# EMBRY-RIDDLE
## Aeronautical University™
### SCHOLARLY COMMONS

Doctoral Dissertations and Master's Theses

Fall 12-2017

# Low-Cost Wearable Head-Up Display for Flight General Aviation

Pavan K. Chinta
*Embry-Riddle Aeronautical University*

Follow this and additional works at: https://commons.erau.edu/edt

Part of the Aviation Safety and Security Commons

# LOW-COST WEARABLE HEAD-UP DISPLAY
# FOR LIGHT GENERAL AVIATION

A Thesis

Submitted to the Faculty

of

Embry-Riddle Aeronautical University

by

Pavan K. Chinta

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Aerospace Engineering

December 2017

Embry-Riddle Aeronautical University

Daytona Beach, Florida

# LOW-COST WEARABLE HEAD-UP DISPLAY
# FOR LIGHT GENERAL AVIATION

By

Pavan K. Chinta

A Thesis prepared under the direction of the candidate's committee chairman, Dr. Richard "Pat" Anderson, Department of Aerospace Engineering, and has been approved by the members of the thesis committee. It was submitted to the School of Graduate Studies and Research and was accepted in partial fulfillment of the requirements for the degree of Master of Science in Aerospace Engineering.

THESIS COMMITTEE

Chairman, Dr. Richard "Pat" Anderson
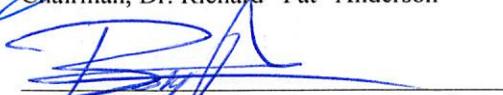
Co-Chair, Dr. Borja Martos

Member, Dr. Richard Prazenica
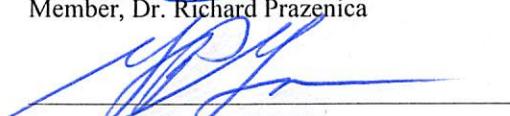
Member, Prof. Glenn Greiner

Graduate Program Coordinator, Dr. Magdy Attia        11.30.2017
                                                      Date

Dean of College of Engineering, Dr. Maj Mirmirani     12/1/2017
                                                      Date
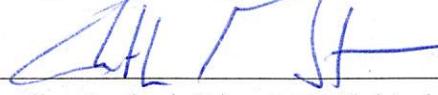
Vice Chancellor, Academic Support, Dr. Christopher Grant    11/30/17
                                                      Date

# ACKNOWLEDGEMENTS

Pavan K. Chinta

November 13th 2017, Daytona Beach, Florida

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ACRONYMS

AHRS      =      Attitude and Heading Reference System

AOA      =      Angle-of-Attack

AOS      =      Angle-of-Sideslip

AR      =      Augmented Reality

COTS      =      Commercial-off-The-Shelf

CT      =      continuous-time

DAS      =      Data Acquisition System

DT      =      discrete-time

EFRC      =      Eagle Flight Research Center

EFVS      =      Enhanced Flight Vision System

FAA      =      Federal Aviation Administration

FPA      =      Flight-Path Angle

FPV      =      First-Person View

GA      =      General Aviation

GAJSC      =      General Aviation Joint Steering Committee

HGT      =      Head-Up Guidance Technology

HMD      =      Head-Mounted Display

HUD      =      Head-Up Display

IMU      =      Inertial Measurement Unit

LOC      =      loss-of-control

| | | |
|---|---|---|
| MEMS | = | Microelectromechanical Systems |
| MR | = | Mixed Reality |
| NTSB | = | National Transportation and Safety Board |
| OOP | = | owner/operator/pilot |

## NOMENCLATURE

$f_h$ = Complementary High-Pass Filter Function

$f_l$ = Complementary Low-Pass Filter Function

$g$ = Gravitational Acceleration

$p$ = Roll Rate

$q$ = Pitch Rate

$s$ = Lapace Variable

$T$ = Complementary Filter Time-Constant

$V$ = Freestream velocity

$V_N, V_E, V_D,$ = North Velocity, East Velocity, Down Velocity with respect to the Earth

$\alpha$ = Angle-of-Attack

$\alpha_f$ = Complementary Filtered Angle-of-Attack

$\alpha_i$ = Angle-of-Attack from Inertial Source

$\alpha_g$ = Angle-of-Attack due to Wind Gust

$\alpha_v$ = Angle-of-Attack from Air Data Source

$\beta$ = Angle-of-Sideslip or Head-Orientation Filter's First Tuning Gain

$\beta_f$ = Complementary Filtered Angle-of-Sideslip

$\beta_i$ = Angle-of-Sideslip from Inertial Source

$\beta_g$ = Angle-of-Sideslip due to Wind Gust

$\beta_v$ = Angle-of-Sideslip from Air Data Source

$n_z$ = Normalized Acceleration in Z-Direction

| | | |
|---|---|---|
| $n_y$ | = | Normalized Acceleration in Y-Direction |
| $\psi$ | = | Aircraft Heading |
| $\theta$ | = | Aircraft Pitch Attitude |
| $\varphi$ | = | Aircraft Roll Attitude |
| z | = | Variable in the Discrete-Time Domain |

# ABSTRACT

Chinta, Pavan MSAE, Embry-Riddle Aeronautical University, December 2017. Low Cost Wearable Head-Up Display for Light General Aviation.


A low-cost wearable Commercial-off-The-Shelf (COTS) Augmented Reality (AR) Head-Up Display (HUD) system is designed, successfully reduced to practice, and flight tested. The system is developed based on the need for a technology that improves loss-of-control (LOC) safety in the General Aviation (GA) sector. The accuracy of the flight-path based system is determined to be within a degree of the truth source. The repeatability of the data from the COTS system is excellent. A complementary filter is proposed for air data flow angles and successfully flight tested for straight and level flight, dynamic maneuvering, and atmospheric turbulence, provided that a reasonably accurate lift curve is determined. A novel accelerometer method is proposed for estimating the relative pitch attitude estimation of the pilot's head. The method is evaluated on the ground and in flight, and is shown to be superior to other commercially available solutions. The HUD system is shown, through various test points, to make flying more intuitive and efficient, thereby affecting the GA LOC. In all the performed tasks, experienced and inexperienced pilots are used to fly the aircraft and evaluate the technology.

# 1. INTRODUCTION

One of the key aspects of the aviation sector is its cautious attitude towards accepting new technologies, primarily due to the ever rising costs of certification. In addition, even after a technical evolution comes to the market, users typically wait "for the technology to age [or be proven safe] before accepting it" (Motors, 2013). For example, in most military and commercial aircraft, there is a dedicated instrument that shows Angle-Of-Attack (AOA), uses this aeronautical concept to warn pilots of a potential stall. In GA, on the other hand, the use of AOA indicators is almost nonexistent and most GA aircraft lack such an indication (Jimenez, 2013). In 2014, the FAA approved the design requirements for installation of AOA indicators on small aircraft by simplifying design approval requirements (News, 2014), causing a surge in the use of AOA indicators in the GA sector (Bertorelli, 2016).

In this thesis, the importance of AOA is recognized along with a need for a reliable method to obtain an accurate AOA signal. The limitation of a recent, innovative technology for obtaining such an accurate signal is established and attempted to be resolved. The technology obtains the signal using an air mast, which vulnerable to producing noisy data in severe atmospheric turbulence. The problem is dealt with effectively by using the popular technique of complementary filtering by combining the inertial signal with air data signal to obtain an overall smooth AOA signal.

Further, there have been more changes in regulations to encourage technologies previously only limited to military and commercial sectors, to enter into the GA sector (Dorr, 2016). In light of these changes, this thesis aims to conduct research to develop and test the usability of a wearable HUD (or simply head-mounted display) using commercially off-the-shelf products to improve a pilot's ability to fly by making flying more intuitive and effective.

In seeking a suitable commercially available HUD System, smart AR glasses were an attractive option due to their inexpensive price point and ready accessibility due to the ever-growing demand in the consumer market (Global Industry Analysts, Inc., 2015). Such a system can be designed with a minimal footprint on the aircraft in terms of having a permanent interference with existing systems on the aircraft (Thales, 2015). In addition, the system only provides advisory information, not replacing any information provided by the existing systems; therefore, the incurring certification costs would be minimal. (Bertorelli, 2013)

Among the different smart AR glasses, the Epson BT-200 seemed like a viable option based on factors such as size, compactness, functionality, and use of the device (Epson, 2014). In this thesis, a fully functional Android application is developed, which obtains information from an onboard Data Acquisition System (DAS) via a network connection and appropriately displays the same information to the pilots. The effectiveness of the system is measured through various flight-test points and the benefit is seen to be very apparent, as an inexperienced subject pilot is able to perform flight maneuvers under tight tolerances with assistance from the developed system.

A key challenge with developing a Head-Mounted Display (HMD) is the ability to estimate the pilot's *head pose* in order to correctly overlay information such as *pitch attitude* and *FPA* with the outside world. Key concerns for the *head-tracking* system are the accuracy, repeatability, and latency of the measurements (Newman & Haworth). Several of the commercially available HMDs have used different techniques to achieve the required qualities for the *head-tracking* system. For example, the AERO GLASS team chose to use a tiny infrared camera on the glasses which estimates the *head pose* by using a lightweight plate of retro-reflective markers attached to the cockpit frame above the pilot's seat. The measurement of the pilot's *head pose* inside the cockpit is then combined with the aircraft's attitude to allow for the correct overlay of visuals under arbitrary viewing and flying directions (Bimber, 2015).

In order to develop the HMD, a novel *head-tracking* solution is proposed to attain the required accuracy, repeatability and latency. The solution relies on measuring the angle between two vectors in different frames of reference. The first unit vector is aligned with the body-axis of the aircraft while the second is aligned with the axis of the pilot's head. The vector along the body-axis of the aircraft comes from a Microelectromechanical Systems (MEMS) accelerometer affixed with the aircraft. The vector along the axis of the pilot's head comes from the accelerometer build into the AR glasses. The computation of the *head pose* occurs in the glasses, where the *head pose* is combined with the aircraft attitude to correctly overlay the information with the outside world.

Once the accuracy, repeatability and latency of the *head-tracking* solution is verified through various flight-test points, the effectiveness of HUD system is measured in other areas of flight-test performance. The flight tests show positive results for the HUD in all conditions and the system promises to be a potential safety device for the future pilots.

## 1.1    Motivating Factors

In 1990, the Flight Safety Foundation conducted a study on Head-Up Guidance Technology (HGT) to improve aviation safety. The study concluded that the HGT could have likely prevented a significant portion of LOC accidents that occurred between 1959 and 1989 in the civil jet transport category (Vandel & Weener, 2009), as shown in Figure 1.1 (only parameters relevant to this thesis are listed).
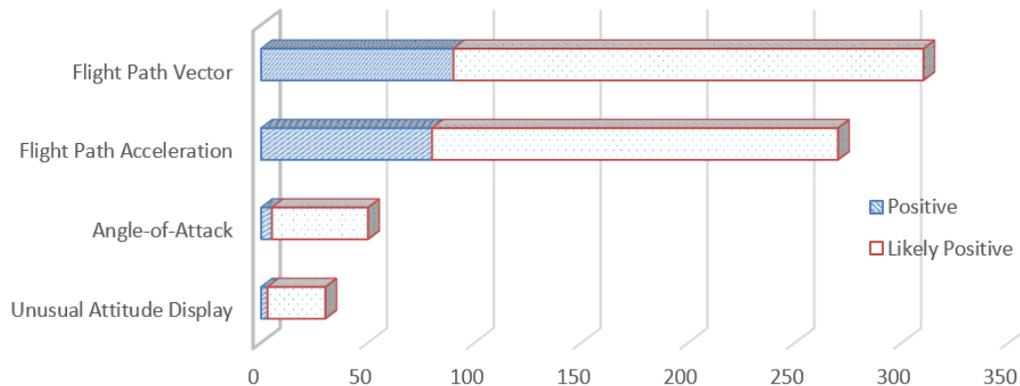


**Figure 1.1 Relevant Parameters with Positive Impact on Accidents between 1959 and 1989 in the Civil Jet Transport Category (Vandel & Weener, 2009).**

Unfortunately, the rate of GA accidents and fatalities is the highest of all aviation categories, being nearly constant for the past decade. According to the National Transportation and Safety Board (NTSB), in 2010 GA accidents accounted for 96 percent of all aviation accidents, 97 percent of fatal aviation accidents, and 96 percent of all fatalities for U.S. civil aviation. In contrast, GA accounted for an estimated 51 percent of total flight time of all U.S. civil aviation in 2010. Other forms of aviation, such as transport category aircraft, have seen an accident rate that has been decreasing to historically low values. In these categories, the airplanes, pilots, equipment, training requirements, maintenance procedures and financial means are, to a large extent, homogenous. In these categories, safety improvements in the form of equipment or pilot training, once identified, are typically mandated. On the other hand, it is the opposite in GA, with a vast diversity in airplanes, pilots, aeronautical experience, training requirements, maintenance procedures and financial abilities. It is also critical to note that additional pilot training, equipment, and other known safety enhancing products historically are not mandated in a large portion of GA. Based on the NTSB statistics, the GA Joint Steering Committee (GAJSC), and other studies, a low cost HUD is recommended equipment to improve GA safety (NTSB, 2015).

Furthermore, in 2016 the Federal Aviation Administration (FAA) overhauled the airworthiness standards for small GA airplanes. The FAA estimates that the overall economic impact would be cost beneficial, as the new proposal attempts to streamline the approval of advancements for small GA aircraft. With this overhaul, a number of technologies previously available to the military and commercial markets are now beginning to be appear in GA. Examples of such technologies include Fly-by-Wire, Enhanced Vision, and HUD systems (Dorr, 2016).

Moreover, there is a growing trend of wearable AR, Mixed Reality (MR), and Virtual Reality (VR) glasses with the advent of Google glass in 2014. Today, there are a large number of sophisticated glasses available at an affordable price. Dramatic price reductions and technology improvements have opened up the possibility of using such technology in GA.

The following research suggests the design of a simple, effective, and affordable wearable HUD system that will affect GA LOC safety by making flying more intuitive and efficient. The HUD system is developed at the Eagle Flight Research Center (EFRC) and will be referred to as the EFRC HUD. The primary focus is the GA retrofit market with over 200,000 registered aircraft. Therefore, the basic premise of this research is twofold: (1) to develop a simple, effective and affordable wearable HUD and (2) to develop a HUD system that will make the owner/operator/pilot (OOP) want to incorporate the system into their aircraft operation. The only way to make the OOP want to do this is to provide a clear cost benefit that is specific to that person and airplane.

## 1.2    Technology Review

In this section, all the commercially available HUDs similar to that proposed in the thesis are briefly discussed. For the purpose of comparison, all similar products regardless of their target audience, are mentioned in Table 1.1 and discussed further along the section. The EFRC HUD is developed for a light GA aircraft that is capable of transmitting (via Wi-Fi or wired connection) the aircraft attitude, Flight-Path Angle (FPA) and AOA signals.

**Table 1.1 List of the Currently Available COTS HMDs.**

| Product | Company | Target Audience | Head-Tracking | Price (USD) |
|---|---|---|---|---|
| Brilliant Eyes | Aerocross Systems | GA | COTS Solution | < 2,000 |
| TopMax | Thales Group | Business/Commercial | IR Camera | 50,000 |
| Aero Glass | Aero Glass | GA | IR Camera | 14,000 |
| Skylens | Elbit Systems | Commercial | IR Camera | 50,000 |
| EFRC HUD | EFRC | GA | Accelerometer | < 1,000 |

### 1.2.1    Brilliant Eyes

Brilliant Eyes is a HMD developed by a startup company named Aerocross Systems, targeted towards the light GA market, shown in Figure 1.2. A prototype version was displayed at EAA's AirVenture in 2013 with a planned low-rate production by late 2014 for less than $2000. The initial design was proposed to only contain the "traditional six-pack of information – altitude, heading, airspeed, vertical speed, horizontal direction and a magnetic compass." In order to ensure conformity of the data presentation to the pilot's head position during flight, there is ongoing research on an appropriate *head-tracking* system. One method that is said to be potentially used is an inexpensive COTS Attitude and Heading Reference System (AHRS), as an on-chip sensor attached to the HMD (Croft, 2013).

**Figure 1.2 Brilliant Eyes, Proposed by Aerocross Systems at EAA's Air Venture in 2013 (Croft, 2013).**

Upon considering all the features such as cost and target audience, Brilliant Eyes is very similar to the EFRC HUD. The major difference however, is the *head-tracking* solution employed – the off-the-shelf AHRS unit suggested for the product has been tested in this project and found to inadequate.

### 1.2.2  TopMax

TopMax is a head-worn HUD developed by the Thales Group, with "a really light footprint within the aircraft compared a [traditional] HUD." The HUD consists of a main module with a display in front of one of the pilot's eyes showing all the flight symbology, plus a camera for orientation purposes, as shown in Figure 1.3. Although TopMax was developed as a solution to the business jets that cannot be equipped with HUD, there has been a lot of enthusiasm from the high-end jet market as well. The product is planned on being entered into production in 2019 (Thurber, 2015).

