	Queensland University of Technology	QUT Systems Engineering TAQ17G1	Doc No: TAQ17G1-WEB-TS-01 Issue: 1.0 Page: 1 of 23 Date: 24 March 2017
---	--	---	---

Final Design Web Visualisation

Prepared by _____ Date _____

Daniel Huffer, Web Visualization

Checked by _____ Date _____



Approved by _____ Date _____


Charlie Shaw-Feather, Project Manager

Authorised for use by _____ Date _____

Dr. Felipe Gonzalez, Project Coordinator

Queensland University of Technology
Gardens Point Campus
Brisbane, Australia, 4001.

This document is Copyright 2017 by the QUT. The content of this document, except that information which is in the public domain, is the proprietary property of the QUT and shall

	Queensland University of Technology	QUT Systems Engineering TAQ17G1	Doc No: TAQ17G1-WEB-TS-01 Issue: 1.0 Page: 2 of 23 Date: 24 March 2017
---	--	---	---


not be disclosed or reproduced in part or in whole other than for the purpose for which it has been prepared without the express permission of the QUT

Revision Record

Document Issue/Revision Status	Description of Change	Date	Approved
1.0	Initial Issue	22/10/2017	

Table of Contents

Paragraph No.	Page
1 Introduction	6
1.1 Scope	6
1.2 Background	6
2 Reference Documents	7
2.1 QUT Avionics Documents	7
2.2 Non-QUT Documents	7
3 Subsystem Introduction	8
4 Subsystem Architecture	9
4.1 Interfaces	10
5 Final Design	11
5.1 Final Hardware Design	11
5.2 Final Software Design	11
5.2.1 HTML & CSS	11
5.2.2 LAMP	11
5.2.2.1 APACHE	11
5.2.2.2 MySQL	12
5.2.2.3 PHP	12
5.2.3 jQuery & Plotly	12
5.2.4 Ajax	12
5.3 Example Mark-up Diagram	13
5.4 Final Mark-Up Design	14
6 Design Features	15
6.1 Image Storage	15
6.2 Auto Refresh	15
6.3 WVI ROS Package	15
7 WVI Components Design	15
7.1 Scrolling sidebar of images	15


	Queensland University of Technology	QUT Systems Engineering TAQ17G1	Doc No: TAQ17G1-WEB-TS-01 Issue: 1.0 Page: 4 of 23 Date: 24 March 2017
---	--	---	---

7.2	Main Image Display	18
7.3	UAV positional information	18
7.4	UAV State Information	19
7.4.1	Armed	19
7.4.2	Flying	20
7.5	3D Positional Map	20
7.6	Bar graph Sensor Displays	21
8	Conclusion	22
9	Appendix A – Plotly 3D JavaScript Graphing	23



Definitions

UAV	Unmanned Aerial Vehicle
QUT	Queensland University of Technology
GCS	Ground Control Station
WVI	Web Visualisation Interface
LAN	Local Area Network
HLO	High Level Objective
LAMP	Linux, Apache, MySQL, PHP
ROS	Robot Operating System
UAVTAQ	Unmanned Aerial Vehicle: Target Air Quality Monitoring
HTML	Hypertext Mark-up Language
PHP	Hypertext Pre-processor
JS	JavaScript
MySQLi	MySQL improved
API	Application programming interface
DOM	Document Object Model
JSON	JavaScript Object Notation
Ajax	Asynchronous JavaScript and XML

 Queensland University of Technology	QUT Systems Engineering TAQ17G1	Doc No: TAQ17G1-WEB-TS-01 Issue: 1.0 Page: 6 of 23 Date: 24 March 2017
--	---	---

1 Introduction


This document outlines the final design for Group 1's WVI subsystem. The WVI subsystem is responsible for displaying data in live time on a website for a UAV tasked to fly around a small simulated underground mine environment. The webpage is hosted on a webserver and is accessible to anyone on the same local area network.

1.1 Scope

The scope of this document outlines the WVI subsystem and its components, where design logic is detailed.

