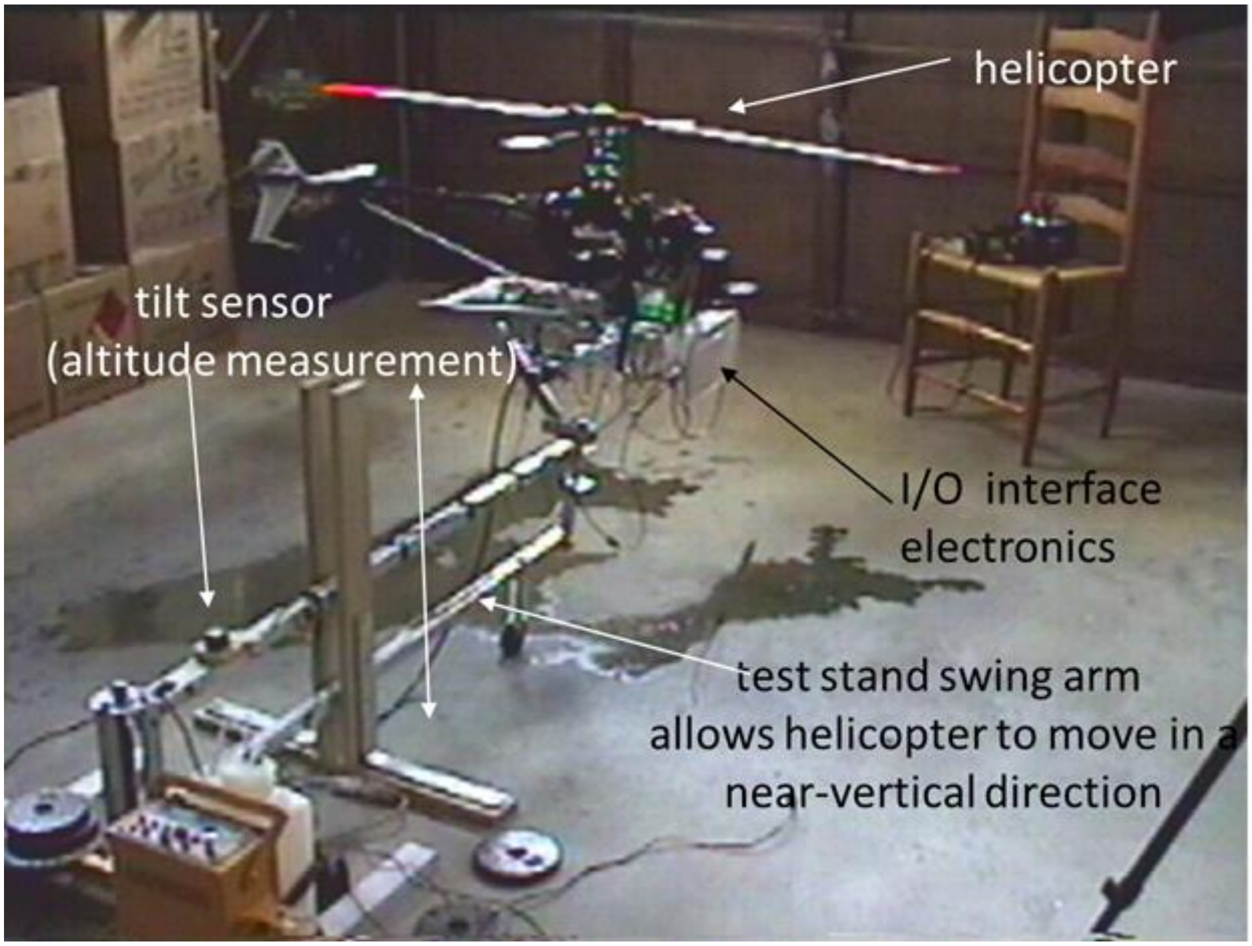


Figure 1 – Helicopter Vertical-Movement-Only Constrained Test Rig

A close-up view of the tilt sensor is shown below in figure 2. The tilt sensor cost approximately $200 and provided an attenuated voltage output which indicated the tilt angle for each of the three axes.