**Figure 1.3 Thales Group is Offering Demonstrations of its Head-Worn TopMax HUD on a Bose Noise-Cancelling Headset. (Thurber, 2015)**

Upon comparing TopMax with the EFRC HUD, apart from the difference of the target audience, TopMax uses a camera for orientation purposes while the EFRC HUD uses a simple MEMS accelerometer. As the camera requires "stickers on the overhead panel", the solution seems more sophisticated and expensive than the *head-tracking* solution presented in this thesis.

### 1.2.3  Aero Glass

The company Aero Glass used the Epson BT-200 glasses to design the initial prototype of the Aero Glass wearable HUD. The company took an approach very similar to that taken by the Thales Group for the *head-tracking* solution and used a tiny IR camera on the HUD, in conjunction with a removable, lightweight plate of retro-reflective markers attached to the cockpit frame above the pilot's seat, as shown in Figure 1.4.
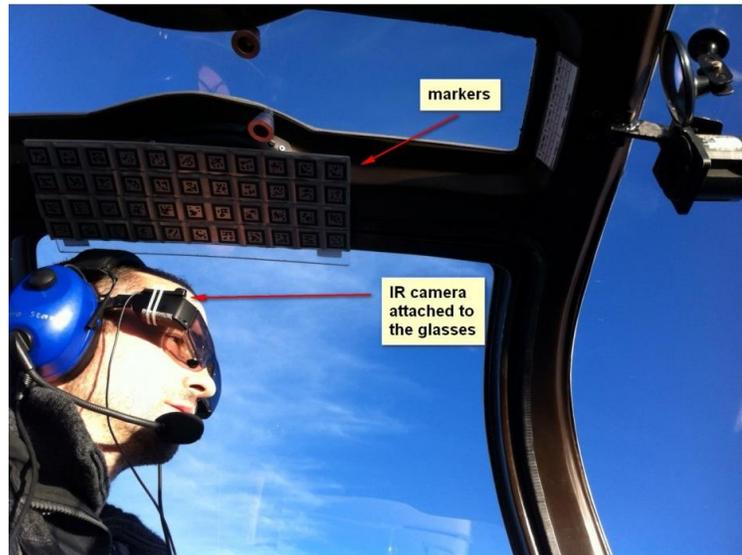
**Figure 1.4 The Aero Glass Wearable HUD with an IR Camera to Estimate the Head Orientation using a Lightweight Plate of Retro-Reflective Markers Attached to the Cockpit Frame Above the Pilot's Head (Bimber, 2015).**

However, the solution is reported to have certain reliability issues when used with the Epson BT-200. It is proposed that another pair of glasses such as the ODG R-7 are to be used in the future as they come with much superior sensors, and can potentially track the orientation simply using the onboard Inertial Measurement Unit (IMU) sensors (Bimber, 2015).

The *head-tracking* solution proposed in this thesis uses the sensors on the Epson BT-200 and is found to be adequately accurate and reliable for estimating the *head pose* in various flight maneuvers and atmospheric conditions.

### 1.2.4 Skylens

Skylens is a wearable HUD developed by Elbit Systems for Enhanced Flight Vision System (EFVS) applications, geared towards the business aviation market for retrofit and forward-fit application (Szondy, 2014). While retrofit refers to adding "a component to something that did not have it when manufactured" (Definition of retrofit in English:, n.d.), forward-fit refers to adding the component while the product is still in production (Motta, 2004).

The HUD unit comes equipped with an IR sensor which tracks orientation with the help of a tracker unit affixed to the roof of the cockpit. (Ebit Systems - Aerospace, 2014)

As seen from Figure 1.5, the entire unit is much more bulky and sophisticated than any of the other HUDs seen previously. The differences between the Skylens and the EFRC HUD are the target audience, *head-tracking* solution and the desired capabilities.



**Figure 1.5 Elbit Systems Skylens on ATR flight (Elbit Systems, 2016).**

In summary, there are currently several commercially available wearable HUDs in all sectors of the aviation market. Majority of the products have used an IR camera with a reflective marker board on the cockpit panel for orientation purposes, which has been reported to have reliability issues by some producers. Through this thesis, an economical, reliable and fully commercially-off-the-shelf solution is aimed to be developed, which although is comparable to those currently available in the market, it is distinct in the *head-tracking* solution employed. In other words, one of the goals of this project is to suggest a simple wearable HUD that uses an economical and reliable algorithm for *head-tracking* and effectively illustrates the benefit of the entire system for use in the light GA sector.

## 1.3    Research Hypothesis

Using FPA as the primary control aide, supplemented with AOA and traditional cockpit information makes performing tasks that otherwise require significant effort rather effortless, even under tight tolerances for inexperienced pilots.

In order to fly the tolerances outlined in the FAA's Airman Certification Standards (Federal Aviation Administration, 2017) using traditional cockpit instruments, a pilot must develop an *acquired* sense of intuition (as opposed to a *natural* sense), which typically requires consistent and disciplined flight training with a certified flight instructor. However, as previously noted, there is a vast diversity in aeronautical experience, training requirements, maintenance procedures and financial abilities in GA.

Consider a typical approach and landing scenario where the pilot is flying with traditional cockpit instruments. To perform a successful landing the pilot must infer the aircraft's flight path from a synthesis of the optical flow, and the relative perspective gradient or sight picture. On the other hand, a HUD with flight-path symbology can explicitly show the pilot where the aircraft is going relative to the landing environment. In this case, the pilot's mental workload is vastly reduced, allowing him to fly the aircraft with a sense of *natural* intuition (Goteman, Smith, & Dekker, 2007).

In fact, it is hypothesized that when a pilot closes the loop on FPA typical flight maneuvers can be performed with tight tolerances, less flight training, and reduced pilot workload. For example, climbing to a new altitude and trimming the aircraft with an altitude tolerance of ±50 feet, landing at a specific spot on the runway with a target tolerance of ±100 feet, or flying at a constant AOA in level flight with an AOA tolerance of ±1°.

In order to test the hypothesis, a subject pilot without flying experience is chosen to perform certain work intensive maneuvers, with and without assistance from the EFRC HUD. Although the subject pilot lacked the required flight training to perform these tasks, the subject pilot received sufficient ground school knowledge to properly interpret traditional cockpit instruments and the EFRC HUD.

## 2. THEORY AND BACKGROUND

In this section, the theory and algorithms required for the development of the EFRC HUD are explained.

### 2.1 Head-Up Display

"HUDs are transparent displays that present data without requiring users to look away from their usual viewpoints" (Stevenson, 2010). The origin of the name stems from a pilot being able to view information from the head positioned "up" and looking forward, instead of angled down at the instrument panel. Most HUDs are focused at optical infinity eliminating the need for the pilot to refocus between the HUD information and real world features viewed through the HUD (Rockwell Collins, January 2016).

Broadly speaking, HUDs can either be hard-mounted in an aircraft or worn by the pilot. Conventional HUDs are hard mounted in commercial and executive sectors. However, in light GA aircraft, a wearable HUD is more suitable. One difference between a conventional HUD and a wearable HUD is that the former has a fixed reference frame while the latter has a moving reference frame with respect to the airplane. The minimum requirements stated in 14 CFR Part 91 required HUDs to display airspeed, altitude, a horizon line, heading, and turn indicator (FAA, n.d.).

In this thesis, the coordinate system for the pilot's head (in neutral position) is selected so that it aligns with the body-axis of the aircraft, i.e. x-axis is positive along the line-of-sight, y-axis is along the right side and z-axis in positive towards the feet of the pilot, as shown in Figure 2.1.
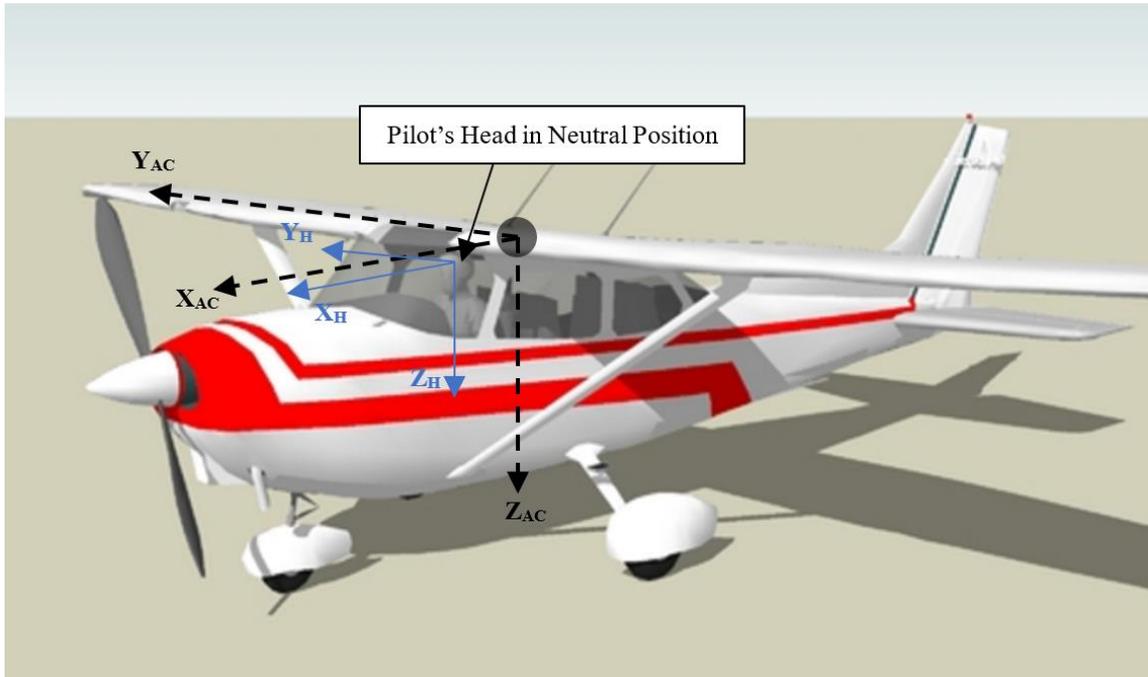
**Figure 2.1 Coordinate Axis Systems for the Aircraft and the Pilot's Head, Held in Neutral Position.**

If the aircraft is assumed to have an orientation of $\theta_{AC}$ (pitch), $\phi_{AC}$ (roll), $\psi_{AC}$ (heading) and the head is assumed to be oriented at $\theta_H$ (pitch), $\phi_H$ (roll), $\psi_H$ (heading); then, the orientation of the aircraft is displayed on the HUD with respect to the head. A sample wearable HUD (based on the F16 HUD design) is shown in Figure 2.2.
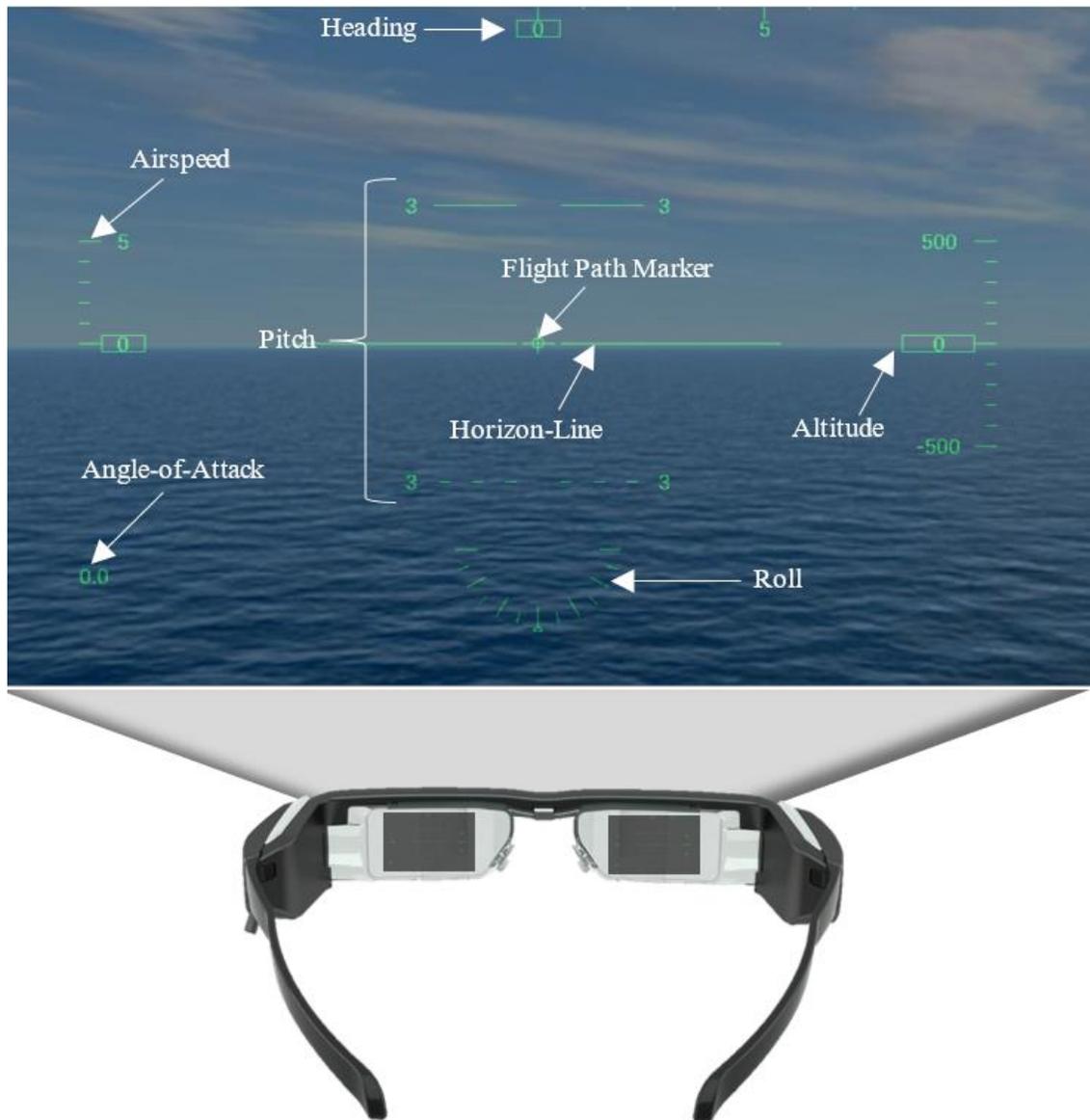
**Figure 2.2 A Sample Wearable HUD Based on the F-16 HUD Design.**

With such an axis system, if the pilot were to roll his head (rotation about the y-axis) counter-clockwise at angle $\phi_H$ and the aircraft is flying "wings-level" (i.e. $\phi_{AC} = 0$), the zero-pitch line and the pitch ladder are drawn at an angle of ($\phi_{AC}$ - $\phi_H$) on the HUD – the sign convention of the rotation is shown in Figure 2.3. Note that $\phi_{AC}$ is shown for completeness.
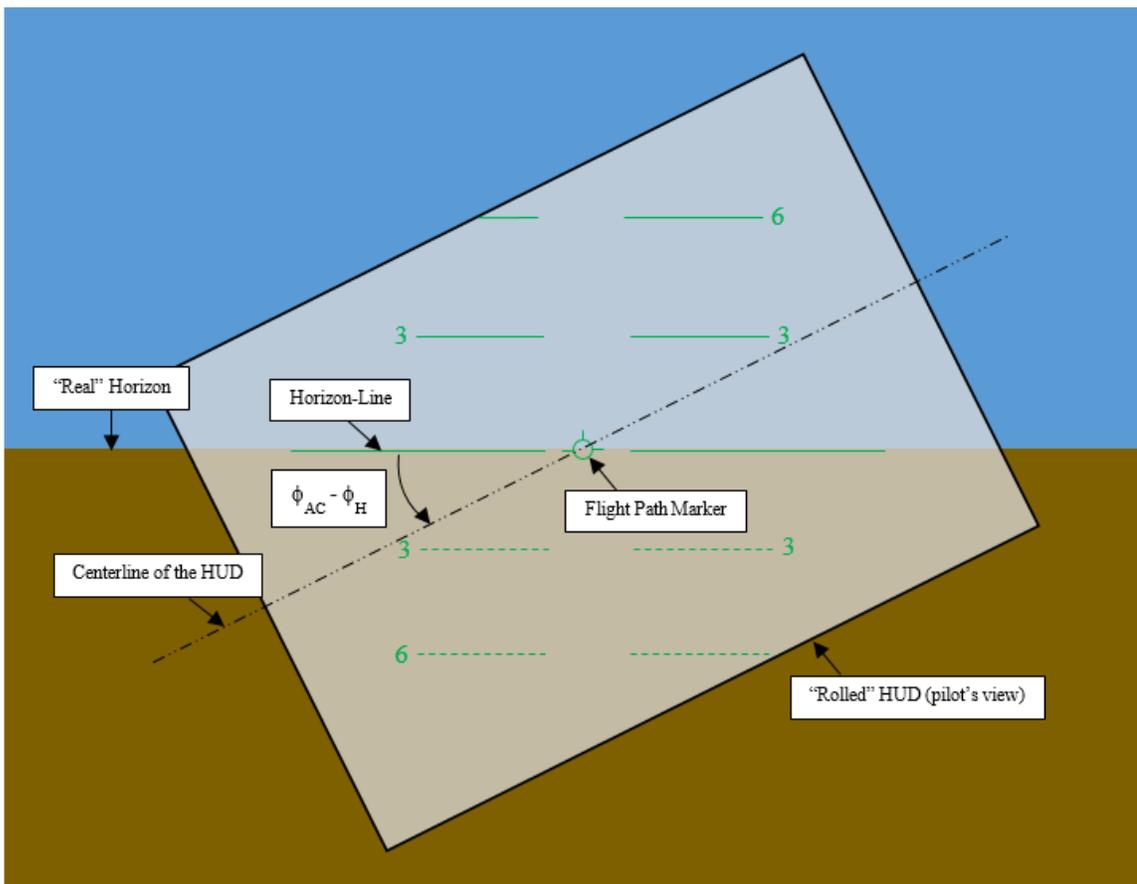
**Figure 2.3 Actual Horizon on the HUD with Rotation of the Pilot's Head.**