1.2 Background

Queensland University of Technology Airborne Sensing Systems are a world-leading research team based in Brisbane, Australia. Focusing on the innovation of autonomous UAV flight using on board sensing equipment they strive to turn cutting edge technology along with leading edge concepts from paper into flight tested reality. As a part of the team they have commissioned students EGB349 to design and build a UAVTAQ multi rotor based UAV to take targeted air samples within a simulated mine. The idea is that there are potentially deadly CO₂ leaks occurring down within a mine which otherwise cannot be physically inspected due to the potential risks involved. This is where the UAVTAQ is required. The UAV will fly in using sensor navigation the known locations and take air samples to confirm the leaks. This information can then be delivered safely to teams outside the mine.

	Queensland University of Technology	QUT Systems Engineering TAQ17G1	Doc No: TAQ17G1-WEB-TS-01 Issue: 1.0 Page: 7 of 23 Date: 24 March 2017
---	--	---	---


2 Reference Documents

2.1 QUT Avionics Documents

RD/1	UAVTAQ - Customer Requirements	UAVTAQ Customer Requirements 2017
RD/2	PMP of UAVTAQ	The PMP for UAVTAQ Group 1
RD/3	UAVTAQ17-PM-SR	System Requirements
RD/4	TAG16GM2-VWI-TS-02	Web Interface Subsystem Trade Study

2.2 Non-QUT Documents

RD/5	https://httpd.apache.org
RD/6	http://wiki.ros.org/ROS/Introduction
RD/7	https://www.oracle.com/mysql/index.html
RD/8	https://http.jquery.com/
RD/9	https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started

 Queensland University of Technology	QUT Systems Engineering TAQ17G1	Doc No: TAQ17G1-WEB-TS-01 Issue: 1.0 Page: 8 of 23 Date: 24 March 2017
--	---	---

3 Subsystem Introduction

The web visual interface subsystem (WVI) derives from a High-Level Objectives (HLO's) specified in the document RD/1, UAVTAQ - Customer Requirements. It is stated that “The system shall have capability to visualise real time flight information, CO2 concentration, temperature, humidity, and imagery data”. It is also mentioned that “The web interface must be accessible from another computer on the Local Area Network (LAN)” and “The interface should be designed and run as a web server, but can be hosted on a local machine”. These HLO requirements were translated into equivalent system requirements (REQ-M-7-1, REQ-M-7-2, REQ-M-7-3). The core of the WVI is what's commonly referred to as a web development solution stack. A solution stack consists of a bundle of different software with unique functionality designed to interact with one another. The basis of this is that you need a web server to handle client requests, a database to store information and one form or another of scripting language to manipulate data based on client requests.

4 Subsystem Architecture

The subsystems architecture and interface control diagram can be seen below in Figure 1.