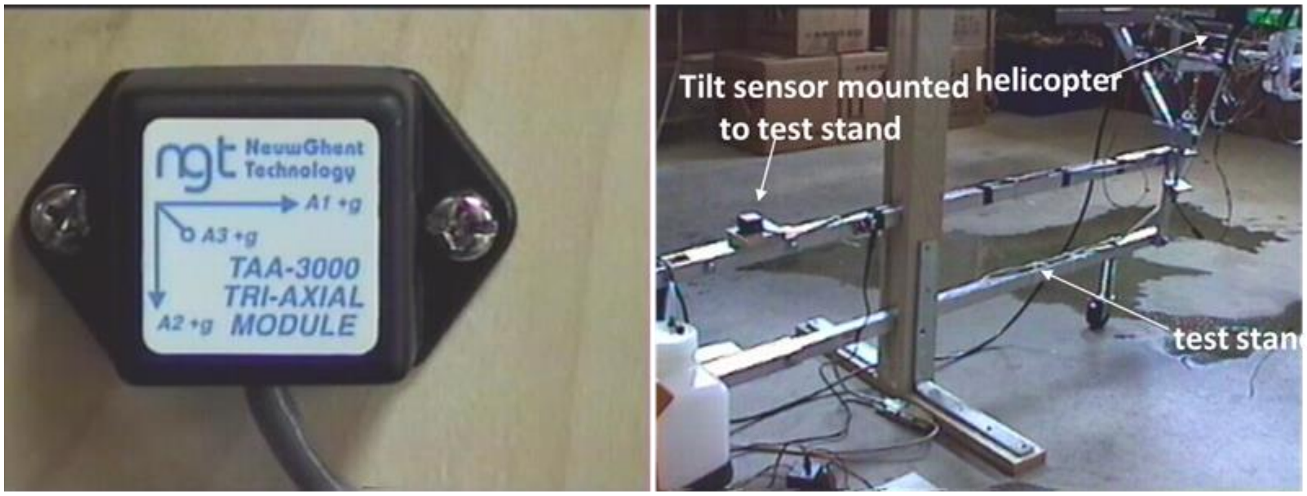


Figure 2 – 3-Axis Accelerometer (Tilt Sensor) Mounted to Flight Test Stand

The one avionics board (which commanded the servo actuators and also read voltage inputs) and relay circuit switch, were mounted in a simple Tupperware container, as shown below in Figure 3.