The same idea is further illustrated in Figure 2.4, where the glasses represent the pilot's head, which is rotated counter-clockwise around the y-axis. Note that the outside world is only shown for visualization purposes. Make note of how all the contents of the HUD are titled with the rotation of the glasses; however, the zero-pitch line (and the pitch ladder) are still aligned with the "real" horizon.
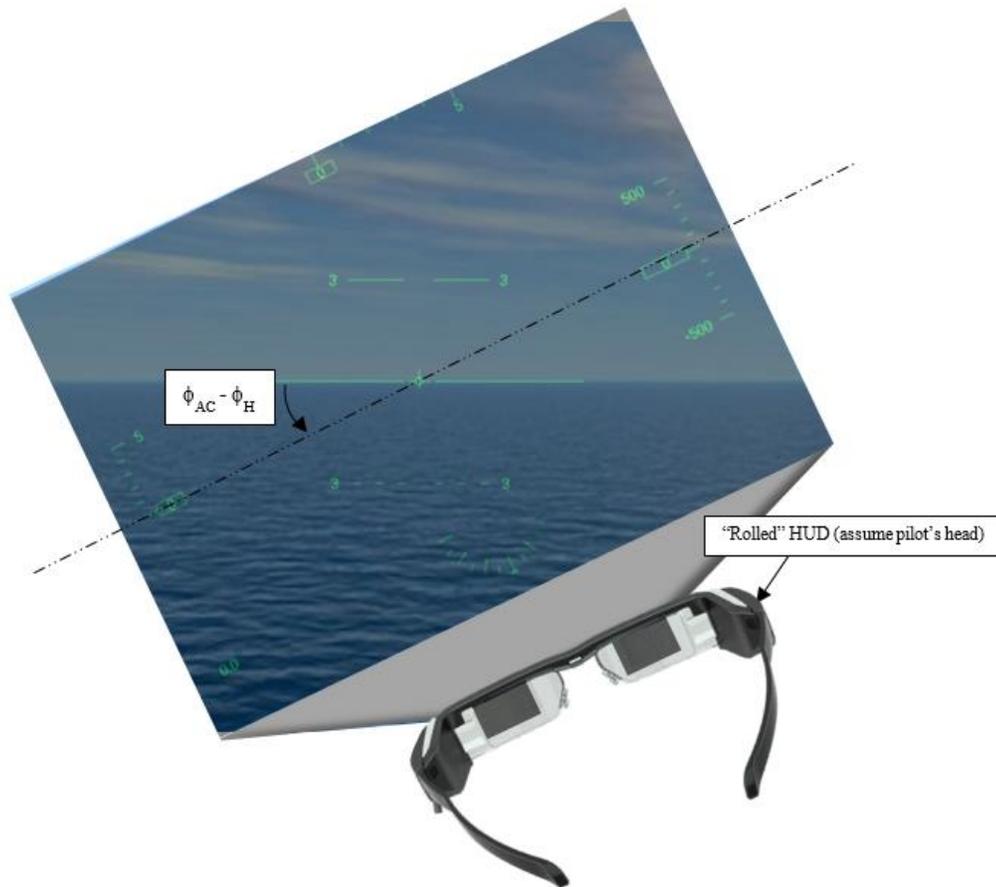
**Figure 2.4 Shifting of the HUD Contents with Rotation of the Pilot's Head.**

## 2.2 Angular Units Calculation

In any pair of smart AR glasses, the display is projected at a specific distance from the eyes. For use of such glasses as a wearable HUD, the following information is appropriately displayed: airspeed, altitude, roll, pitch, heading, flight-path marker and AOA. Among which, the pitch attitude and flight-path marker must conform to the outside world; therefore, the pixels per degree for the AR glasses must be known to correctly display graduations on the pitch ladder.

In order to understand what "conforming to the outside world" means, consider a situation where the user is looking with "eyes-leveled" with the horizon and a bird is flying above the user's head, at $6^o$ to the horizon, as shown in Figure 2.5.
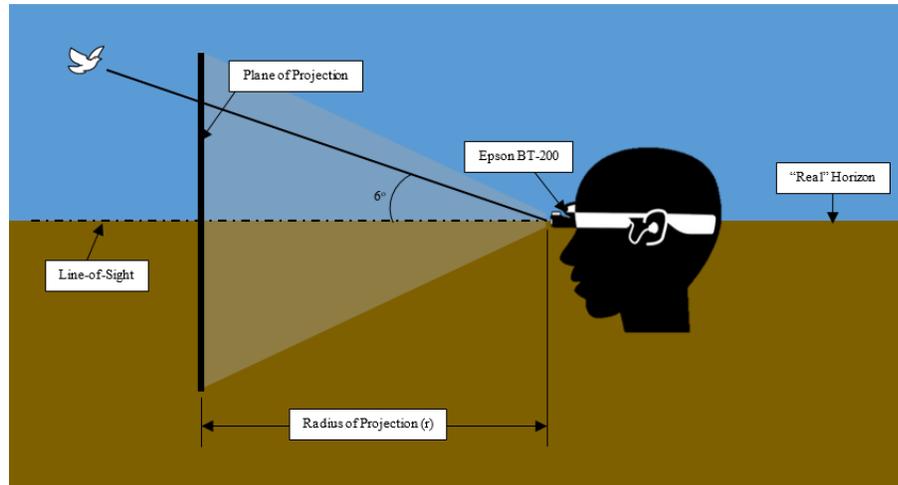


**Figure 2.5. Radius of Projection of the Epson BT-200 glasses.**

The First-Person View (FPV) for the same situation is shown in Figure 2.6. Note that, for illustration purposes, only the pitch ladder is shown on the HUD.
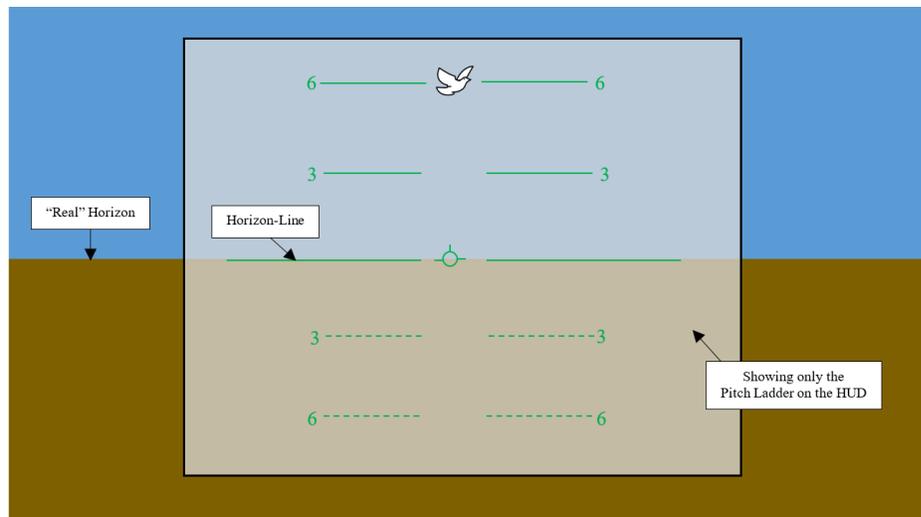


**Figure 2.6 First Person View of the HUD, with a Bird at $6^o$ above the Horizon.**

In the above illustration, to ensure that the bird is at 6º to the zero-pitch line, the angular resolution ($\omega_{BT200}$) must be known – the following procedure illustrates estimation of the same parameter.

According to the user manual of the Epson BT-200, the screen resolution is 960 x 540 pixels ($w \; x \; h$), the aspect ratio is 16:9 and the horizontal field-of-view, $hfov$, is 23º. The vertical field-of-view, $vfov$ (degrees), is calculated as follows.

$$vfov = \frac{hfov}{aspect \; ratio} = \frac{23}{16/9} = 12.9375^o$$

Next, the pixels per degree is calculated using the height of the screen (540 pixels) and the vertical field-of-view.

$$\omega_{BT200} = \frac{h}{vfov} = \frac{540}{12.9375} = 41.739 \; pixel/degree$$

The calculated angular resolution is used on the EFRC HUD to draw graduations on the pitch ladder and overlay the pitch cross-hair and the flight-path marker with respect to the outside world.

**2.3    Head-Tracking Algorithm**

In order to estimate the absolute orientation of the wearable HUD, an accurate and reliable sensor fusion algorithm is required for use on the built-in sensors such as gyroscope, accelerometer and magnetometer. However, no such algorithm is identified based on the required accuracy of ±2 degrees (of the orientation). Most of the available algorithms use accelerometers to detect the tilt of the head and smooth the signal using gyroscopic corrections, which works well in a quasi-static environment. However, in a dynamic environment, the accelerometer sensor becomes practically useless due to the distortion of the signal, resulting in an inaccurate attitude solution. (Ahmed & Tahir, 2016) Therefore, most wearable HUD manufacturers resort to estimating relative orientation of the pilot's head (i.e. with respect to the aircraft) instead of the absolute orientation. Majority of the manufacturers listed earlier in the document used an IR camera in conjunction with a retro-reflective marker board to estimate the orientation of the pilot's head. However, such a head-tracking setup results in a high price point, clearly reflected in the difference in the prices between the Aero Glass and the Brilliant Eyes – while Aero Glass used the IR camera solution, Brilliant Eyes suggested using a COTS solution.

In this thesis, a slightly different approach is suggested for estimating the head orientation with respect to the aircraft, using two MEMS accelerometers – one fixed with the aircraft and another fixed with the head.

Consider two accelerations, $a_{AC}$ and $a_H$, which have components in the aircraft-body reference frame and pilot's head reference frame respectively.

$$\vec{a}_{AC} = a_{x_{AC}}\hat{i}_{AC} + a_{y_{AC}}\hat{j}_{AC} + a_{z_{AC}}\hat{k}_{AC} \tag{1}$$

$$\vec{a}_H = a_{x_H}\hat{i}_H + a_{y_H}\hat{j}_H + a_{z_H}\hat{k}_H \tag{2}$$

The procedure below is used to estimate the head orientation with respect to the aircraft, from the two accelerations shown in Equations (1) and (2).

Normalize the accelerations using the corresponding Euclidean norm to obtain unit vectors in the respective reference frames, as shown in Equations (3) and (4).

$$\hat{a}_{AC} = \frac{a_{x_{AC}}\hat{i}_{AC} + a_{y_{AC}}\hat{j}_{AC} + a_{z_{AC}}\hat{k}_{AC}}{\sqrt{a_{x_{AC}}^2 + a_{y_{AC}}^2 + a_{z_{AC}}^2}} \tag{3}$$

$$\hat{a}_H = \frac{a_{x_H}\hat{i}_H + a_{y_H}\hat{j}_H + a_{z_H}\hat{k}_H}{\sqrt{a_{x_H}^2 + a_{y_H}^2 + a_{z_H}^2}} \tag{4}$$

The acceleration in the aircraft body-fixed reference frame, $\hat{a}_{AC}$, can then be converted to the inertial reference frame using the rotation matrix obtained from ZYX rotation. The rotation matrix is shown in Eq. (6), where $\phi_{AC}$, $\theta_{AC}$, and $\psi_{AC}$ are the aircraft's Euler angles about x, y and z-axes respectively (Stengel, 2004).

$$\hat{a}_I = R^{-1}\hat{a}_{AC} \tag{5}$$

$$R = \begin{bmatrix} c\theta_{AC}c\psi_{AC} & c\phi_{AC}s\psi_{AC} & -s\theta_{AC} \\ -c\phi_{AC}s\psi_{AC} + s\phi_{AC}c\psi_{AC}s\theta_{AC} & c\phi_{AC}c\psi_{AC} + s\phi_{AC}s\theta_{AC}s\psi_{AC} & s\phi_{AC}c\theta_{AC} \\ s\phi_{AC}s\psi_{AC} + c\phi_{AC}s\theta_{AC}c\psi_{AC} & -s\phi_{AC}c\psi_{AC} + c\phi_{AC}s\theta_{AC}s\psi_{AC} & c\phi_{AC}c\theta_{AC} \end{bmatrix} \tag{6}$$

Consider the 2D rotation of $\hat{a}_I$ onto $\hat{a}_H$ by an angle θ, on a plane with normal along $\hat{a}_I$ x $\hat{a}_H$, as shown in Figure 2.7.
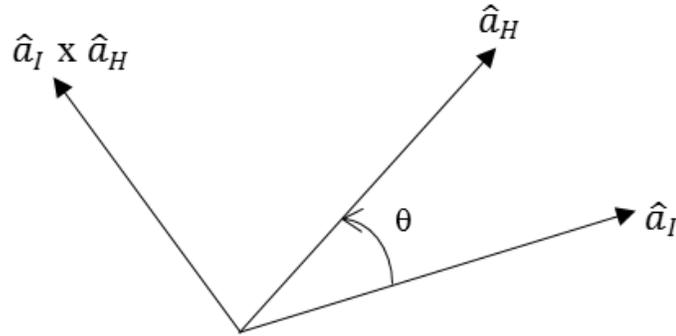
**Figure 2.7 2D Rotation of âᵢ to âₕ by an Angle θ, on a plane with normal along âᵢ x âₕ.**

The rotation matrix for the 2D rotation is shown in Eq. (7).

$$G = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \hat{a}_I \cdot \hat{a}_H & -||\hat{a}_I \text{ x } \hat{a}_H|| & 0 \\ ||\hat{a}_I \text{ x } \hat{a}_H|| & \hat{a}_I \cdot \hat{a}_H & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

The matrix G represents the rotation from $\hat{a}_I$ to $\hat{a}_H$ in the basis set (say Q) consisting of the following vectors.

1. Normalized vector projection of $\hat{a}_H$ onto $\hat{a}_I$

$$\hat{u} = \frac{(\hat{a}_I \cdot \hat{a}_H)\hat{a}_{AC}}{||(\hat{a}_I \cdot \hat{a}_H)\hat{a}_I||} = \hat{a}_I = \begin{bmatrix} a_{I_x} \\ a_{I_y} \\ a_{I_z} \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{8}$$

2. Normalized vector rejection of $\hat{a}_H$ on $\hat{a}_I$ (which is orthogonal to the vector projection of $\hat{a}_H$ onto $\hat{a}_I$)

$$\hat{v} = \frac{\hat{a}_H - (\hat{a}_I \cdot \hat{a}_H)\hat{a}_I}{||\hat{a}_H - (\hat{a}_I \cdot \hat{a}_H)\hat{a}_I||} = \begin{bmatrix} \dfrac{a_{I_x} - (a_{I_x}a_{H_x})a_{H_x}}{||a_{I_x} - (a_{I_x}a_{H_x})a_{H_x}||} \\[3mm] \dfrac{a_{I_y} - (a_{I_y}a_{H_y})a_{H_y}}{||a_{I_y} - (a_{I_y}a_{H_y})a_{H_y}||} \\[3mm] \dfrac{a_{I_z} - (a_{I_z}a_{H_z})a_{H_z}}{||a_{I_z} - (a_{I_z}a_{H_z})a_{H_z}||} \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \tag{9}$$

3. The cross product of $\hat{a}_H$ and $\hat{a}_I$ (which is orthogonal to both, the vector projection

and the vector rejection of $\hat{a}_H$ onto $\hat{a}_I$)

$$\hat{w} = \hat{a}_H \; x \; \hat{a}_I = \begin{bmatrix} a_{H_y}a_{I_z} - a_{H_z}a_{I_y} \\ a_{H_z}a_{I_x} - a_{H_x}a_{I_z} \\ a_{H_x}a_{I_y} - a_{H_y}a_{I_x} \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \tag{10}$$

The basis change matrix for the same basis is as follows. (Hefferon, 2017)

$$F = [u \; v \; w]^{-1} = \begin{bmatrix} u_1 & v_1 & w_1 \\ u_2 & v_2 & w_2 \\ u_3 & v_3 & w_3 \end{bmatrix}^{-1}$$

$$\Rightarrow F = \begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix}$$

Therefore, to rotate a vector in the inertial reference frame onto the head reference

frame, the vector must be right-multiplied by the rotation matrix U, shown in Eq. (11), to

change the basis set to Q, rotate the vector by the rotation matrix G and finally revert the

basis set to the original basis set (in that order).

$$U = F^{-1}GF = \begin{bmatrix} u_1 & v_1 & w_1 \\ u_2 & v_2 & w_2 \\ u_3 & v_3 & w_3 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix} \tag{11}$$

However, to rotate $\hat{a}_I$ onto $\hat{a}_H$ using the ZYX rotation sequence, the same rotation

matrix shown in Eq. (6) can be re-written in terms of the Euler angles of the pilot's head

as shown in Eq. (12), where $\phi_H$, $\theta_H$, and $\psi_H$ are angles about x, y and z-axes respectively.

$$U = \begin{bmatrix} c\theta_H c\psi_H & c\phi_H s\psi_H & -s\theta_H \\ -c\phi_H s\psi_H + s\phi_H c\psi_H s\theta_H & c\phi_H c\psi_H + s\phi_H s\theta_H s\psi_H & s\phi_H c\theta_H \\ s\phi_H s\psi_H + c\phi_H s\theta_H c\psi_H & -s\phi_H c\psi_H + c\phi_H s\theta_H s\psi_H & c\phi_H c\theta_H \end{bmatrix} \tag{12}$$

Comparing Eq. (11) and Eq. (12), Euler angles of the pilot's head can be estimated through the following equations.

$$\theta_H = \sin^{-1}(-U_{13})$$

$$U_{13} = u_1(u_3\cos\theta - v_3\sin\theta) + v_1(u_3\sin\theta + v_3\cos\theta) + w_1w_3$$

(13)

$$\phi_H = \sin^{-1}(U_{23}/\cos\theta_H)$$

$$U_{23} = u_2(u_3\cos\theta - v_3\sin\theta) + v_2(u_3\sin\theta + v_3\cos\theta) + w_2w_3$$

(14)

$$\psi_H = \sin^{-1}(U_{12}/\cos\phi_H)$$

$$U_{12} = u_1(u_2\cos\theta - v_2\sin\theta) + v_1(u_2\sin\theta + v_2\cos\theta) + w_1w_2$$

(15)

Most of the parameters shown on the HUD are conventionally available on an aircraft. The following two parameters however are not typically available on light GA aircraft and need to be computed specially for the HUD.

i)      AOA

ii)     Flight-Path Marker (FPM)

## 2.4   Angle-of-Attack

By definition, absolute AOA is the angle between the oncoming air or relative wind and the aircraft zero lift line. It is a direct measure of stall margin independent of aircraft weight.