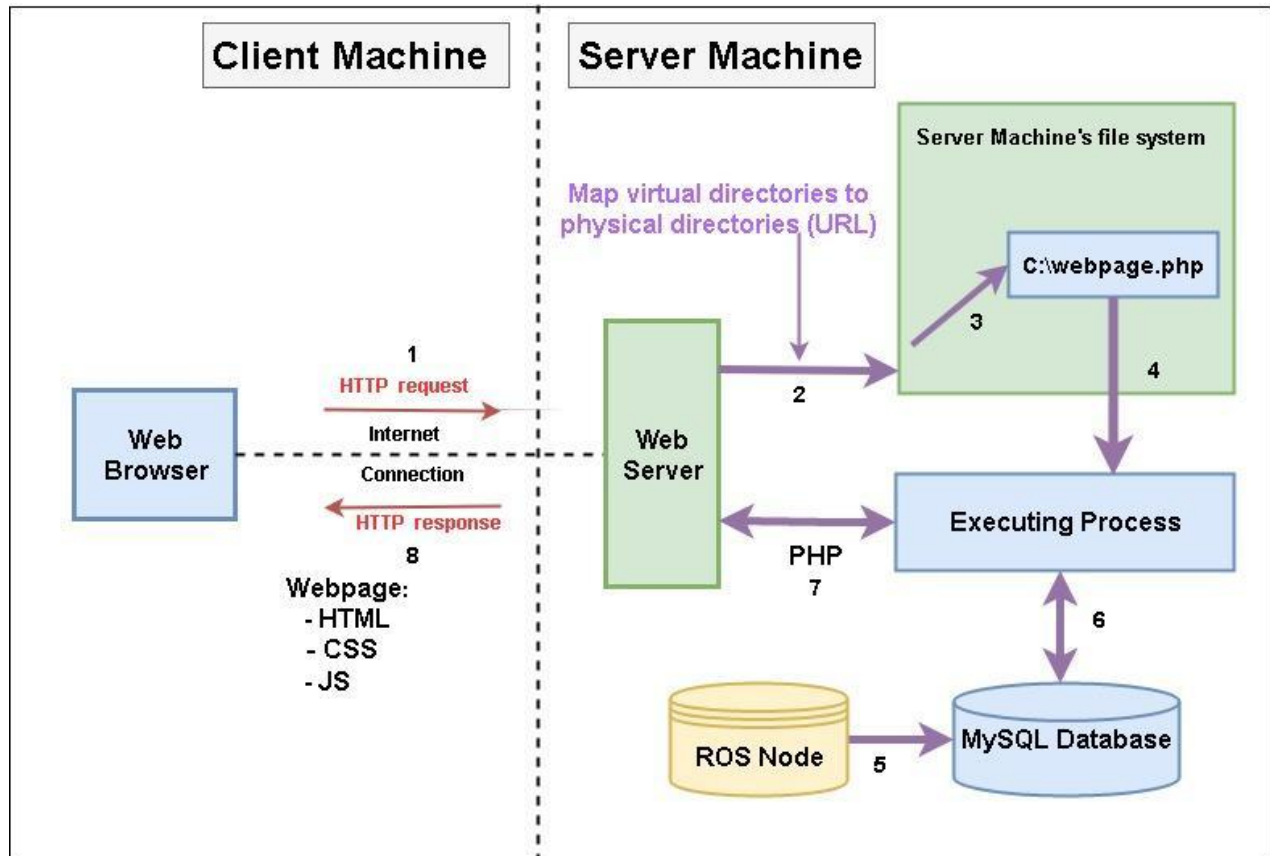



Figure 1 - Web Visualization Subsystem Overview

 Queensland University of Technology	QUT Systems Engineering TAQ17G1	Doc No: TAQ17G1-WEB-TS-01 Issue: 1.0 Page: 10 of 23 Date: 24 March 2017
--	---	--

4.1 Interfaces

- Interface 1 Client web browser makes a request, upon request the webserver receives a URL.

- Interface 2 Web Server uses URL to map virtual directories to physical directories on the local machine.

- Interface 3 Web Server locates PHP file on the machine.


- Interface 4 PHP script is executed.

- Interface 5 UAV Data from the ROS network is sent to the MySQL database.

- Interface 6 PHP can retrieve and modify data stored in the MySQL database.

- Interface 7 PHP interfacing with apache webserver, execution output streamed down the HTTP connection.

- Interface 8 HTML, CSS and JS and any additional output sent back over the internet / network to the requesting party.

 Queensland University of Technology	QUT Systems Engineering TAQ17G1	Doc No: TAQ17G1-WEB-TS-01 Issue: 1.0 Page: 11 of 23 Date: 24 March 2017
--	---	--

5 Final Design

The WVI subsystem was designed to have the core elements auto refreshing through Asynchronous JavaScript and XML (Ajax). To accomplish this, data was passed from the server in JavaScript Object notation (JSON) format to the client side to be parsed and displayed asynchronously.

5.1 Final Hardware Design

The entirety of the WVI sub system is comprised of various pieces of software. The only hardware interaction been that of the GCS and client computer. Additionally, there are no requirements for these pieces of hardware other than they operate to an acceptable standard. It's for these reasons that no hardware design is specified.

5.2 Final Software Design

The final software design was comprised of the LAMP stack and the WVI ROS package. The WVI ROS package contained nodes for the current/goal x,y,z position, images from the raspberry pi and sensor data.

5.2.1 HTML & CSS

Much of the code for the subsystem is written in HTML and CSS, the HTML code accounts for the structure of the webpage. While the CSS enables the customisation and styling of all structured HTML elements.

5.2.2 LAMP

The LAMP software bundle is completely open source and as previously mentioned runs on the Linux operating system, a further break down of the components can be seen below.