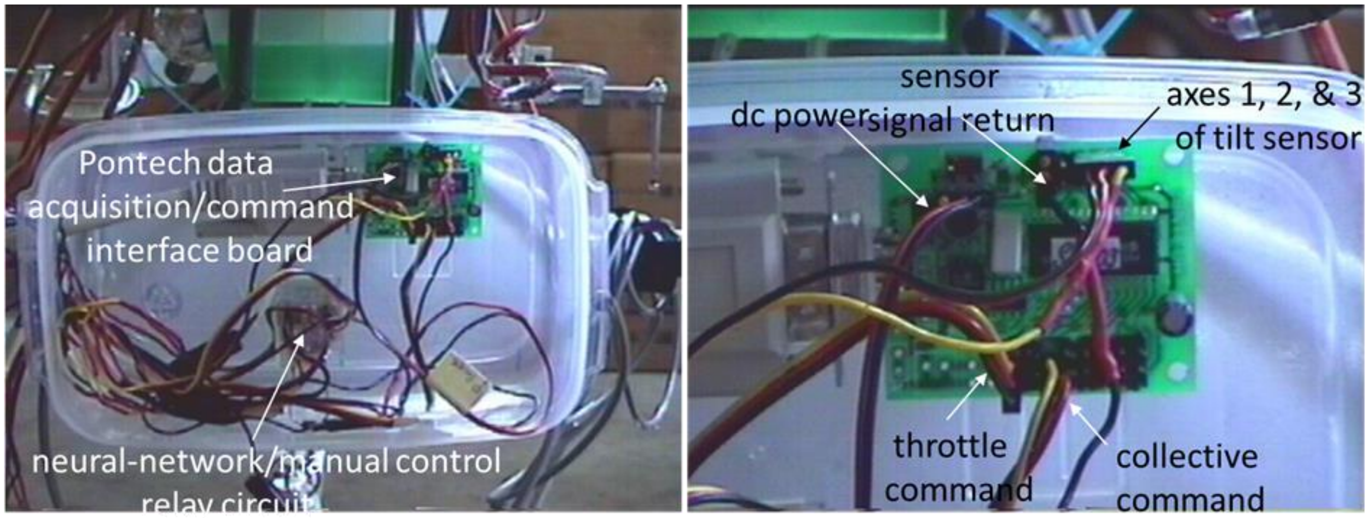

Pontech data acquisition/command interface board
neural-network/manual control relay circuit
dc power
sensor signal return
axes 1, 2, & 3 of tilt sensor
throttle command
collective command

Figure 3 – Simple Avionics Setup

The helicopter altitude flight control system diagram is shown below in Figure 4.