Reference (Rogers, Martos, & Rodrigues, 2015) demonstrated a low-cost design for obtaining AOA using a COTS differential pressure probe with an accuracy between $0.25^\circ - 0.50^\circ$. The signal from an air data source is the sum of the inertial component, $i$, and the gust component, $g$, (due to wind gust or atmospheric turbulence), as shown in Eq. (16) and (17).

$$\alpha_v = \alpha_i + \alpha_g \tag{16}$$

$$\beta_v = \beta_i + \beta_g \tag{17}$$

In order to get a reliable AOA signal using a complementary filter, $\alpha_v$ (air data signal) is passed through a low-pass filter to suppress the high-frequency gust component. Simultaneously $\alpha_i$ (inertial signal) is passed through a complementary high-pass filter to smooth the overall signal without introducing phase delay. (Knotts & Priest)

The AOA complementary filter is shown in Eq. (18), where $\alpha_f$ represents the filtered $\alpha$, $f_l$ is the low-pass filter function with $\alpha_i + \alpha_g$ being the input with the gust component, $f_h$ is the high-pass filter function with $\int \dot{\alpha}_i \, dt$ being the input, and $\dot{\alpha}_i$ is the inertial time-varying AOA obtained from the IMU data.

$$\alpha_f = f_l(\alpha_i + \alpha_g) + f_h\left(\int \dot{\alpha}_i \, dt\right) \tag{18}$$

The initial condition for the integral in Eq. (18) is obtained using Eq. (19), where $\gamma$ is the inertial FPA, $\theta$ is the pitch angle, $\phi$ is the roll angle, and $\beta_i$ is the Angle-of-Sideslip (AOS) from the inertial source (Knotts & Priest). For the initial condition, AOS from the inertial source is approximated to be the AOS from the air-data source, i.e. $\beta_g \approx 0$ for the initial condition.

$$\alpha_i = \frac{-sin\gamma + sin\theta - \beta_i sin\phi}{cos\phi} \tag{19}$$

$$\dot{\alpha}_i \cong \frac{g}{V}(n_z + cos\,\phi) + q - p\beta_f \tag{20}$$

In Eq. (20), $g$ is the gravitational acceleration, $V$ is the freestream velocity, $n_z$ is the normalized acceleration in z-direction, $q$ is the pitch rate, $p$ is the roll rate and $\beta_f$ is the complementary filtered AOS, which is computed using Eq. (21), where $f_l$ is the low-pass filter function with $\beta_i + \beta_g$ being the input, $f_h$ is the high-pass filter function with $\int \dot{\beta}_\iota\, dt$ being the input, and $\dot{\beta}_\iota$ is the time-varying inertial AOS obtained from the IMU data, as shown in Eq. (22).

$$\beta_f = f_l(\beta_i + \beta_g) + f_h\left(\int \dot{\beta}_\iota\, dt\right) \tag{21}$$

$$\dot{\beta}_i \cong \frac{g}{V}(n_y + sin\,\phi) - r + p\alpha_f \tag{22}$$

In Eq. (22), $n_y$ is the normalized acceleration in the y-direction, and $r$ is the yaw rate. Note that Eq. (20) and Eq. (22) assume that $\theta \approx 0$ (small angle assumption) and $\dot{V} \approx 0$ (cruise flight assumption).

There are at least three values of AOA that are of particular interest to pilots, viz., maximum endurance, maximum range and Carson cruise which are used symbolically in the approach and cruise modes of the EFRC HUD.

### 2.4.1 Maximum Endurance

The maximum endurance AOA represents the AOA where the aircraft consumes the least amount of power. When holding for weather or traffic to clear, it is common to fly at the best endurance condition. Also, at angles of attack beween stall and best endurance, the aircraft is operating on the backside of the power required curve, where the aircraft does not have speed stability. Providing accurate information of AOA that clearly depicts this regime is useful in preventing departure from controlled flight.

### 2.4.2 Maximum Range

The AOA for maximum lift to drag ratio corresponds to maximum range for a piston engine propeller aircraft. This AOA does not change with density altitude, weight, load factor, etc. Hence, in a fuel critical situation and/or an engine failure situation it is critical that the pilot have accurate access to this AOA.

### 2.4.3 Carson Cruise

The Carson cruise AOA represents the most efficient way to fly fast with the least increase in fuel consumption. With today's rising fuel costs, it certainly is of significant interest.

## 2.5    Complementary Filter

As suggested previously, obtaining an accurate $\alpha$ signal can be challenging because most GA COTS indicators do not have the required accuracy for use as a control aide. Reference (Rogers, Martos, & Rodrigues, 2015) suggests a simple, low-cost design for obtaining $\alpha$ using a COTS differential pressure probe with an accuracy between $0.25^o$ – $0.50^o$. However, the accuray of an air data $\alpha$ signal is degraded in the presence of atmospheric turbulence.

A complementary filter is a filtering technique used to combine similar data from different sensors. In this research, complementary filter will be used to account for atmospheric turbulence, where information from air data sensors is combined with that from inertial sensors. This technique has been successfully used in the United States Air Force Total In-Flight Simulator (TIFS) aircraft to produce high quality $\alpha$ and $\beta$ signals for flight control systems (Knotts & Priest).

A typical complementary filter has a low-pass filter and a corresponding high-pass filter, which combine data from two different sources to produce a blended signal with little to no phase lag or amplitude change. The zero-phase lag and unity amplitude change in the output signal is one the major advantages of a complementary filter, illustrated in Figure 2.8.
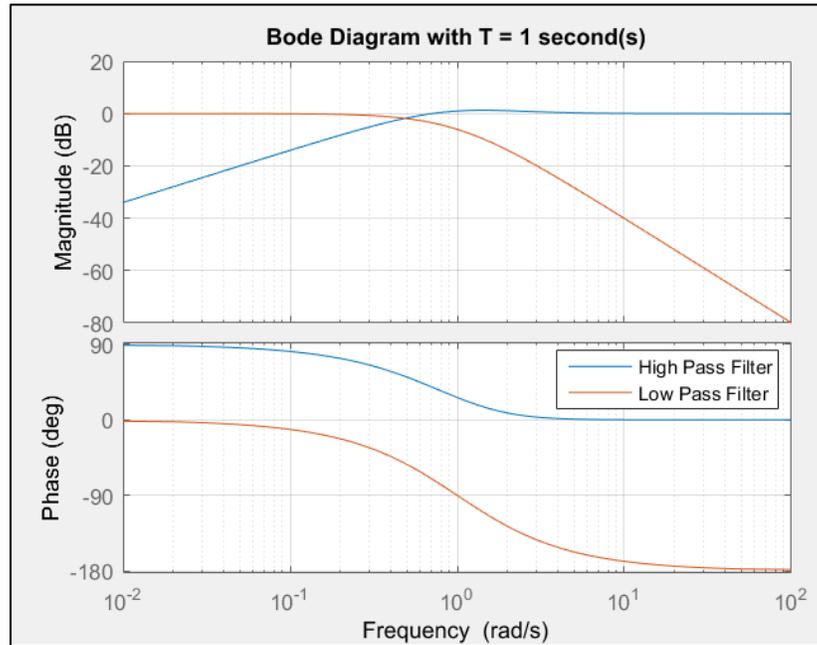
**Figure 2.8 Bode Plot Illustrating Zero Amplitude Change and Zero Phase Delay for T = 1 sec using the Complementary Filter in Eq. (23) and Eq. (24).**

The specific form of the complementary filter used in this thesis is shown in Eq. (23) and (24), where $f_l$ is the low-pass filter function, $f_h$ is the complementary high-pass filter function, $T$ is the filter time-constant, and $s$ is the Laplace variable. The final output of the complementary is the sum of that from the low-pass filter and the high-pass filter, as shown in Eq. (25), where $Y$ is the output signal. Additional information on the design of this filter can be found in Ref. (Knotts & Priest).

$$f_l = 1/(Ts + 1)^2 \tag{23}$$

$$f_h = 2Ts\left(\frac{Ts}{2} + 1\right)/(Ts + 1)^2 \tag{24}$$

$$Y_{Complementary-Filtered} = Y_{Low-Pass} + Y_{High-Pass} \tag{25}$$

In the implementation of the complementary filter, the AOA and AOS pressure sensors provide a steady state and low frequency content. As the vane information is degraded in the presence of wind gusts and turbulence, accelerometers and gyros, sensors which are not affected by external gusts, provide the high frequency content.

### 2.5.1   Discrete Form

The complementary filter shown in Eq. (23) and (24), can be implemented in a software such as MATLAB; however, in order to implement these on C++ code, the continuous-time (CT) domain form must be converted to a discrete-time (DT) domain form, as illustrated below using *Tustin* approximation.

*Tustin Approximation: Low-Pass Filter*

In order to convert a *Laplace* transfer function, in continuous-time domain, to the DT domain, the *Tustin* approximation is a common method, as shown in Eq. (26), where $z$ the DT variable and $T$ is is the sampling period. (Starr, 2006)

$$s = \frac{2}{T} * \frac{z-1}{z+1} \tag{26}$$

The low-pass filter of the complementary filter from Eq. (23), is shown below.

$$H(s)_{Low-Pass} = \frac{1}{(\tau s + 1)^2} \tag{27}$$

Substituting Eq. (26) into Eq. (28), the low-pass filter is converted to DT domain as shown in (28), where $Y(t)$ is the output signal and $U(t)$ is the input signal. For example, for a low-pass filter on AOA, $U(t)$ is the input AOA (say, obtained either from an AOA vane), and $Y(t)$ is the filtered AOA.

$$H(z,T)_{Low-Pass} = \frac{Y(t)}{U(t)} = \frac{1}{\left(\tau * \frac{2}{T} * \frac{z-1}{z+1} + 1\right)^2} = \frac{(z+1)^2}{(\frac{2\tau}{T}(z-1) + z + 1)^2}$$

$$= \frac{(z+1)^2}{(z(\frac{2\tau}{T} + 1) + 1 - \frac{2\tau}{T})^2}$$

Then by cross-multiplying, Eq. (28) is obtained.

$$Y * z^2 \left(\frac{2\tau}{T} + 1\right)^2 + Y * 2z * \left(\frac{2\tau}{T} + 1\right)\left(1 - \frac{2\tau}{T}\right) + Y * \left(1 - \frac{2\tau}{T}\right)^2 \tag{28}$$

$$= U * z^2 + U * 2 * z + U$$

Multiplying throughout by $z^{-2}$, and letting $\zeta = \frac{2\tau}{T}$, the above equation can be re-written as shown in Eq. (29).

$$Y * (\zeta + 1)^2 + Y * z^{-1} * 2(\zeta + 1)(1 - \zeta) + Y * z^{-2} * (1 - \zeta)^2 \tag{29}$$

$$= U + U * 2 * z^{-1} + U * z^{-2}$$

Equation (29) can be re-written as a difference equation, as shown in Eq. (31), using the notation shown in Eq. (30).

$$z^{-2} * Y(t) \equiv Y_n \text{ and } z^{-2} * U(t) \equiv U_n$$

$$=> z^{-1} * Y(t) \equiv Y_{n+1} \text{ and } z^{-1} * U(t) \equiv U_{n+1} \tag{30}$$

$$=> Y(t) \equiv Y_{n+2} \text{ and } U(t) \equiv U_{n+2}$$

$$Y_{n+2} = \frac{(U_{n+2} + U_{n+1} * 2 + U_n) - 2 * Y_{n+1} * (\zeta + 1)(1 - \zeta) - Y_n * (1 - \zeta)^2}{(\zeta + 1)^2} \tag{31}$$

In Eq. (31), the subscript $n$ stands for the curent time-step while $n + 1$ and $n + 2$ are one time-step and two time-steps ahead, respectively.

*Tustin Approximation: High-Pass Filter*

The same procedure shown above for low-pass filter is applied to the high-pass filter, as shown below.

$$H(s)_{High-Pass} = \frac{2\tau s\left(\frac{\tau s}{2} + 1\right)}{(\tau s + 1)^2} \tag{32}$$

Substituting the *Tustin* approximation from Eq. (26) into Eq. (27), the high-pass filter is converted to DT domain, as shown in Eq. (33).

$$H(z,T)_{High-Pass} = \frac{Y(t)}{U(t)} = \frac{2 * \tau * \frac{2}{T} * \left(\frac{z-1}{z+1}\right) * (\frac{\tau}{2} * \frac{2}{T} * \frac{z-1}{z+1} + 1)}{\left(\tau * \frac{2}{T} * \frac{z-1}{z+1} + 1\right)^2} \tag{33}$$

Multiplying by $z^{-2}$ and letting $\zeta = \frac{2\tau}{T}$, Eq. (33) can be re-written, as shown in Eq. (34).

$$Y(t) * (\zeta + 1)^2 + z^{-1} * Y(t) * 2 * (\zeta + 1) + z^{-2} * Y(t) * (1 - \zeta)^2$$

$$= 2\zeta \left(U(t) * \left(\frac{1}{2}\zeta + 1\right) + z^{-1} * U(t) * (-\zeta) + z^{-2} * U(t) * \left(\frac{1}{2}\zeta - 1\right)\right) \tag{34}$$

Equation (34) can be re-written as a difference equation, as shown in Eq. (35), using the notation shown in Eq. (30).

$$Y_{n+2} = \frac{2 * \zeta * \left(U_{n+2} * \left(\frac{1}{2}\zeta + 1\right) + U_{n+1} * (-\zeta) + U_n * \left(\frac{1}{2}\zeta - 1\right)\right)}{(\zeta + 1)^2}$$

$$+ \frac{-Y_{n+1} * 2 * (\zeta + 1) - Y_n * (1 - \zeta)^2}{(\zeta + 1)^2} \tag{35}$$

## 2.6   Flight-Path Angle

The definition of FPA varies based on the reference frame. In the air-mass referenced frame, it is defined as the angle between the flight-path vector (the velocity vector) and the local atmosphere, as shown in Eq. (36), where $\theta$ is the pitch attitude and $\alpha$ is the AOA.

$$\gamma_a = \theta - \alpha \tag{36}$$

In the inertial-referenced frame, it is the angle between the flight-path vector and the horizon, also known as the climb (or descent) angle. The two reference frames coincide when there is no wind or vertical air movement. The wind triangle shown in Figure 2.9 illustrates the difference between the inertial FPA and the air-mass referenced FPA (Beard & McLain, 2012).
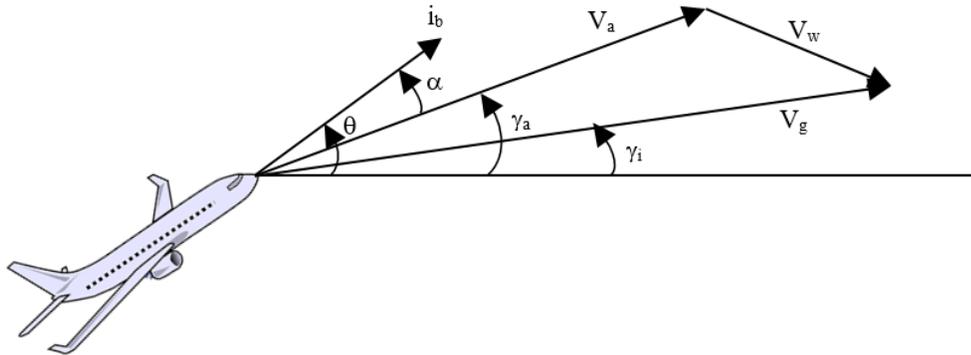


**Figure 2.9 The Wind Triangle Projected onto the Vertical Plane (Beard & McLain, 2012).**

Typically, the inertial-referenced flight-path angle is displayed on a HUD. Therefore, the inertial-referenced flight-path angle is used in the EFRC HUD. It is calculated using Eq. (37), where $V_N$, $V_E$, $V_D$ are the north, east, and down velocity components with respect to the earth.

$$\gamma_i = \tan^{-1}(-V_D/\sqrt{V_E{}^2 + V_N{}^2}) \tag{37}$$

One of the most important innovations that a HUD introduces to the traditional GA cockpit is the concept of out the window flight-path flying. With Flight-Path Marker (FPM) displayed on the HUD, the pilot is in direct control of the flight-path. Therefore, flight-path flying automatically reflects the effects of cross winds, AOA, drag, thrust and other factors which impact the dynamic state of the aircraft. For example,

- To maintain a constant altitude flight, the pilot can fix the FPM on the horizon line.