5.2.2.1 APACHE

Apache provides a secure, efficient and extensible server that provides HTTP services in sync with current HTTP standards (RD/5) As part of the solution stack version 2.4.x from the stable branch was used.

5.2.2.2 MySQL

MySQL is a relational database management system based on Structured Query Language (SQL). SQL is labelled one of the most popular languages for managing database content due to its flexibility and ease of use (RD/7). The MySQL database used in the WVI subsystem was interfaced with PhpMyAdmin to assist with development. All the data except the POV image from the UAV that the customer would like to be able to visualise was sent to tables constructed in MySQL. The live POV images were stored in a local directory to avoid additional overheads associated with image storage within MySQL. MySQL queries in PHP was done using the PDO API over MySQLi. For the subsystem, the latest release MySQL 5.7.x was used.

5.2.2.3 PHP

PHP which is a recursive acronym for PHP: Hypertext Pre-processor, is a server side scripting language for web development. Writing PHP code into your HTML files essentially allows you to generate HTML code dynamically, manipulate data and interface with databases. The latest release PHP 7.1.x was used to interface with Apache and MySQL.

5.2.3 jQuery & Plotly

jQuery is a feature rich JavaScript library that makes tasks like HTML document traversal and manipulation, event handling, animation and Ajax much simpler to do (RD/8). The jQuery library was used in the WVI to generate the graphs displayed in the WVI. API from plotly was also used to create 3D graphing within the JavaScript framework. The 3D graph was used as a model to map the current and previous UAV locations.

5.2.4 Ajax

Ajax stands for Asynchronous JavaScript and XML and is a method to communicate with servers through the XMLHttpRequest object (RD/9). The reason for using Ajax in the WVI final design was to allow the webpage to make requests to Apache web server without refreshing the page. This vastly improved the user experience of the WVI.

5.3 Example Mark-up Diagram

An example layout of the visual web interface can be seen above in Figure 3. This example layout was provided as part of the customer needs documentation (RD/1). The core elements of the example have been labelled with a small description briefly relating the mentioned software in this document to its display functionality.