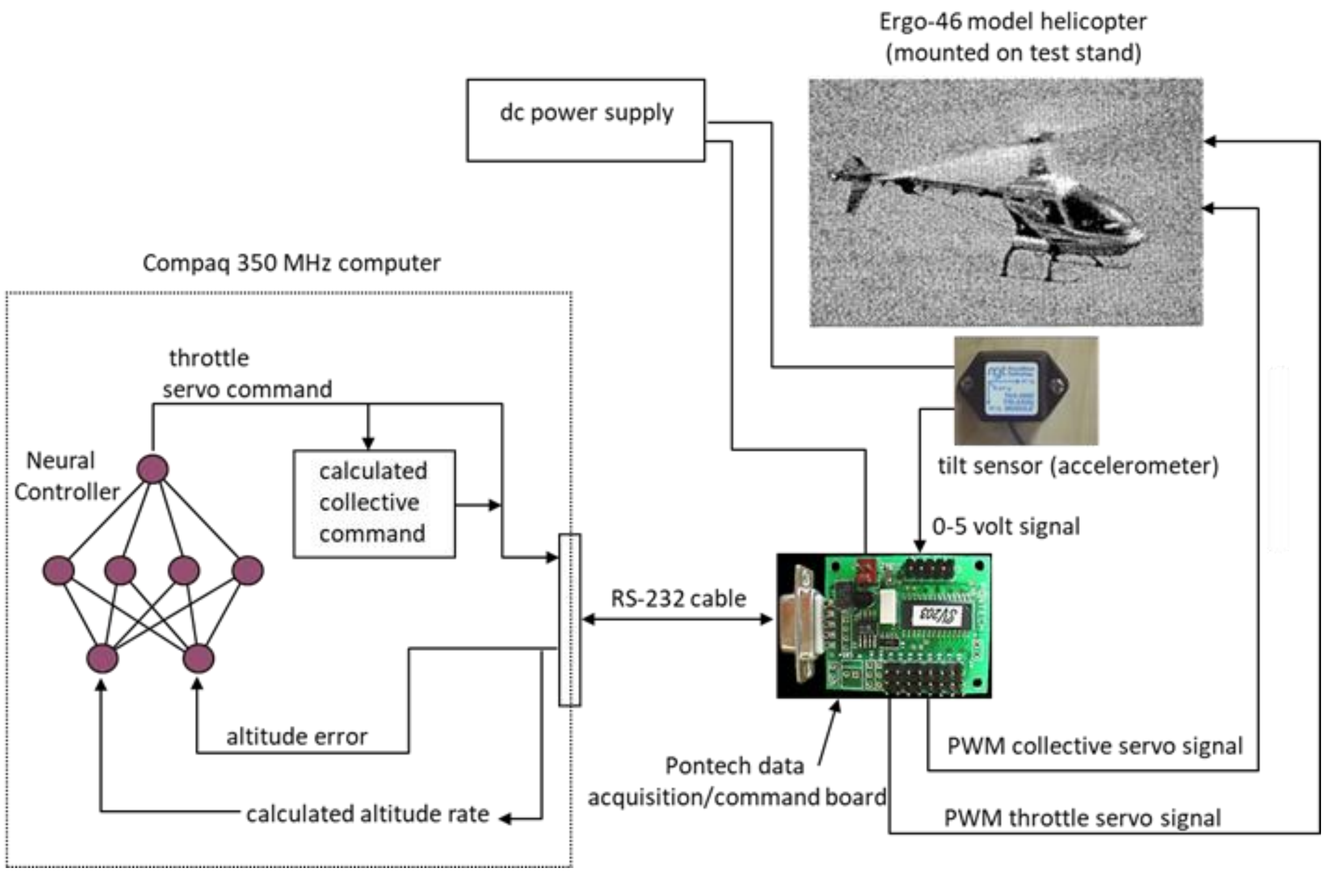


Figure 4 – Helicopter Neural Network Altitude Flight Control System

Given the simplicity of the test setup and avionics components, the control system results were remarkable – the Neural controller was able to raise and lower the helicopter to the commanded altitudes in a smooth and stable manner, as shown below in Figure 5.

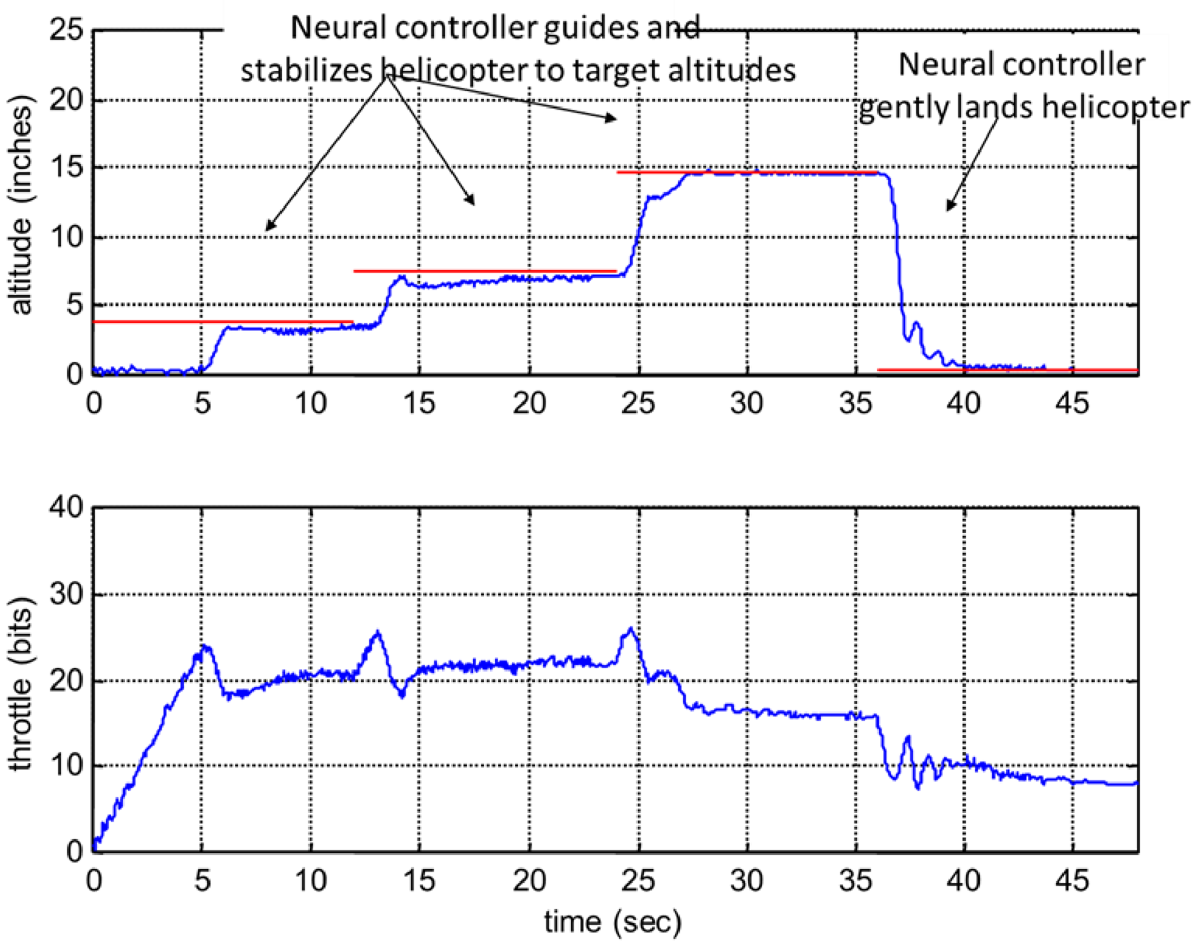


Figure 5 – Performance of Neural Network Altitude Control System

Rotary Potentiometer for Heading Control System Development

A different flight test rig was customized for constraining the helicopter to only move about the yaw axis. A rotary potentiometer – shown below in Figure 1, purchased from a local electronics store, was used to measure the helicopter movement about the yaw axis.