- To maintain a constant glide path, the pilot can hold the FPM at a fixed position with respect to the horizon line.

- In the presence of wind, the pilot can align the FPM with a fixed spot on the runway.

- In cruise, the pilot can rely on the FPM to confirm that the aircraft can safely avoid a thunderstorm.

## 2.7   Areas of Focus

In this section, the flight phase performance, certification costs, and atmospheric turbulence issues related to the design of the EFRC HUD are discussed. In each area, relevant problems are identified and practical solutions are proposed. These include post-takeoff, cruise, and landing phases.

### 2.7.1  Flight Phase Performance

A major benefit of a HUD over traditional cockpit instruments is the improvement in flight phase performance. There are at least three phases of flight that would greatly benefit from the use of a HUD.

### 2.7.2  Post-Takeoff Phase

One of the most common maneuvers performed inflight is acquiring straight and level flight after a climb segment. Here, a pilot closes the loop on airspeed, altitude and pitch, while trying to control the aircraft flight-path angle. However, with a HUD, the pilot can directly close the loop on the flight-path angle, thereby requiring less workload and training to perform the maneuver.

### 2.7.3  Cruise Phase

In 1971, NASA conducted a study to evaluate $\alpha$ as a primary flight parameter based on two considerations: (1) AOA is a direct measure of stall margin independent of aircraft weight (2) AOA responds quicker than airspeed to the pilot's control stick and throttle inputs. However, the expected advantages were negated by the inherent aerodynamic characteristics of the airplane, such as the Phugoid dynamic mode and directional-control stability. For example, "[t]he performance in low-speed maneuvers was roughly equivalent whether angle-of-attack or airspeed information was used. Maneuvers are normally performed primarily by using attitude control, with reference to airspeed or AOA as a secondary control parameter.

Although AOA responds faster to the pilots' pitch or throttle inputs, this expected advantage is more or less negated by the necessity of compensating for any induced Phugoid, or stated another way, AOA does not necessarily command the correct control inputs" (Gee, Gaidsick, & Enevoldson, 1971). The following research aims to leverage the expected advantages of AOA by combining it with FPA.

### 2.7.4 Landing Phase

The landing phase is difficult to master for student pilots because of the lack of intuition. Without the knowledge of where the aircraft is going to, it is impossible to precisely and consistently land at the desired spot on the runway. Additionally, landing at the desired spot is especially important in safety critical situations where the runway is short or there are equipment malfunctions.

### 2.7.5 Certification Cost

Traditional HUD systems such as those certified for commercial and business jets are more expensive than the cost of a typical GA aircraft. The EFRC HUD is complementary to existing GA aircraft cockpits and is intended to only provide advisory information. In addition, the EFRC HUD is designed as a wearable device, it is not hard-mounted, it is self-powered, and it receives data over Wi-Fi. Therefore, the EFRC HUD does not need to be certified. One possible source of flight data is Avidyne's IFD 550 (certified in March, 2017) with cockpit Wi-Fi capability (Avidyne Press Release, 2017). Other manufacturers are also being considered.

## 2.8    HUD Modes

Most modern HUDs come in different modes, with varying symbology based on the phase of the flight. The EFRC HUD is developed to affect three important phases of the flight, namely, post-takeoff, cruise and landing.

### 2.8.1    Post-Takeoff Mode

The post-takeoff mode of the EFRC HUD is based on the F-16 HUD design, as shown in Figure 2.10. The different parameters shown on the HUD can be obtained from a certified COTS unit such as Avidyne's IFD 550. In post-takeoff mode, information from traditional cockpit instruments such as airspeed, altitude, and attitude is included. In addition, FPM and AOA are shown.



**Figure 2.10 EFRC HUD Post-Takeoff Mode.**

**2.8.2 Cruise Mode**

In cruise mode, it is assumed that the pilot is primarily focused on achieving one of three conditions: maximum range, maximum endurance, or Carson cruise, using AOA and FPA. Therefore, the airspeed is removed along with the pitch-ladder creating a clutter-free design. Note that the pitch attitude can be mentally calculated as the difference between AOA and FPA. The cruise mode is illustrated in Figure 2.11.
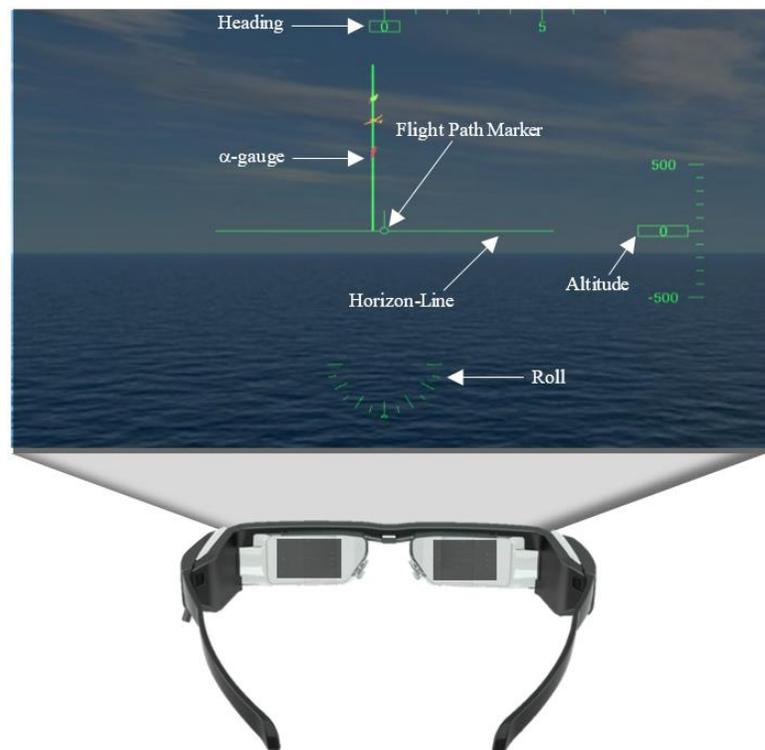


**Figure 2.11 EFRC HUD Cruise Mode.**

**2.8.3 Landing Mode**

The major difference between the cruise and landing modes is the addition of airspeed and the modified α-gauge. In the landing mode, the pilot aims to land the aircraft

at a desired spot on the runway, without stalling the aircraft. Therefore, the α-gauge is modified with caution and stall regions, as shown in Figure 2.12.
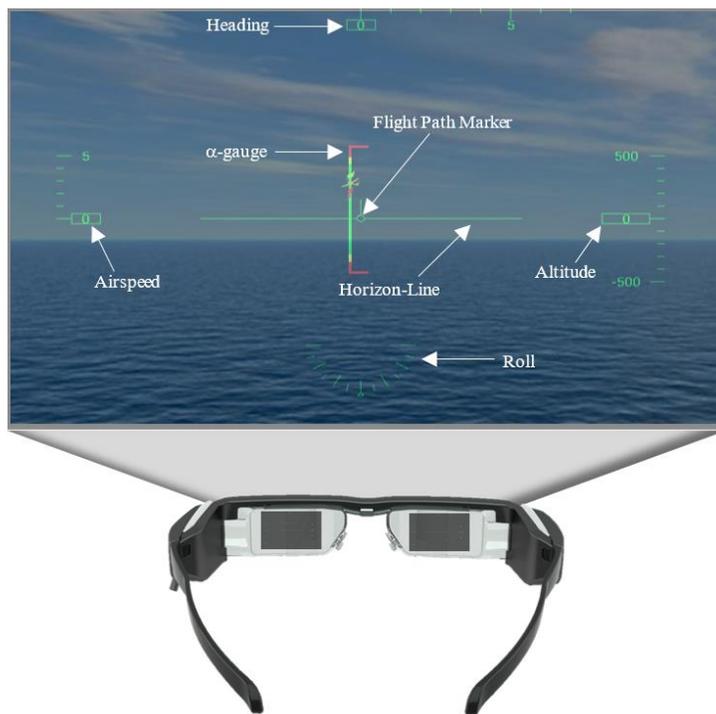


**Figure 2.12 EFRC HUD Landing Mode.**

## 3. METHODOLOGY

### 3.1 Experimental Hardware

In this section, the equipment used in this project will be discussed briefly. To help better understand the entire setup, the overall schematic is shown below. In Figure 3.1, the arrows represent the direction of data flow; in the case of a double-sided arrow, the data flow occurs in both directions.
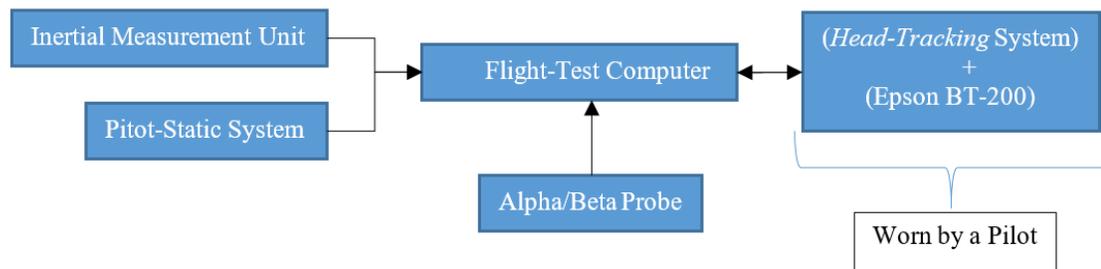


**Figure 3.1 Overall schematic of the EFRC HUD.**

The IMU helps obtain the attitude of the aircraft (roll, pitch, and heading) and flight-path angle, while the pitot-static system helps estimate the indicated airspeed and pressure altitude. The pitot-static system may not necessarily represent the aircraft's primary system, as any interference to the primary system will lead to increased certification costs. As an alternative, a COTS unit such as Avidyne's IFD550 can be used, which is a full-featured flight management system or navigation system (Avidyne Press, 2017) can provide all the information required by the EFRC HUD. Similar products are available with other manufacturers such as Garmin and Honeywell.

The Alpha/Beta probe represents an angle-of-attack/angle-of-sideslip system which is required for the complementary filter. In the previous section, a reference is mentioned for building a simple, effective and reliable angle-of-attack system, which can be used in its place.

The flight-test computer hosts all the filtering and data processing software. In the final product, the software can be installed directly on the data display device such as the Epson-BT 200, which comes with an Android 4.0 system and is capable of performing all the processing done by the flight-test computer. The only additional piece of equipment required would be a USB hub to increase the number of communication ports.

The Epson BT-200 is the data display device worn by the pilot. The head-tracking system may be a device attached either to the pilot's head or in another form as suggested in this thesis.

### 3.1.1 Experimental Aircraft

The experimental aircraft chosen for this research is a Cessna 182Q, equipped with an air data probe, Inertial Measurement Unit (IMU), Global Position System (GPS), Precision Differential and Static Pressure Transducers (PPTs), and a Data Acquisition System (DAS).

The air data probe consists of AOA and AOS mechanical vanes and a swivel head pitot-static system. The probe extends approximately one mean aerodynamic chord length ($\approx 60$") in front of the wing leading edge, as shown in Figure 3.2. The AOA and AOS vanes are attached to separate potentiometers, Control Position Transducers (CPTs), which send AOA and AOS measurements as changes in voltage.

**Figure 3.2 Air Data Probe Mounted on the Experimental Aircraft (Cessna 182Q).**

The IMU installed on the aircraft is a NovAtel UIMU-HG1700, shown in Figure 3.3. The IMU records aircraft attitude angles, rates, accelerations and velocities relative to the ground. For position information, the IMU communicates with a ProPak-V3 GPS, manufactured by NovAtel.

| Category | Description |
|---|---|
| Model | NovAtel UIMU-HG1700 |
| Component | HG1700 IMU |
| Sensors | Tactical Grade Gyro, Servo Accel |
| Solution Type | Blended GNSS+INS position, velocity and attitude |
| Output Data Rate | 100 Hz |



**Figure 3.3 NovAtel UIMU-HG1700.**

The Differential Pressure Transducer is used to measure total pressure, and the Static Pressure Transducer is used to measure static pressure. These precision pressure transducers (PPTs), manufactured by Honeywell, transform changes in pressure measured on the air data boom into digital signals.

The aircraft DAS collects data from the IMU, GPS, PPTs, and CPTs. All sensor signals are interpreted by a National Instruments (NI) Compact Reconfigurable Input Output (cRIO) communications module in real time at 50 Hertz. The cRIO simultaneously logs data in a raw format and outputs the data to the Flight Test Computer (FTC). The reference frame of the DAS is chosen so that it aligns with the body-frame of the experimental aircraft, i.e. x-axis is positive out the nose, y-axis is positive along the right wing and z-axis is positive down.

Several in-flight calibrations are required for the air data probe. The probe pitot-static system is calibrated using a GPS 4-leg maneuver. The probe AOA vane is calibrated using steady trim shots during the GPS 4-leg maneuver. Details of inflight calibrations, theory, practical considerations, and results are included in Ref. (Rogers, Martos, & Rodrigues, 2015).

In this research, the DAS is meant to represent a generic sensor processing system that may be replaced by any COTS system capable of providing accurate information of AOA, FPA, ground-speed, altitude, and attitude.

It must be noted that typical COTS $\alpha$ indicators are merely stall indicators and do not provide accurate information throughout the flight envelope. Therefore, the simple, effective COTS AOA design suggested in Ref. (Rogers, Martos, & Rodrigues, 2015) may be used to obtain an accurate signal throughout the entire flight envelope.

### 3.1.2 Display Device

The Display Device is the Moverio BT-200 made by Epson. It has the specifications shown in Table 3.1. The glasses can be worn even if the pilot has corrective lenses. The glasses are also shipped with optional sunlight shades.

The Epson BT-200 is meant to represent a pair of generic wearable AR glasses and can be replaced with a specific manufacturers glasses, based on the needs and demands of the final HUD design. Specification of the Epson BT-200 glasses are listed in Table 3.1.

**Table 3.1 Epson BT-200 Specifications.**

| Category | Description |
| --- | --- |
| Model | MOVERIO BT-200 |
| Resolution | 960x540 (QHD）/ 16：9 |
| Virtual Screen Size | 80" (at virtual distance of 5m) |
| Field of View | 23º |
| Color | 24bit full color (16770K color) |
| Sensors | Gyro / Accelerometer / Compass |
| System Software | Android 4.0.4 / Linux 3.0.21 |

The device reference frame is chosen so that it aligns with that chosen on the aircraft, i.e. x-axis is positive along the line-of-sight, y-axis is positive to the right and z-axis is positive down, as shown in Figure 3.4.

**Figure 3.4 Epson Moverio BT-200 and Assumed Reference Frame on the Experimental Aircraft.**

### 3.1.3 Flight Test Computer (FTC)

The FTC is a Windows 10 Notebook made by Asus, with 8 GB SDRAM, 1 TB hard drive and 56 Wh battery capacity. It serves as the master control for the DAS and the HUD, as well as a data processing and logging device. It saves the processed data into a Comma Separated Value (CSV) file that is then reduced and interpreted post-flight. In a real-world application, the FTC can be completely avoided, by deploying the HUD software on the AR glasses and modifying the software to read the DAS data via Wi-Fi or Bluetooth.

### 3.1.4 Head-Tracking System

For the purpose of tracking the head-motion of the pilot, different COTS units are tested. The 6 degree-of-freedom Bosch BNO055 IMU is shown in Figure 3.5, which is tested under the conditions described in Section 'Experimental Setup'. The unit comes with a built-in extended Kalman filter that computes the attitude solution at 100 Hz. The Bosch BNO055 is fit on an Arduino Uno board so that the sensory information can be obtained using the serial connection from the Arduino Uno board.

**Figure 3.5 COTS Bosch Unit Connected to Arduino for Head-Tracking Purposes.**

In addition, a PixHawk Mini, shown in Figure 3.6, is also tested for the head-tracking purposes. The PixHawk Mini also comes with a 24-state open-source Kalman filter, which computes the attitude solution at approximately 30 Hz.



**Figure 3.6. PixHawk Mini for Head-Tracking Purposes.**

In addition, the Epson BT-200's built-in accelerometer sensor is used to test the *accelerometer method* mentioned in the 'Theory and Background' section. All the systems are tested against a NovAtel IMU, which comes with a laser ring gyroscope. Assuming the NovAtel unit is the truth source, the three other algorithms are tested against it.