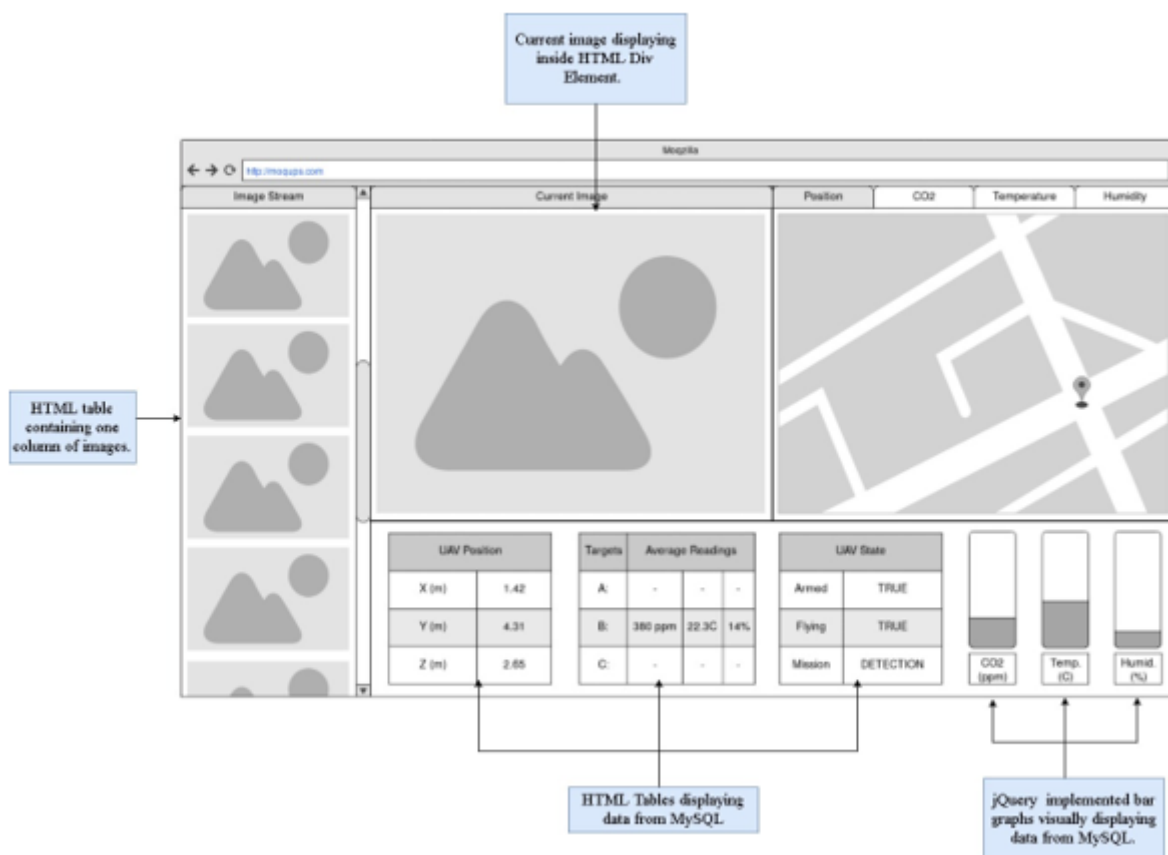


Figure 2 - Example mark-up design



5.4 Final Mark-Up Design

Following on from the example mark-up, the final design of the webpage UI can be seen below in Figure 3. Labelled features of the design are discussed below in section 7 - WVI Component Design.