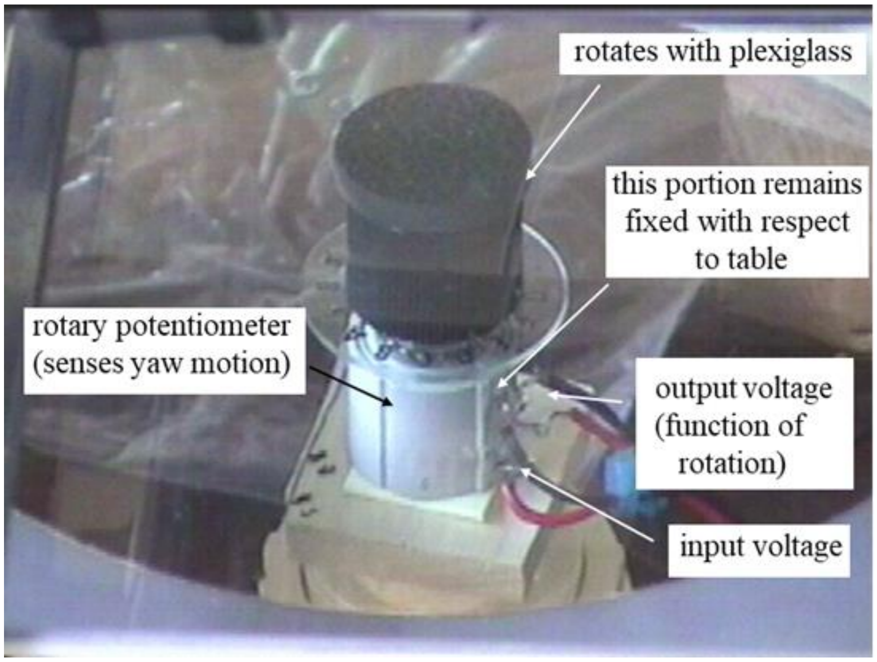

rotates with plexiglass
this portion remains fixed with respect to table
rotary potentiometer (senses yaw motion)
output voltage (function of rotation)
input voltage

Figure 1 – Rotary Potentiometer for Measuring Heading Angle about Yaw Axis

A “Lazy Susan” (normally used on a dining table) framework was integrated with circular pieces of plexiglass and wood and mounted on top of the rotary potentiometer, as shown below in Figure 2. When the helicopter moved about the yaw axis, the rotary potentiometer provided voltage levels that indicated that angular (heading) position of the helicopter.