## 3.2    Experimental Software

The software is an integral portion of the project; therefore, it is necessary to elaborate upon the different features of the software for a better understanding of the suggested product.

### 3.2.1    Software Overview

The entire software is written in Java, which is the default language on the Android Framework. The Epson BT-200 device itself comes with an Android 4.0.4 Operating System. Using the Android Studio application, an application is designed and deployed on the BT-200. However, the device is always connected to the laptop even during flight-tests as the laptop acts as a central controlling unit. The overall architecture of the software is shown below in Figure 3.7.
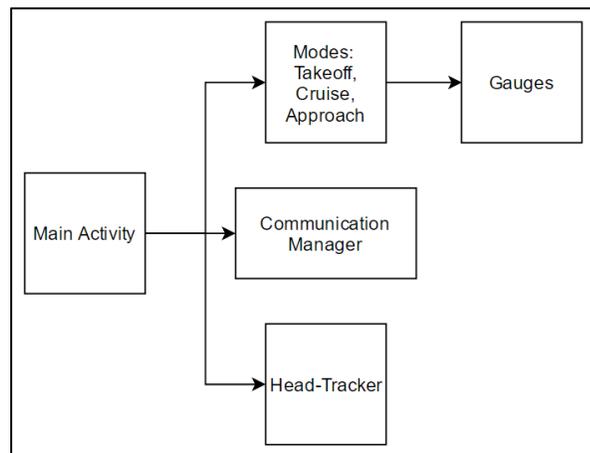


**Figure 3.7. Overall Architecture of the EFRC HUD Software.**

The *Main Activity* is where the entire application is started – all the different parts of the application are controlled form here. The different modes (Post-Takeoff, Cruise, and Landing) are mutually exclusive modules, i.e. only one can be selected at a time. The different modes correctly select the right type of gauges from the gauges "bucket" and draw, them at a specified frame rate the instant dictated by the *Main Activity*.

A 'Gauge' in this application is a public interface that encapsulates all the common properties of the different gauges such as the airspeed indicator, the pitch ladder, the altitude indicator, roll indicator etc. All the different gauges implement this interface and add the appropriate properties and modifications. Such a data abstraction helps ease any future modifications.

The Communication Manager is the main component responsible for communicating with the outside world and saving data for processing and use on the glasses.

The Head-Tracker module implements the 'Accelerometer' methodology discussed previously to estimate the relative pitch attitude of the glasses with respect to the aircraft body axis. It does so by accessing the built-in accelerometer on the Epson BT-200 glasses and comparing it with the acceleration from the Bosch BNO055 unit. The accelerations from the Bosch unit are read via a TCP channel that the head-tracker module is responsible for.

The data in this application only comes from the Data Acquisition System in the format shown below, where 'START_DATA' is the header that ensures the floating point data is intact, 'airspeed' is airspeed in knots, 'altitude' is altitude in feet, 'roll' is roll in degrees, 'heading' is heading in degrees, 'flightpath' is the FPA in degrees, 'aoa' is the AOA in degrees, '\n' is the end-of-line character.

```
START_DATA:airspeed;altitude;roll;pitch;heading;flightpath;aoa;\n
```

In order to ensure integrity of the data, a simple check is performed at the beginning of every read operation. If the incoming data has the proper 'header'; then, each of the subsequent floating point number is split and stored in a specific variable to be later used on the appropriate gauge. For example, consider the following incoming message shown below.

```
START_DATA:95.7;800.84;0.15;2.56;090;0.15;2.41;\n
```

In such a case, the data would be considered valid and processed accordingly. An airspeed of 95.7 knots is read and displayed on the airspeed gauge, the altitude of 800.84 feet is read and displayed on the altitude gauge and so on.

### 3.2.2  Gauges Description

A different number of vector notations is used to draw the different gauges in the software, which are briefly discussed below. In each of the gauges below, the direction for any parameter can be defined using any two 2D points in space, for example, (0, 1) and (0, 5) will specify a direction along the positive y-direction.

*Linear Gauges*

A linear gauge represents a simple tape with graduations that can be used for showing Airspeed, Altitude or Heading. The linear gauges can be specified to be directed 'Left', 'Right' or 'Down', where the direction indicates the place the text is placed. The Airspeed Indicator for example, would use the 'Right' linear gauge, the Altitude Indicator on the other hand, would use the 'Left' linear gauge and the Heading Indicator the 'Down' linear gauge. A sample linear gauge is shown below.
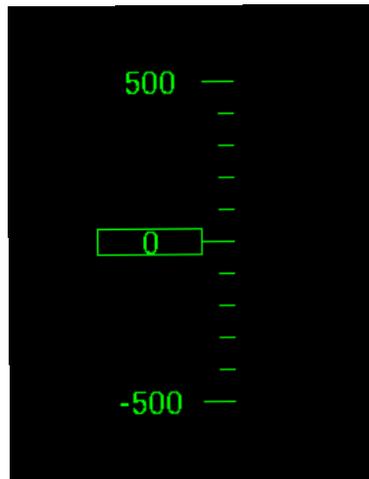


**Figure 3.8 Altitude Indicator Implementing the Abstract Linear Gauge.**

In order to draw the entire gauge, the current value is read first. If the current value is a multiple of the large graduation value, a large graduation mark is drawn or if the current value is a multiple of the small graduation value, a small graduation mark is drawn, else, the nearest "valid" value is estimated in either direction from the center. A valid value is one that is either a multiple of the small graduation value or the large graduation value. Once the "valid" value is estimated, each graduation mark is drawn at appropriate increments until the maximum height of the gauge is reached in either direction.

For example, let's assume that the small graduation mark on the Altitude Indicator is 100 feet and the large graduation mark 500 feet. If the current altitude is 0 feet, then a large graduation mark is drawn at the center, as it is a multiple of the large graduation mark value. Next, going in the up direction, each small graduation mark is drawn at increments of 100 feet. As soon as a multiple of 500 feet is encountered, a large graduation mark is drawn. The process is repeated in the other direction until the maximum height of the gauge is reached.

*Symmetric Gauges*

A symmetric gauge is one that has two linear gauges with a mirror in the center so that the two are a mirror image of the other. The pitch ladder is constructed using the symmetric gauge abstract class. Since the pitch indicator is the only gauge that uses this kind of gauge, the direction of the text and the gauge is fixed to a hard-coded value. The direction of the gauge is vertical and the text is both right and left. A sample symmetric gauge is shown below.
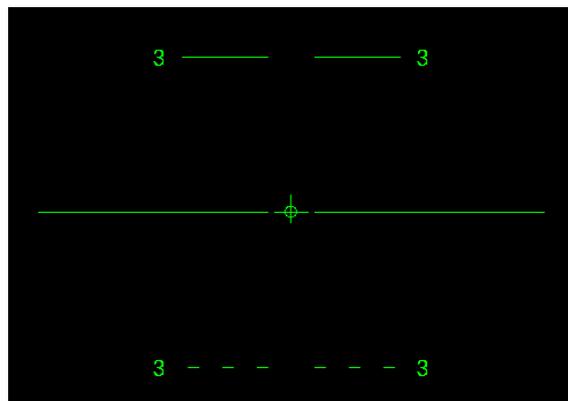


**Figure 3.9. The Pitch Indicator has implemented the Abstract Symmetric Gauge Class.**

The symmetric gauge follows the same algorithm described under linear gauge for drawing the different graduation lines on the gauge. The horizon line however, cannot rotate with the entire gauge, as the horizon does not rotate with the aircraft, when the head is held fixed in space.

*Curvilinear Gauges*

A curvilinear gauge is one that has a non-linear shape, typically circular to convey the information to the user. The Roll Indicator implements the abstract curvilinear gauge class. The current value on the gauge is displayed on the curvilinear using the same algorithm described under the linear gauge; the difference being that the direction of the graduation in this case is described using sine and cosine, i.e. the points are described in polar coordinates instead of rectangular coordinates, as done previously. The current value however, unlike the linear or symmetric gauges is not necessarily displayed at the center of the gauge. An isosceles triangle is used to represent the current value, which is constructed using the height and base length specified in the code. A sample curvilinear gauge is shown below.
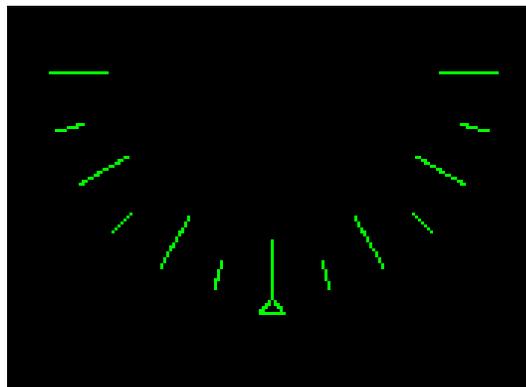


**Figure 3.10 The Roll Indicator has implemented the Abstract Curvilinear Gauge Class.**

*Angle of Attack*

The AOA is implemented in an innovative fashion, different from any of the gauges described above. For this reason, it is implemented as a separate abstract class. As mentioned previously, the EFRC HUD software is designed to accommodate different modes in flight, namely takeoff, cruise and approach. Each of the different modes can display the AOA in different fashion. The takeoff and approach modes display the AOA as a vertical bar with different symbols representing different pieces of information to the pilot. A sample AOA gauge is shown below.
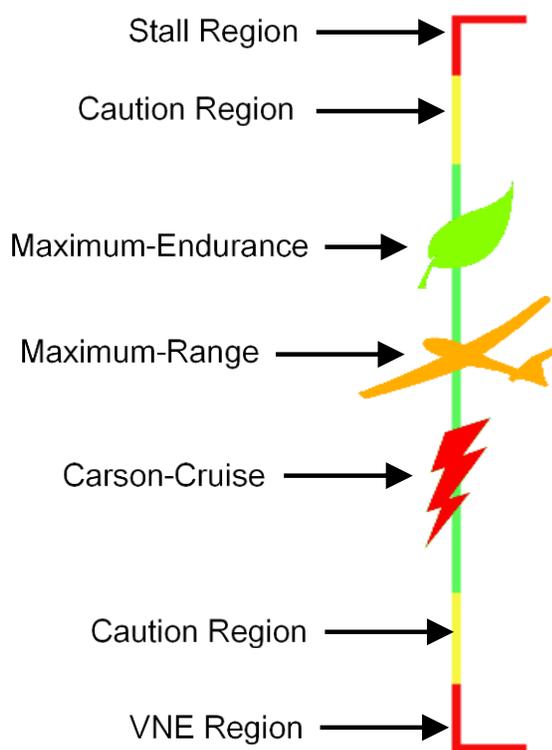


**Figure 3.11 AOA in Post-Takeoff and Landing Modes of the EFRC HUD.**

The current value of the AOA is represented at the center of the gauge. In order to draw the gauge, the same algorithm as that for a linear gauge is used.

### 3.2.3  Key Software Features

The HUD software is programmed on the Android platform using Object-Oriented Programming principles – some of the software characteristics are listed below.

*Abstraction*

Abstraction in the software lies in the design of the different gauges. To ensure a flexible design, a common interface is created, from which the different gauges are derived. This leaves room for future expansion.

*Multi-Threaded*

The HUD software requires multiple tasks to occur simultaneously; therefore, to avoid any bottlenecks, the software is designed in a multi-threaded fashion. For example, the *Communication Manager* is run in a different thread than *Head Motion Tracking*.

*Thread-Safe*

Multi-threaded design requires thread-safety measures to ensure the software does not misbehave due to thread calls to an inaccessible object. The safety measure incorporated into the design of the EFRC HUD is the *Singleton* design pattern, which ensures only one instance of the object (such as the *Communication Manager*) exists in the entire application.

*Low-Coupling*

The HUD software is made robust and flexible through low-coupling, so that changes in one class do not have a cascading effect on other classes.

*High-Cohesion*

The software is designed to have a high level of cohesion, i.e. each class is made responsible for a single logic, so that any logical changes can be implemented with ease.

## 3.3    Experimental Setup

In this section, the ground and flight tests performed to validate the effectiveness of head-motion tracking module, EFRC HUD, and complementary filter are discussed.

### 3.3.1    Pitch Ladder Calibration

One of the advantages of a HUD is the presence of the flight-path angle which (if presented accurately) tells the pilot where the aircraft is going. In order to position the FPA correctly on the HUD with the outside world, the pitch ladder must be calibrated. Once calibrated, the zero-pitch line will correspond to the horizon, as shown in Figure 3.12, when the pilot's head is level and the pilot's eyes are directly looking forward at the horizon.
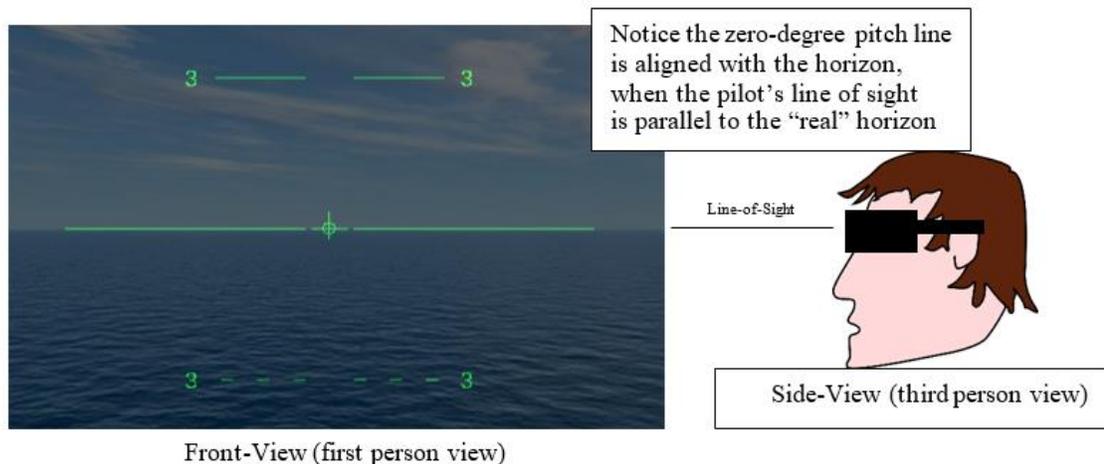


**Figure 3.12 Properly Calibrated Pitch Ladder (Illustration is Shown when the Pilot's Line-of-Sight is Parallel to the "Real" Horizon).**

The pitch ladder if not properly calibrated, i.e. the zero pitch line is not positioned at the proper location when the pilot's line-of-sight is parallel to the "real" horizon, will result in an apparent offset between the "real" horizon and the zero-pitch line, as shown in Figure 3.13. Note that the apparent offset in Figure 3.13 is positive as the horizon line on the glasses is above the "real" horizon.
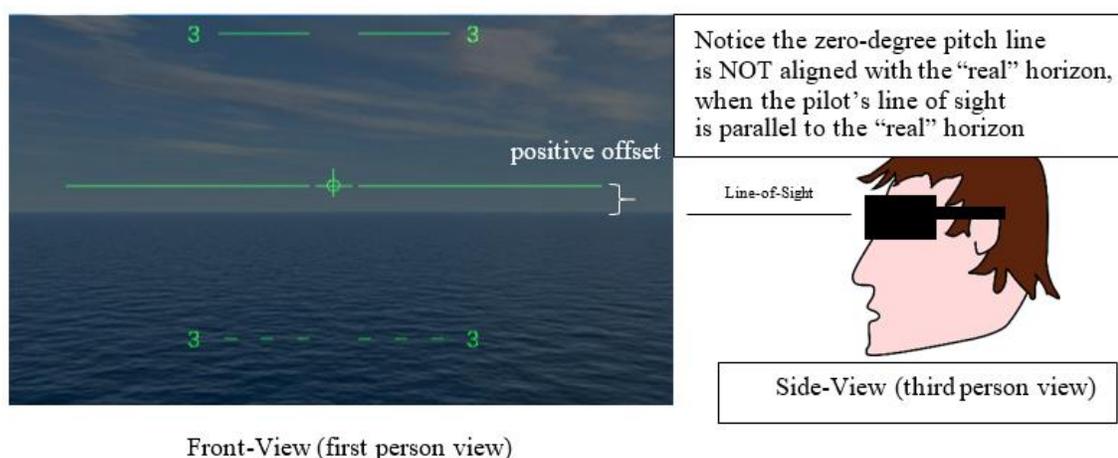


**Figure 3.13 Uncalibrated (or Misaligned) Pitch Ladder (Illustration is Shown when the Pilot's Line-of-Sight is Parallel to the "Real" Horizon).**

The Android framework provides a module to estimate the attitude of the android device. The solution of the methodology works very well in a quasi-static environment and the same methodology will be used to calibrate the pitch ladder for the EFRC HUD before taxi. In order to properly calibrate the pitch ladder, the following steps were followed.

i)      The Android attitude estimation is verified against the Bosch BNO055 solution. A setup for such a procedure is shown below.

**Figure 3.14 Rotating the Securely Fit Glasses and the Arduino Uno about the Different Axes to Verify the Reliability of the Android's Attitude Estimation Scheme (assuming that the Bosch BNO055 Solution is True).**

ii)   Using the verified Android attitude estimation, the pilot's "eyes level" position is identified.

iii)   The pilot's "eyes level" position is used to draw the zero pitch line along the line of sight of the pilot. The "real" horizon line is used to identify the "eyes level" position as shown in Figure 3.15.