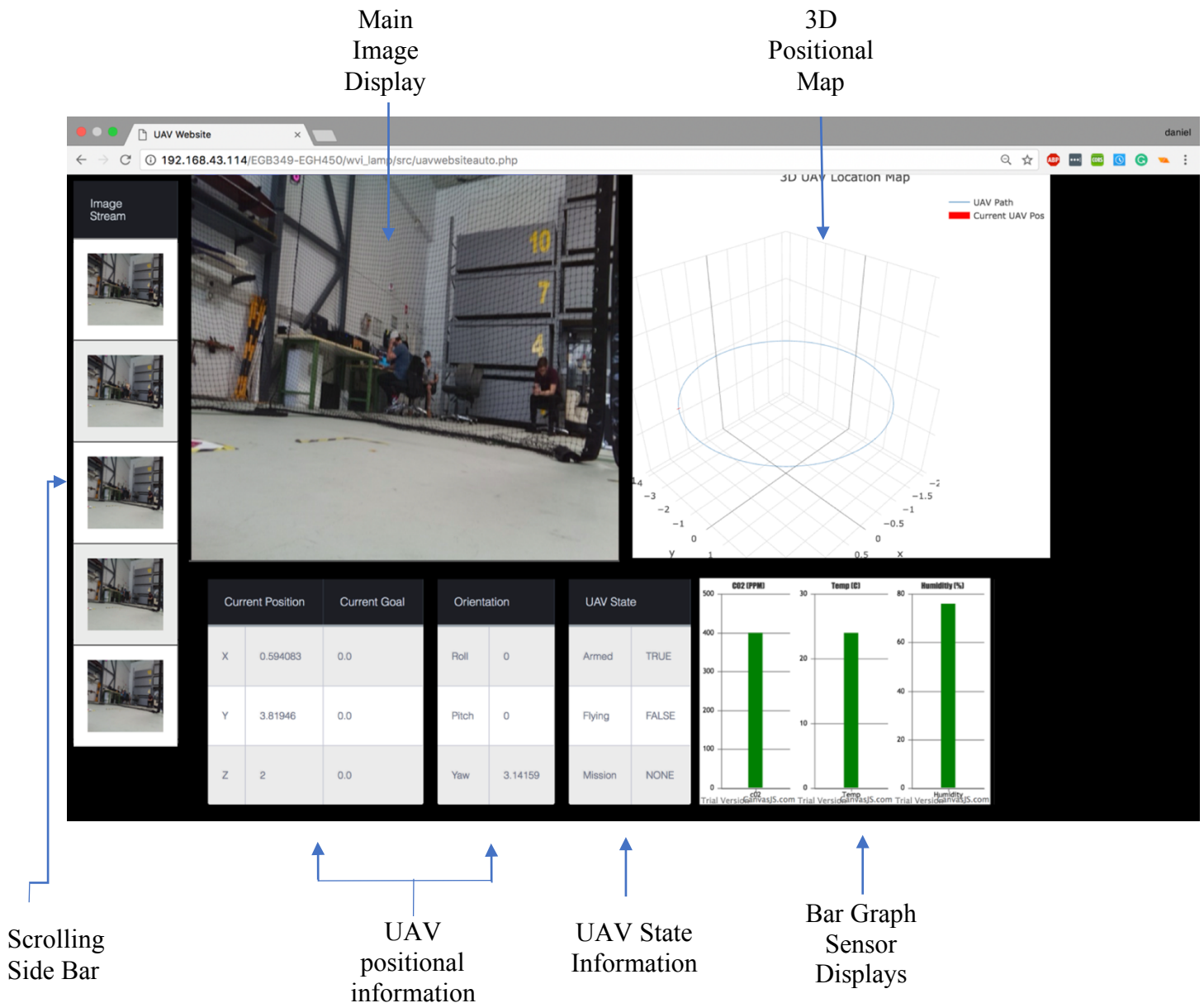



Figure 3 - Final WVI Design

 Queensland University of Technology	QUT Systems Engineering TAQ17G1	Doc No: TAQ17G1-WEB-TS-01 Issue: 1.0 Page: 15 of 23 Date: 24 March 2017
--	---	--

6 Design Features

6.1 Image Storage

The images obtained through ROS from the TAQ subsystem could be stored straight into MySQL or on the local disk. For this implementation, images were stored locally within the root directory of the webserver. This design choice avoided additional overheads associated with image storage within MySQL. Additionally, as the images were saving into the webserver directory they were still accessible over the network if required.

6.2 Auto Refresh

In the initial design of the WVI the data displaying would not update on the webpage until the user manually refreshed the page. This was inconvenient so for a nicer user experience, all the core components of the page displaying live data were programmed to automatically update using Ajax.

6.3 WVI ROS Package


Due to the nature of what the WVI was required to do many ROS scripts had to be running simultaneously. To make this process as simple as possible, a package called wvi was made in the catkin_ws. Within this package all required ROS nodes were stored inside. A launch script was created so all nodes could be initialised at once through a roslaunch command in the linux terminal.

7 WVI Components Design

This section details the design logic behind each of the core components on the WVI.

7.1 Scrolling sidebar of images

The image stream on the left-hand side of the WVI was designed to display the most recent 10 images. To accomplish this, a MySQL table was made to store all the image names as images were received through ROS. The method of using the MySQL table to store the image names was necessary as iterating the image files in the directory using straight PHP was not displaying the saved images in the correct order of most recent first. During code execution of the PHP file setup for the image sidebar the most recent 10 image names were fetched and formatted into JSON format before been passed through to the client side using Ajax. From there JS was used to iterate elements in the DOM assigning the images the src directory. The

 Queensland University of Technology	QUT Systems Engineering TAQ17G1	Doc No: TAQ17G1-WEB-TS-01 Issue: 1.0 Page: 16 of 23 Date: 24 March 2017
--	---	--

scrolling sidebar component was coded so that clicking on an image will then display the given image in the main image display. This was done using a simple JavaScript function that takes the clicked document object as an input parameter and assigns the src to the main image on the page. The scrolling sidebar was set to update itself on the webpage every 4 seconds. sCode required for the scrolling sidebar of images can be seen below in Figures 2,3,4, and 5.

```
setInterval(function(){
$.getJSON("sidebarauto.php", function(data){
for (i = 0; i < 10; i++) {
    image = "image"+i.toString();
    //console.log(image);
    document.getElementById(image.toString()).src = "../../images/"+data[i];
document.getElementById(image.toString()).name = data[i];
}
});
},4000);
```

Figure 1 - Ajax for sidebar, 4000 ms refresh timer

```
<?php
$pdo = new PDO('mysql:host=127.0.0.1;dbname=UAVDATA', 'root', 'mysql');
$stmt = $pdo->query("SELECT image FROM imgnames ORDER BY id DESC LIMIT 10");
$stmt->execute();
$arr = array();
foreach($stmt as $row){
$arr[] = $row['image'];
}
$sidebar = json_encode($arr);
echo $sidebar;
?>
```