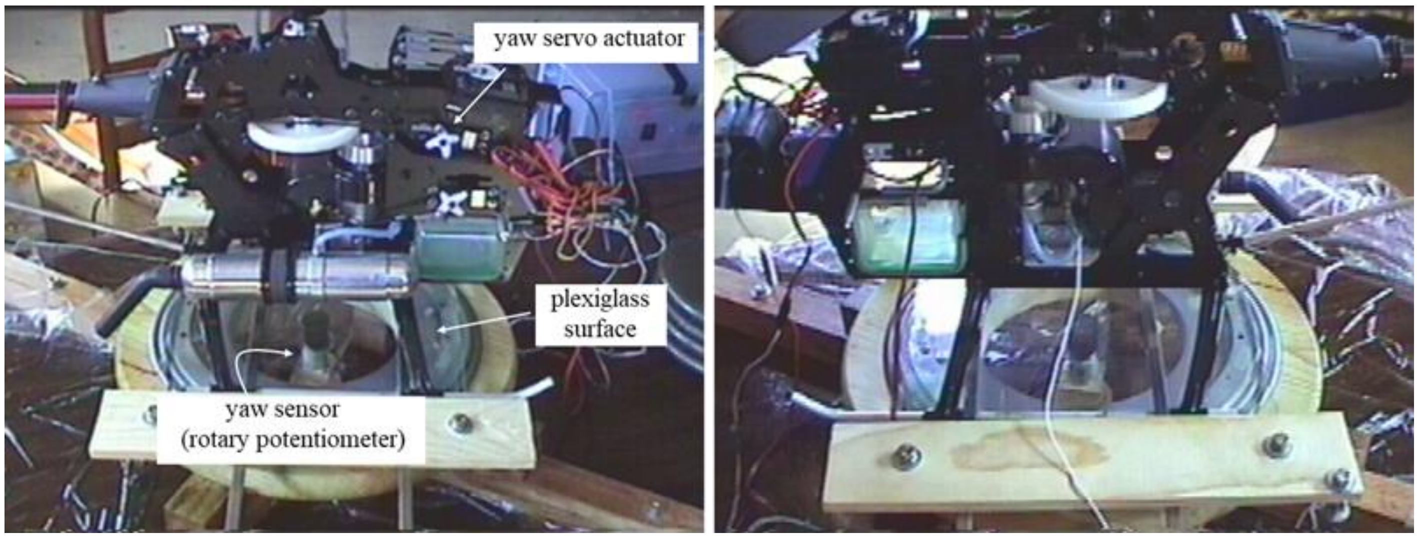


Figure 2 – Helicopter Yaw-Movement-Only Constrained Test Rig

The complete system is shown below in Figure 3.

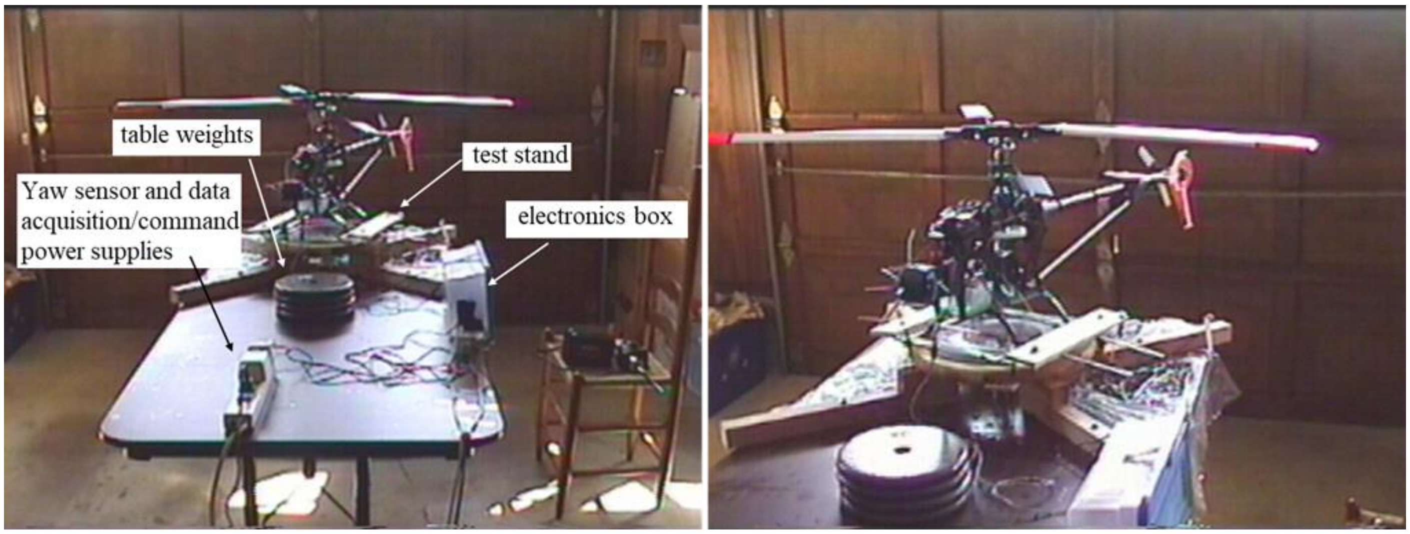


Figure 3 – Complete Helicopter Yaw/Heading Control Test Rig

The flight control system diagram is shown below in Figure 4.