**Figure 3.15 Looking at Different Angles from the "Real" Horizon to Calibrate the Pitch Ladder.**

iv)      The process is repeated until the zero pitch line is drawn at the same position three consecutive times. Repetition of the process is important to eliminate error due to discrepancies in the way the glasses are worn each time.

In order to verify the accuracy of the Android solution, the glasses and the Bosch BNO055 units are placed securely in a container so that neither unit can move. Once a snug-fit is achieved, the entire container is rotated about the appropriate axes to verify the reliability of the Android's attitude estimation scheme. The pitch attitude is of primary concern and therefore, only those results are shown later in the report.

### 3.3.2   Head-Tracking System Selection

To select the best head-tracking system based on the different systems and algorithms available, the following setup is organized. Each of the different instruments comes with a built-in attitude estimation scheme, which is compared to the truth source (NovAtel).
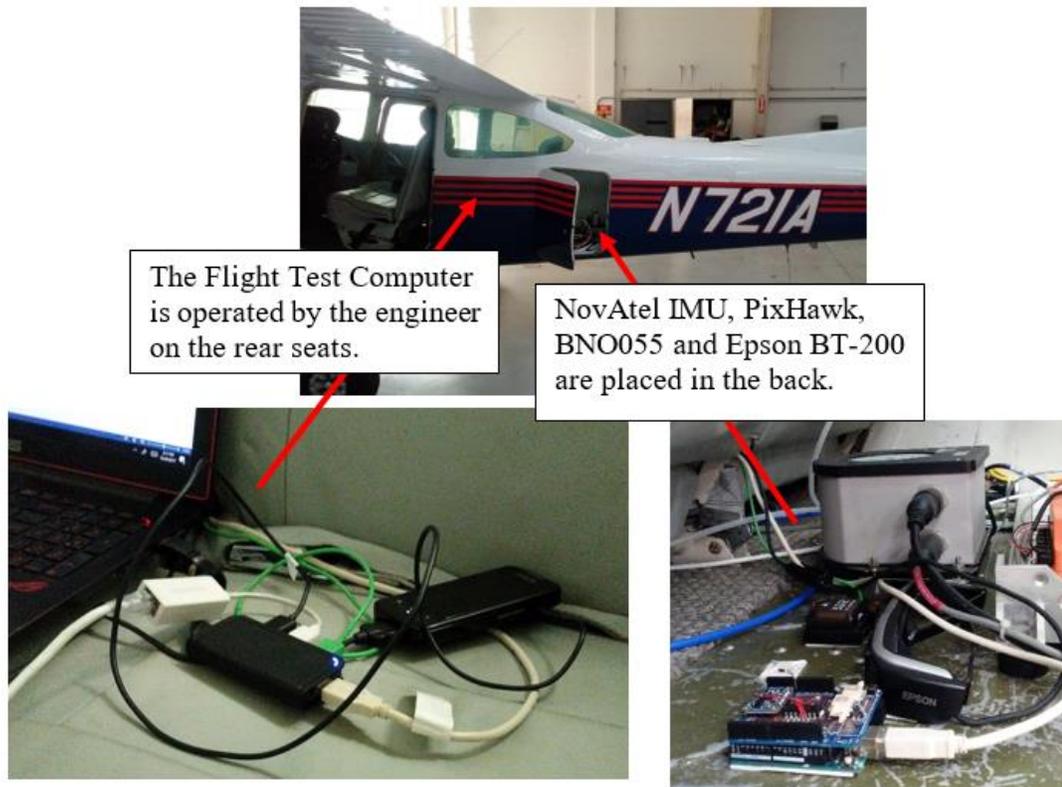
**Figure 3.16 Head-Tracking System Selection.**

Each of the instruments is calibrated per the corresponding manual. With the PixHawk module, the calibration is performed by placing the unit in different stable positions and letting the unit record the offsets for each sensor. Following which, the offset values are hard-coded into the system so that the unit stays calibrated through power cycles.

The Bosch BNO055 IMU is calibrated per the manual, which suggests moving the device in a figure 8 pattern for calibrating the magnetometer, placing the device in different stable positions for the accelerometer and leaving the device still for gyroscope calibration. Once the device is calibrated, the various sensor offsets are stored in the system's non-erasable memory (EEPROM) so that on the next power cycle, the offsets can be restored and the unit does not have to be recalibrated. The Bosch unit code for calibrating and reading angles is placed in the Appendix section.

With the NovAtel IMU, calibration occurs automatically as soon as the device is powered and the attitude solution is transmitted once the calibration process is complete. A similar procedure occurs on the Android OS, where the accelerometer is automatically calibrated as soon as the device is powered and the accelerometer values are transmitted once the calibration process is complete. The following table summarizes the test matrix of the different methodologies used for computing the attitude of the aircraft. Note that the accelerometer method does not compute the absolute attitude and is therefore not capable of estimating the attitude of the aircraft; it only computes attitude of a moving device with respect to the attitude of a device fixed to the aircraft. The results from the test are shown in the next section. The table below summarizes the different solutions used for head-tracking in the EFRC HUD.

**Table 3.2 Summary of the Different Attitude Estimation Methodologies Being Compared.**

| Attitude Estimation Method | Source | Relative/Absolute |
|---|---|---|
| Sensor Fusion (Ring-Laser Gyro, Accelerometer, Magnetometer) | NovAtel | Absolute |
| Sensor Fusion (MEMS Gyro, Accelerometer, Magnetometer) | PixHawk | Absolute |
| Sensor Fusion (MEMS Gyro, Accelerometer, Magnetometer) | BNO055 | Absolute |
| Accelerometer Data Compilation | BNO055 & BT-200 | Relative |

### 3.3.3   Overall Effectiveness

To test the overall effectiveness of the HUD, the following test matrix is proposed.

**Table 3.3 Flight Test Matrix**

| Task Description | Assistance | Acceptable Tolerance |
|---|---|---|
| Climb and Level-Off | *NO Assistance* | *+/- Δ50ft in final altitude* |
| Climb and Level-Off | *EFRC HUD (takeoff)* | *+/- Δ50ft in final altitude* |
| Fly Constant α | *EFC HUD (cruise)* | *+/- Δ½deg in α;  +/- Δ 50ft in altitude* |
| Touch Target on Runway | *No Assistance* | *+/- Δ50ft from selected target* |
| Touch Target on Runway | *EFRC HUD (landing)* | *+/- Δ50ft from selected target* |

A subject pilot with zero flight experience is chosen to perform these tasks. This will allow the effectiveness of flying with a HUD to be more easily identified. In each task, a professional pilot first stabilizes the aircraft and then transfers the flight controls to the inexperienced subject pilot.

i)      Climb and level-off

In this task, the professional pilot levels off at an arbitrary test altitude and transfers control to the inexperienced pilot, who first performs the task without any assistance and then performs the same task with the assistance of the EFRC HUD.

ii)     Fly constant α

In this task, the professional pilot levels off the aircraft at an arbitrary energy state at an arbitrary altitude. After stabilizing the aircraft, the same pilot begins adjusting the power setting and levels altitude to maintain the aircraft at a constant AOA using the cruise mode of the EFRC HUD.

iii)    Spot Landing

In order to land the aircraft, the professional pilot aligns the aircraft with the runway on a long final and 1000 feet above ground level and transfers the flight controls to the inexperienced subject pilot. The subject pilot then attempts to land the aircraft with the assistance of the EFRC HUD. In another attempt, the inexperienced pilot attempts to land the aircraft without any assistance. The professional pilot marks each task as successful or a failure based on the distance from selected target 50 feet above the runway. If the subject pilot is able to land within 100 feet of the selected target, the task is successful, else it is marked as a failure.

In order to test the overall effectiveness of the EFRC HUD, the head-tracking module is required. Based on the results, the accelerometer method seemed to be the most reliable method, which is used by placing the accelerometer (Bosch BNO055) on the fuel selector pane of the aircraft. The glasses are worn by the pilot and the accelerometer algorithm is used to estimate the attitude of the head with respect to the aircraft. In the figure below, the entire setup with the head-tracking module is shown for a test with the assistance of the EFRC HUD.

**Figure 3.17 Experimental Setup for the Head-Tracking Module to be Functional In-Flight.**

### 3.3.4 Complementary Filter Testing

To test the effectiveness of the complementary filter, the alpha-beta probe shown in Figure 3.2 is used. The alpha-beta probe is put through the FAA code to obtain the angle-of-attack and angle-of-sideslip, which are then post processed using a code developed based on the discrete form of the complementary filter shown in the 'Theory and Background' section.

# 4. RESULTS AND ANALYSIS

## 4.1 Calibration of the Pitch Ladder

As mentioned previously, Android has a sensor fusion algorithm in store, which estimates Euler angles of the Epson BT-200. In order to verify the reliability of the Android's sensor fusion algorithm, the device is placed in a secure snug-fit container and both devices are rotated about the different axes to verify the accuracy of the Android's solution. The results are plotted in Figure 4.1. From the results, it can be concluded that the BT-200 glasses are in a close agreement with the Bosch unit.
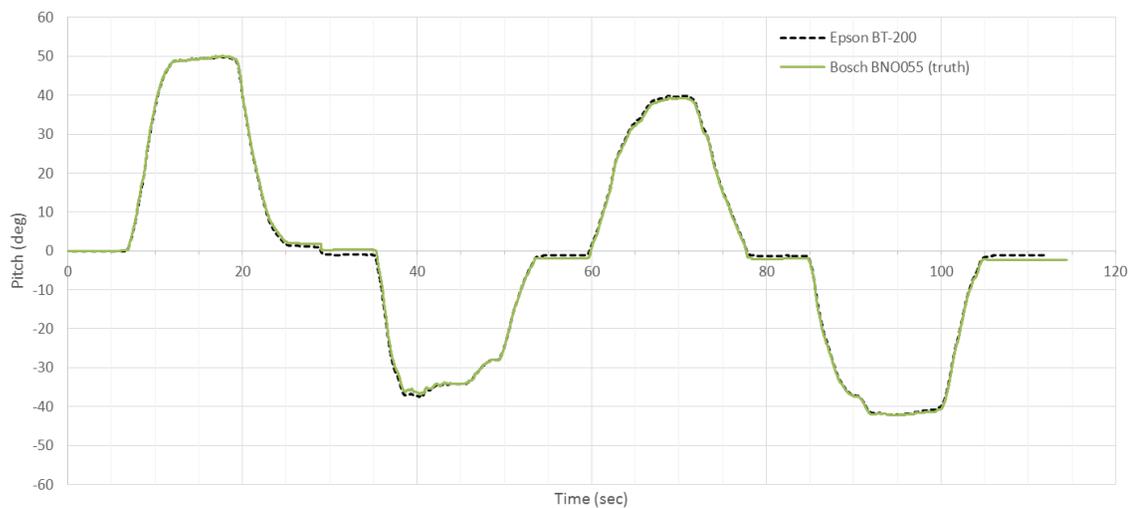


**Figure 4.1 Rotating the Epson BT-200 and BNO055 Simultaneously about the X-Axis to Get Pitch.**

Table 4.1 shows the location of the horizon line on the Epson BT-200's screen when the pilot's eyes are "leveled". As seen in the table, the initial location of the horizon line is arbitrarily chosen to be 50% of the screen height. However, upon inspection, it is learned that there is a negative offset; therefore the location of the horizon is reduced to 40%. Note that the glasses are taken off and re-worn at every attempt to ensure that the result is not affected by the way the glasses are worn. In addition, the same person that calibrated the glasses is selected to test the glasses in flight.

**Table 4.1 Iterative Calibration of the Epson BT-200's Pitch Ladder using the "Real" Horizon.**

| Attempts | Location (in percentage of the screen height) |
|----------|----------------------------------------------|
| Attempt 1 | 50% |
| Attempt 2 | 40% |
| Attempt 3 | 45% |
| Attempt 4 | 42% |
| Attempt 5 | 42.5% |
| Attempt 6 | 42.5% |
| Attempt 7 | 42.5% |

## 4.2    Head-Tracking System Selection

The results of the test of the COTS solutions such as the PixHawk Mini Autopilot and the Bosch BNO055 units are shown below. As seen in Figure 4.2 and Figure 4.3, neither the PixHawk nor the Bosch unit can be used to produce an attitude solution of required accuracy and reliability. The peak to peak error is around 15 degrees for both units. In order to use the solution reliably, an accuracy of at least 2 degrees is required, which could not be achieved from either of the commercially available solutions.
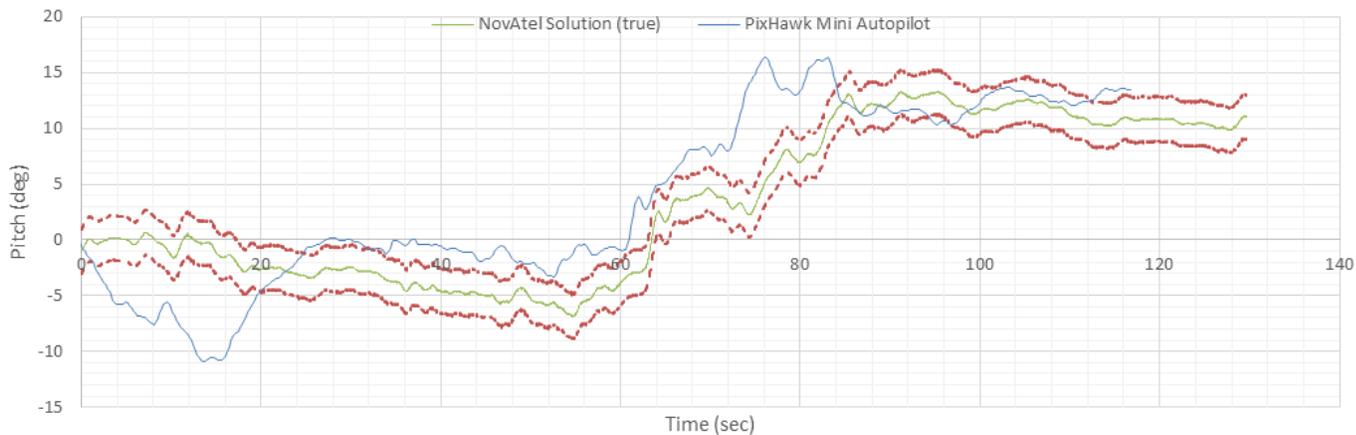
**Figure 4.2 PixHawk Mini Autopilot Solution Compared to the NovAtel Solution (with ± 2 Degree Tolerance Lines).**
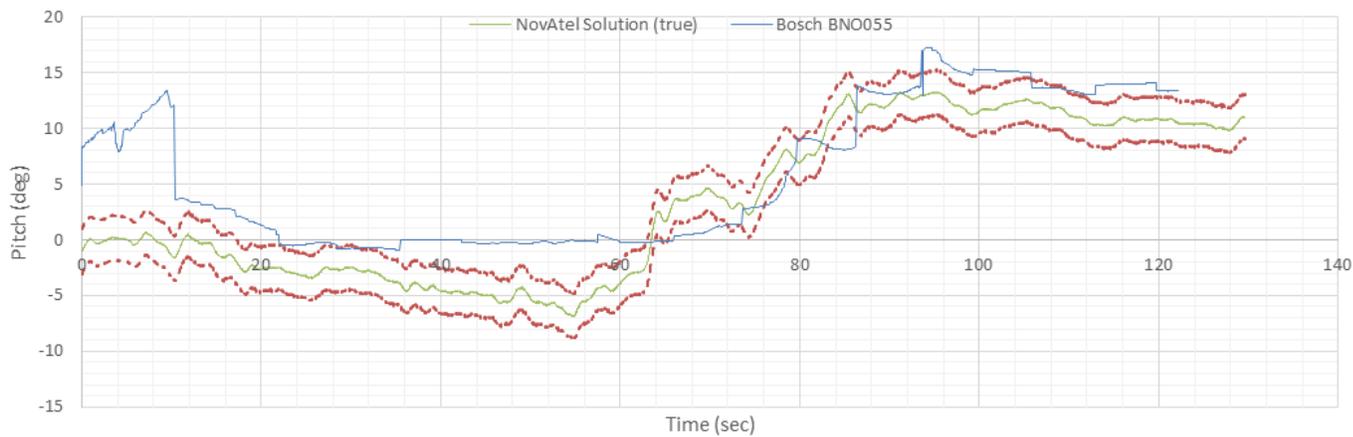