Figure 2 - PHP code in "sidebarauto.php" used to assign images to the document elements set up for the side bar images

```
<?php
for($i = 0; $i < 10; $i++){
echo'<tr><td style="width:30px; height:100px; bgcolor=blue" > <img src="" id="image'.$i.'"
onclick="swap(this)" name="" style="height:100%;width: 100%;> </td></tr>';
}
?>
```

Figure 3 - PHP code to generate sidebar images with swap function for on click event handler



```
function swap(image){  
document.getElementById("mainimg").src = image.src;  
}
```

Figure 4 - JavaScript function to display selected sidebar image

7.2 Main Image Display

To display the most recently captured image from the UAV, the MySQL table mentioned in 7.1 containing all image name strings was queried. Using JS and the queried result, the image was then able to be assigned to the div element styled specifically to display the main image.

```
$stmt = $pdo->query("SELECT image FROM imgnames ORDER BY id DESC LIMIT 1");
$stmt->execute();
foreach($stmt as $row){
  $fullimg = '../..../images/'. $row['image'];
  $img = json_encode($fullimg);
  $all[4] = $img;
}
```

Figure 5 - PHP code used to pass the most recent image name stored in the database to the client side

7.3 UAV positional information

The positional information for the UAV was obtained through the distributed ROS network and stored into a MySQL table. The table was accessed using PHP to take the latest data and display it in the HTML table seen below in Figure 7.

Current Position		Current Goal	Orientation	
X	0.594083	0.0	Roll	0
Y	3.81946	0.0	Pitch	0
Z	2	0.0	Yaw	3.14159

Figure 6 - HTML table for positional information

In the script used to listen to positional data over ROS, it was setup so that when the script runs, it will delete an existing table of positional information if one exists, then re make a fresh table to store the data. This was useful in helping keep control of how much data was been stored in the database and made working with the data easier.


```
def initDB():|
#setup cursor
    global db
    global cursor
#make a fresh table for the data every session
    cursor.execute("DROP TABLE IF EXISTS uavloc")
#remaking the table
    sql = """CREATE TABLE uavloc (
        x FLOAT,
        y FLOAT,
        z FLOAT,
        id INT,
        roll FLOAT,
        pitch FLOAT,
        yaw FLOAT )"""
    cursor.execute(sql)
#creating primary key
    sql2 = """ALTER TABLE uavloc ADD PRIMARY KEY(id)"""
    cursor.execute(sql2)
```

Figure 7 - Database function in the script to start subscriber for the UAV positional information

7.4 UAV State Information

7.4.1 Armed

To display the Boolean true/false result for the current state of the UAV, a subscriber node was made to listen to the specific MAVROS topic publishing the state of the UAV. If the UAV was armed, a value of 1 was set in a MySQL table holding a variable responsible for the state of the UAV. PHP was then used to query the database and determine if the armed variable is either a 1 or a 0 and then display either TRUE or FALSE using JS to modify the DOM. All this logic was included in an Ajax set interval function and was periodically executed every 4 seconds.

 Queensland University of Technology	QUT Systems Engineering TAQ17G1	Doc No: TAQ17G1-WEB-TS-01 Issue: 1.0 Page: 20 of 23 Date: 24 March 2017
--	---	--

7.4.2 Flying

The logic used to determine if the UAV was flying or not was to simply check if the UAV is armed and check the UAV's position in the z axis. If the UAV is armed and it is above the ground then it was deemed that the UAV is flying.

```

if(z[0] > 0.3 && document.getElementById("armed").innerHTML == "TRUE"){
//if the z position of the drone is 300 mm of the ground and the drone is armed then it is
flying
document.getElementById("flying").innerHTML == "TRUE";
}else{
document.getElementById("flying").innerHTML == "FALSE";
}