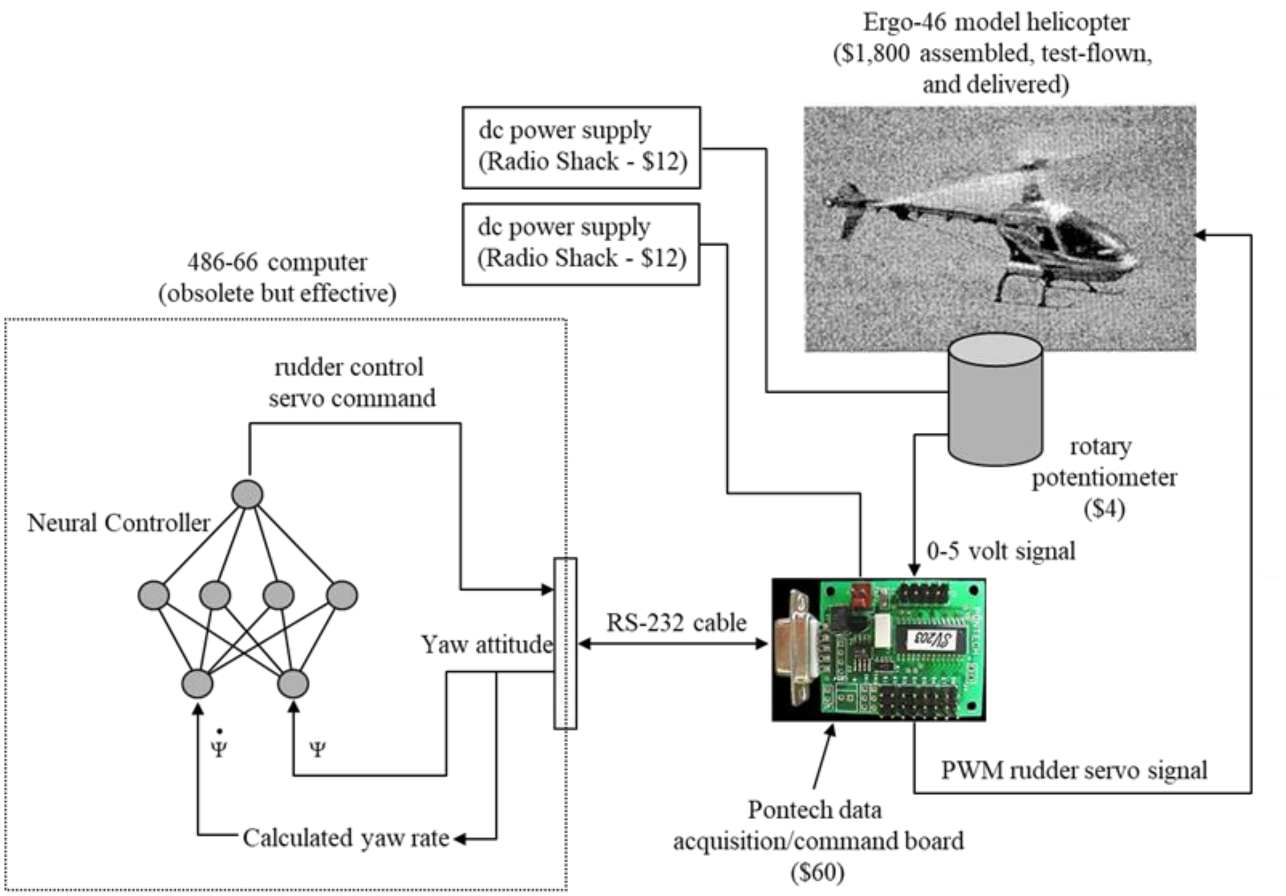

Ergo-46 model helicopter ($1,800 assembled, test-flown, and delivered)
dc power supply (Radio Shack - $12)
dc power supply (Radio Shack - $12)
486-66 computer (obsolete but effective)
rudder control servo command
Neural Controller
Yaw attitude
RS-232 cable
rotary potentiometer ($4)
0-5 volt signal
$\dot{\Psi}$
Ψ
Calculated yaw rate
PWM rudder servo signal
Pontech data acquisition/command board ($60)

Figure 4 – Helicopter Neural Network Heading Flight Control System

Given the simplicity of the test setup and avionics components, the control system results were remarkable – the Neural controller was able to slew the helicopter to the commanded heading in a smooth and stable manner, as shown below in Figure 5.