```

Figure 8 - JS code for flying logic

7.5 3D Positional Map

To display the position of the UAV relative to the flight zone a 3D graphing API plotly was used. In the JavaScript code for map, the UAV trace was assigned as the most recent values fetched in the MySQL database call for the x,y,z positional information. All the positional data to be plotted was passed as three separate vectors into a function designed to display the map. An example of how the map itself is implemented can be seen in Appendix A - Plotly 3D JavaScript Graphing. The x,y,z positional data was obtained through ROS and stored into a UAV location table in MySQL. To pass this data into the plot function, PHP was used to query the table holding the data for the last 100,000 or less entries in descending order by id. The 100,000 limit on the query was put in place as it was a good amount of previous entries to display a short history of the UAV's flight path. The data obtained from MySQL was then put into an array of arrays and echo'd from the server side to the client side using Ajax and passed into the map function. The PHP for the server side processing code and JS client side processing can be seen below in Figure 10 and 11.



```
<?php
$x = Array();
$y = Array();
$z = Array();
$all = Array();
$pdo = new PDO('mysql:host=127.0.0.1;dbname=UAVDATA', 'root', 'mysql');
$stmt = $pdo->query("SELECT x,y,z FROM uavloc ORDER BY id DESC LIMIT 100000");
$stmt->execute();
foreach($stmt as $row => $val){
    $x[] = $val['x'];
    $y[] = $val['y'];
    $z[] = $val['z'];
}
$xjson = json_encode($x);
$yjson = json_encode($y);
$zjson = json_encode($z);
$all[0] = $xjson;
$all[1] = $yjson;
$all[2] = $zjson;
```


Figure 9 - PHP code for server side processing of x,y,z positional data for 3D map display

```
setInterval(function(){
    $.getJSON("xyz_auto.php", function(data){
        var x = JSON.parse(data[0]);
        var y = JSON.parse(data[1]);
        var z = JSON.parse(data[2]);
        UAVPos(x, y, z);
    });
}, 1000);
```

Figure 10 - Client side processing of x,y,z positional data for 3D map display

7.6 Bar graph Sensor Displays

Temperature, humidity and c02 readings were displayed in the form of bar graphs so the values can be easily observed, the bars were also setup so that hovering the mouse over the bar itself will give you the exact value. One design feature that was added into the display was threshold values. Certain values for each sensor were set as critical threshold values, if readings exceed the threshold limit, colours of the display are set to change from green to red. This was added so critical readings could instantly be noticed.

	Queensland University of Technology	QUT Systems Engineering TAQ17G1	Doc No: TAQ17G1-WEB-TS-01 Issue: 1.0 Page: 22 of 23 Date: 24 March 2017
---	--	---	--

8 Conclusion

The final design of the WEB subsystem was successful in satisfying specified subsystem requirements (RD/3) relevant to the WEB HLOs (RD/1). The webpage was hosted on the Apache webserver and was accessible over the local area network. The subsystem served real time dynamic content from the UAV in the form of HTML, through PHP server side scripting and Ajax client/server communication. All dynamic data displayed in the WVI was obtained strictly through the ROS node network.

9 Appendix A – Plotly 3D JavaScript Graphing

Below is an example JavaScript mark-up function using plotly API. In the code below, the function UAVPos takes in three input parameters xvector, yvector, zvector. All three of the input parameters are arrays containing the positional information for each axis of the plot. This data will be plotted to represent the current flight path of the UAV, current goals and identified targets will also be marked within the 3D Map.

```
function UAVPos(xvector, yvector, zvector){  
  Plotly.d3.csv('https://raw.githubusercontent.com/plotly/datasets/master/_3d-line-plot.csv', function(err, rows){  
    function unpack(rows, key) {  
      return rows.map(function(row)  
        { return row[key]; });  
    }  
    var trace1 = {  
      x: xvector,  
      y: yvector,  
      z: zvector,  
      mode: 'lines',  
      name: 'UAV Path',  
      marker: {  
        color: '#1f77b4',  
        size: 12,  
        symbol: 'circle',  
        line: {  
          color: 'rgb(0,0,0)',  
          width: 0  
        }  
      },  
      line: {  
        color: '#1f77b4',  
        width: 1  
      },  
      type: 'scatter3d'  
    };  
  
    Plotly.newPlot('second', data2, layout);  
  });  
}
```