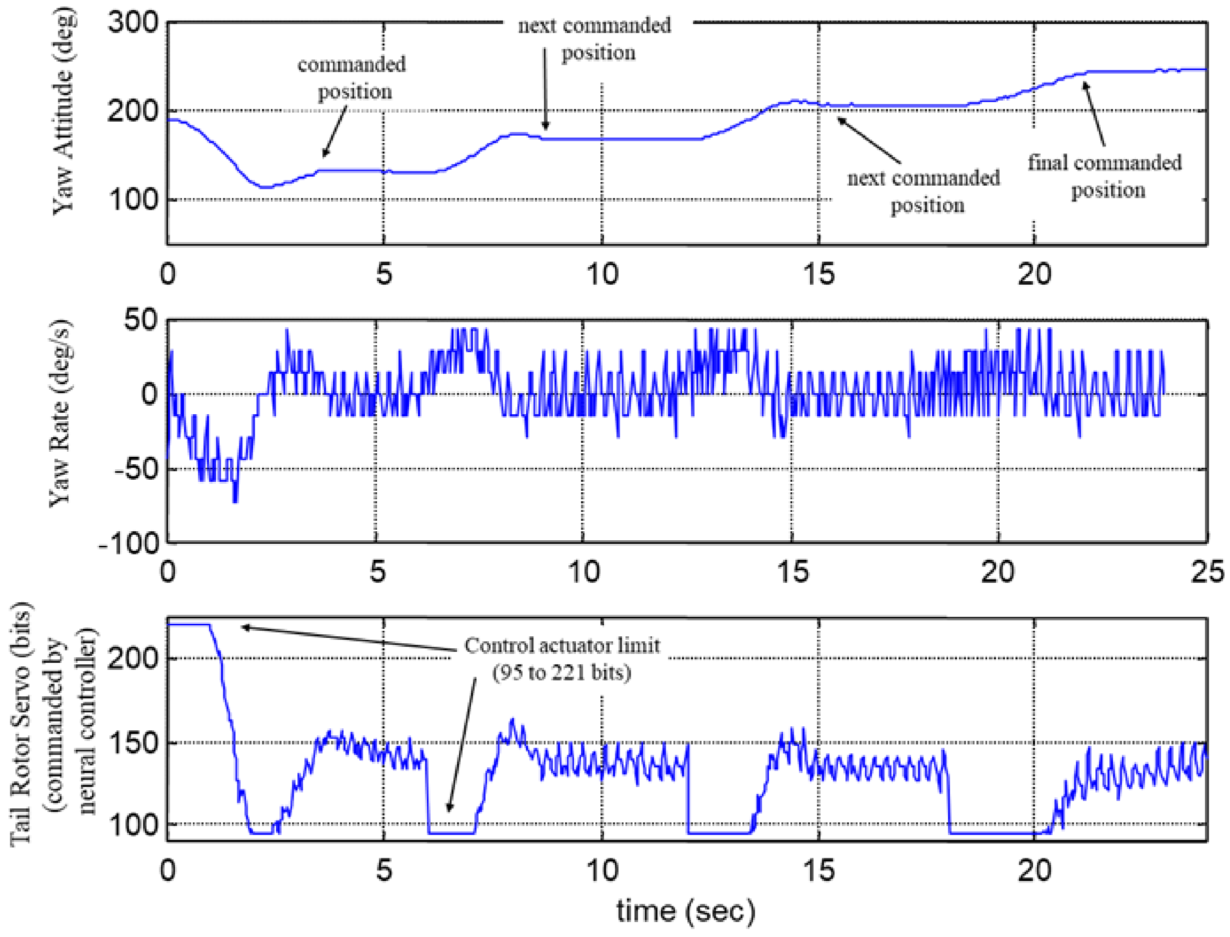


Figure 5 – Performance of Neural Network Heading Control System