**Figure 4.3 Bosch BNO055 Solution Compared to the NovAtel Solution (with ± 2 Degree Tolerance Lines).**

The proposed solution of the Accelerometer method is much more accurate than either of the commercial solutions, as shown in Figure 4.4. However, it must be noted that the accelerometer method relies on the accuracy of the NovAtel solution for the final solution to be accurate. In other words, the accelerometer method only computes attitudes relative to the aircraft; therefore, if a self-contained methodology is required for computing the absolute orientation of the head, the accelerometer method is not viable. It is only applicable for aircraft that have the capability to supply information via Wifi or other means. For that matter, the entire EFRC HUD, suggested in this thesis, is only applicable for aircraft with such a capability.
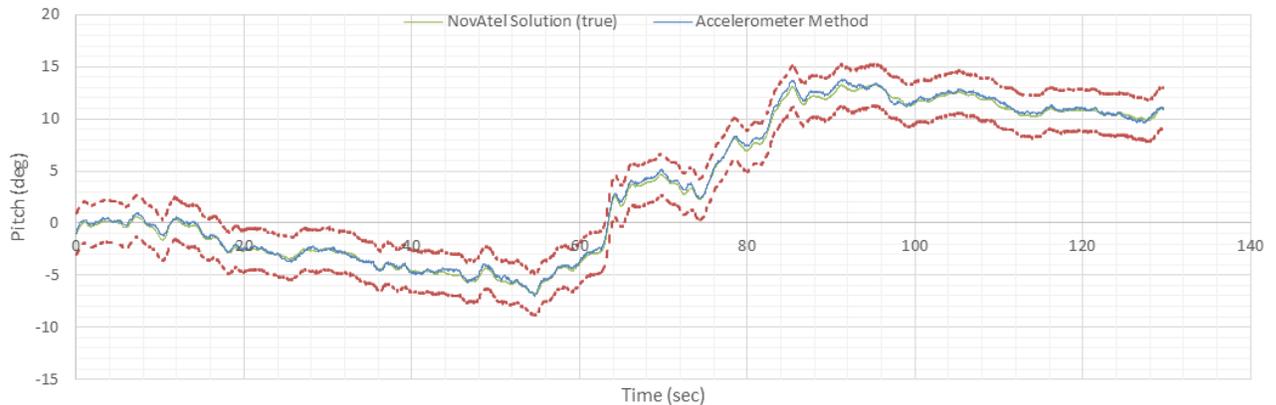


**Figure 4.4 Accelerometer Method Solution Compared to the NovAtel Solution (with ± 2 Degree Tolerance Lines).**

Flight path angle of the aircraft during the approach is shown in Figure 4.5.
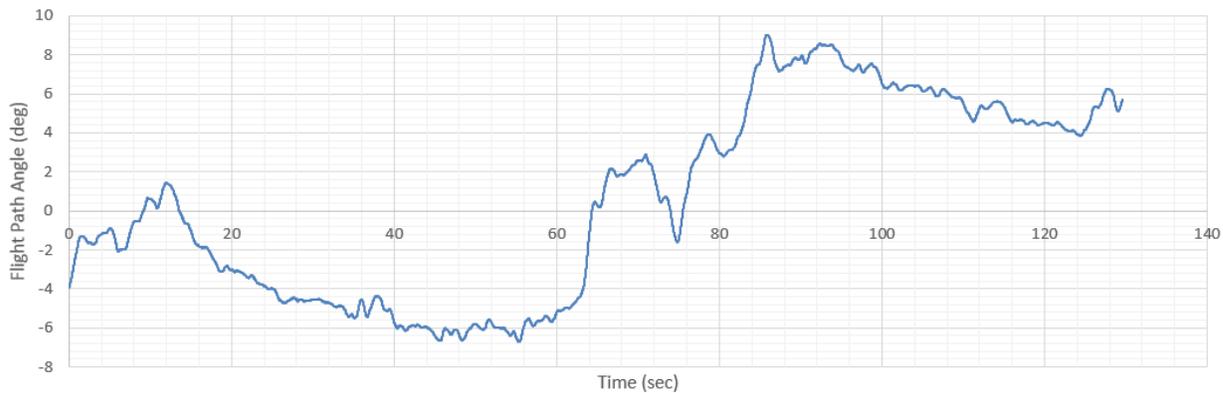
**Figure 4.5 Flight Path Angle of the Test-Aircraft during Approach from the NovAtel Unit.**

Flight path angle of the aircraft during the approach is shown in Figure 4.6.
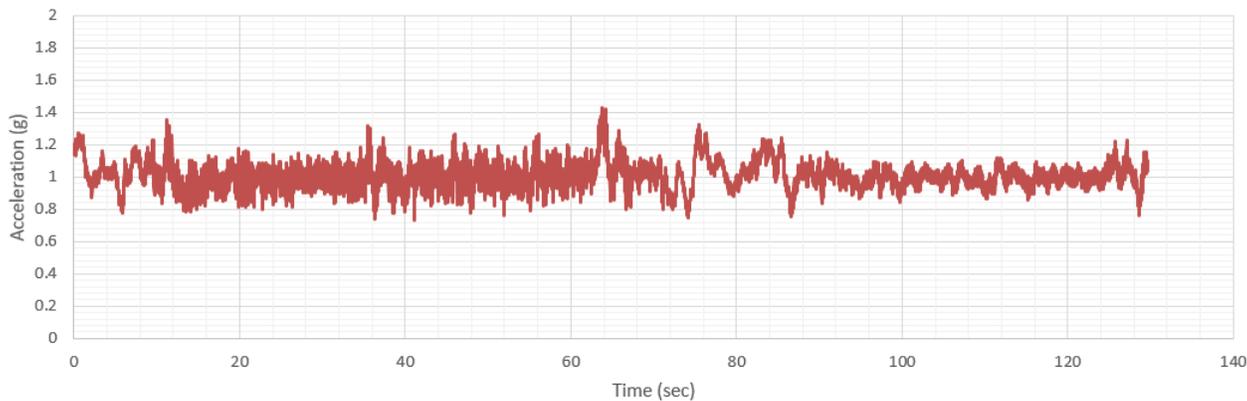


**Figure 4.6 Acceleration of the Test-Aircraft during the Approach from the NovAtel Unit.**

The angular rates of the aircraft are shown in Figure 4.7 through Figure 4.9.
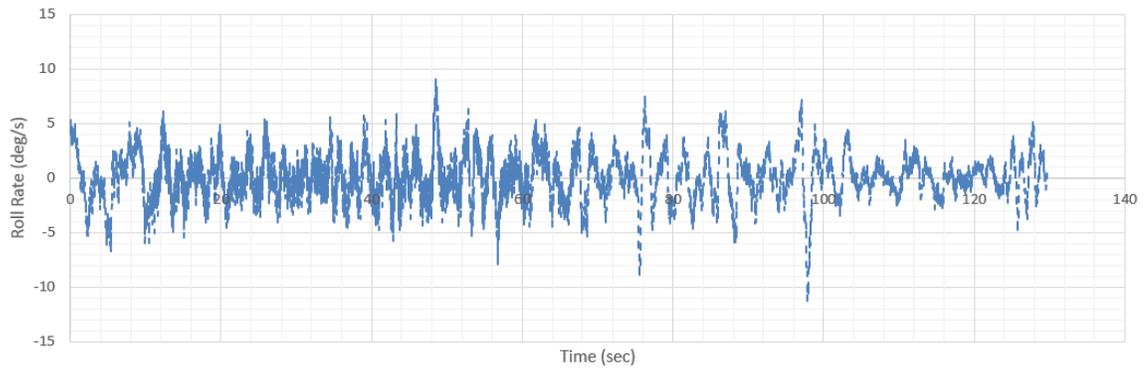
**Figure 4.7 Roll Rate of the Aircraft during the Test Maneuver from the NovAtel Unit.**
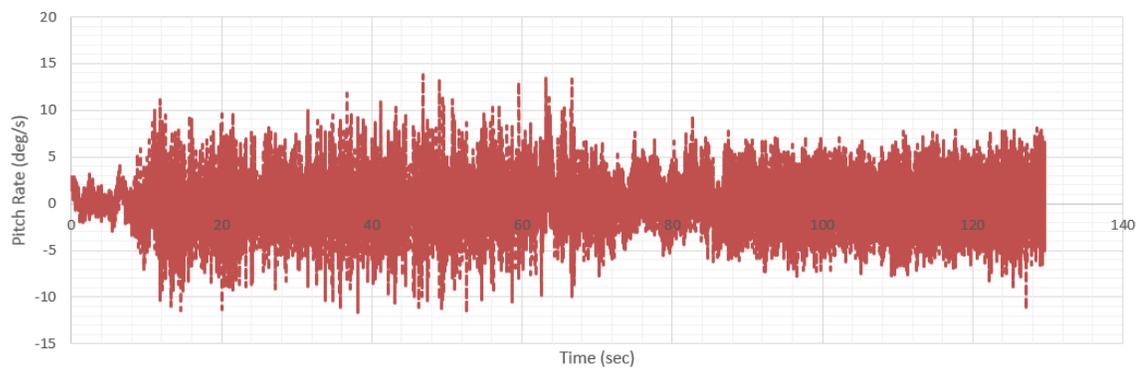


**Figure 4.8 Pitch Rate of the Aircraft during the Test Maneuver from the NovAtel Unit.**
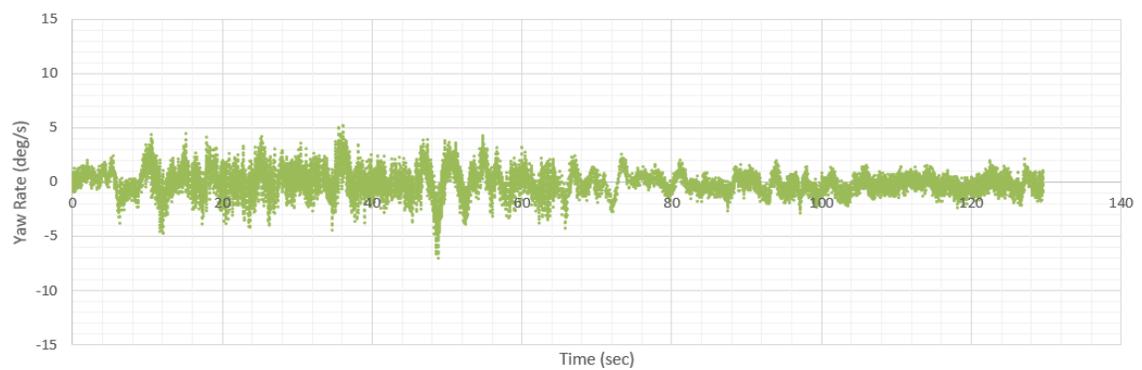


**Figure 4.9 Yaw Rate of the Aircraft during the Test Maneuver from the NovAtel Unit.**

In order to test the accuracy of the method in straight and level flight, a setup similar to that previously tested is used to obtain the results shown in Figure 4.10.
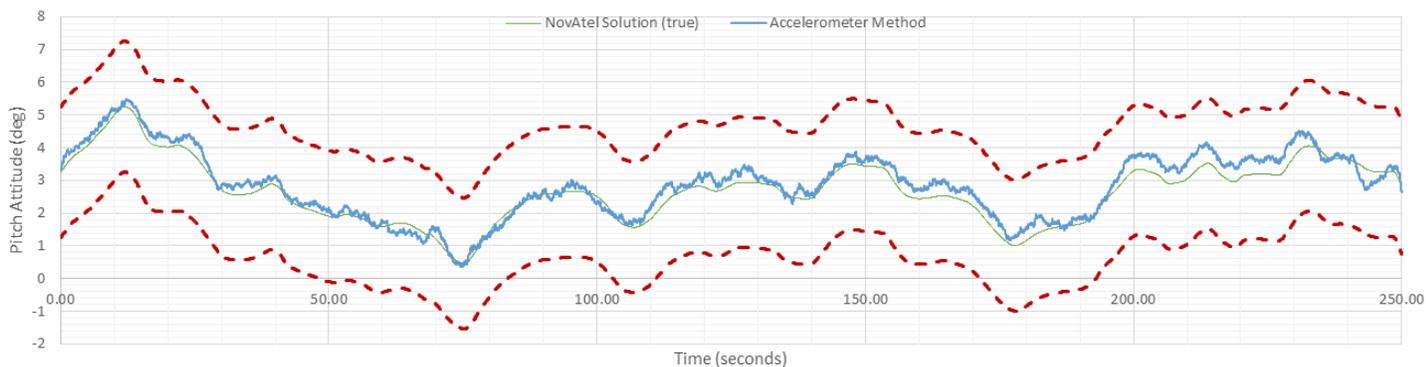


**Figure 4.10 Accelerometer Method Solution compared to the NovAtel Solution in Straight-and-Level Flight.**

## 4.3 Overall Effectiveness

The climb test results demonstrate the benefit of the EFRC HUD. While the inexperienced subject pilot could not achieve the desired tolerance when not wearing the EFRC HUD, the subject pilot was able to achieve the task within the specified tolerances with the EFRC HUD. This phase of flight suggests a clear advantage of using the EFRC HUD, as seen in Figure 4.11 and Figure 4.12.
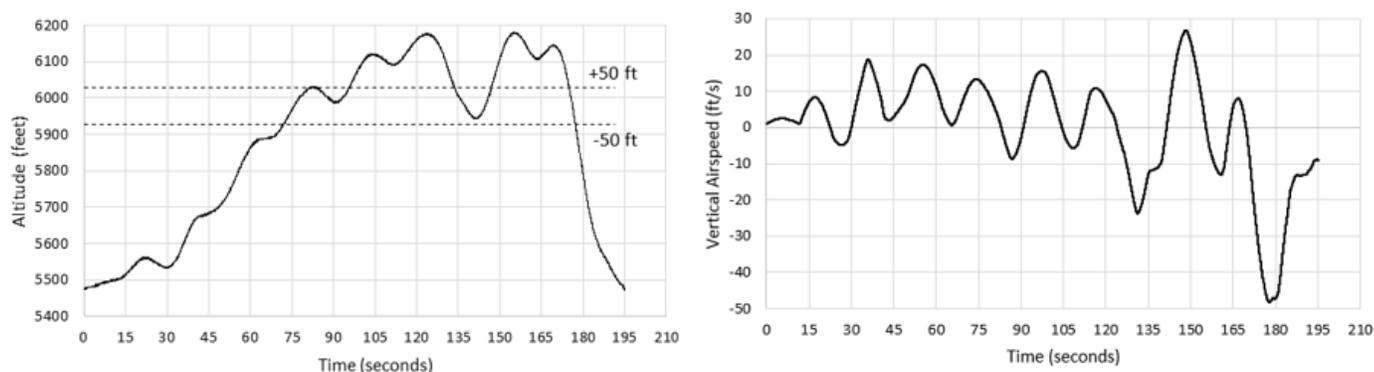


**Figure 4.11 Climbing from 5500 feet to 6000 feet Without Any Assistance.**
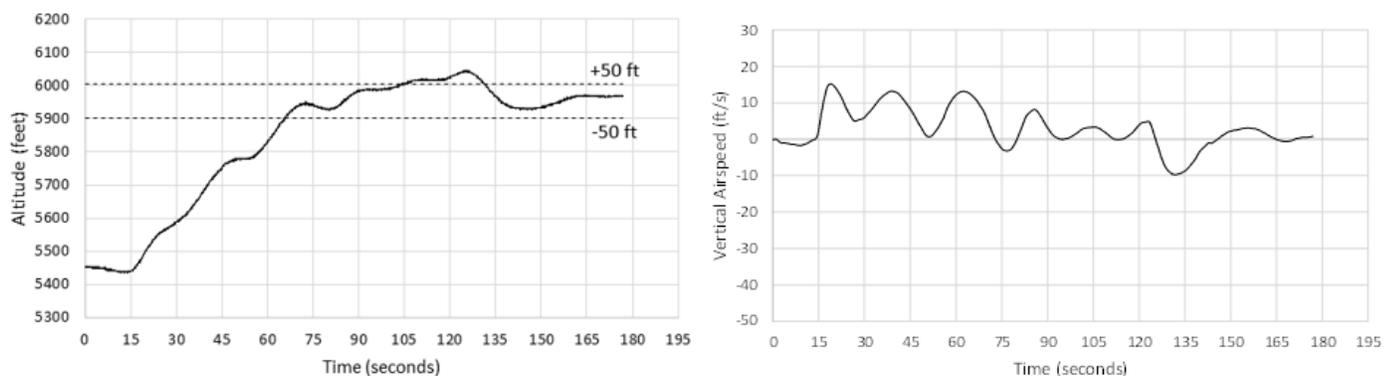
**Figure 4.12 Climbing from 5500 feet to 6000 feet Using the EFRC HUD.**

Figure 4.13 shows the inexperienced pilot performing a climb task using the assistance of the EFRC HUD.



**Figure 4.13 Inexperienced Pilot Performing Climb Task Using the Glasses.**

In testing the EFRC HUD in approach mode, the inexperienced pilot was able to show that the glasses are beneficial in providing the much needed confidence that the glide slope is correct and that the aircraft is indeed headed to the right target spot on the runway. Nonetheless, the atmospheric turbulence is a major disturbance that needs to be properly dealt with. With heavy turbulence, the aircraft becomes very bouncy and the glasses cannot be used effectively as the display becomes difficult to follow.

A sample of the approach where the inexperienced pilot uses the assistance from the EFRC HUD is shown Figure 4.14.



**Figure 4.14 Inexperienced Pilot Performing Approach Task with the Head-Tracking Module in Session.**

## 4.4 Complementary Filter Testing

Flight-test data are used to measure the effectiveness of the complementary filter in different scenarios, as shown in Figure 4.15 through Figure 4.18. Flight test results using a complementary filter demonstrate the effectiveness of the filter in different scenarios with light, moderate, and heavy atmospheric turbulence.

During the Short-Period mode excitation, shown in Figure 4.15, the atmospheric turbulence was light, as evident from the scatter in the vane output. The output from the complementary filter aligned well with the vane data.
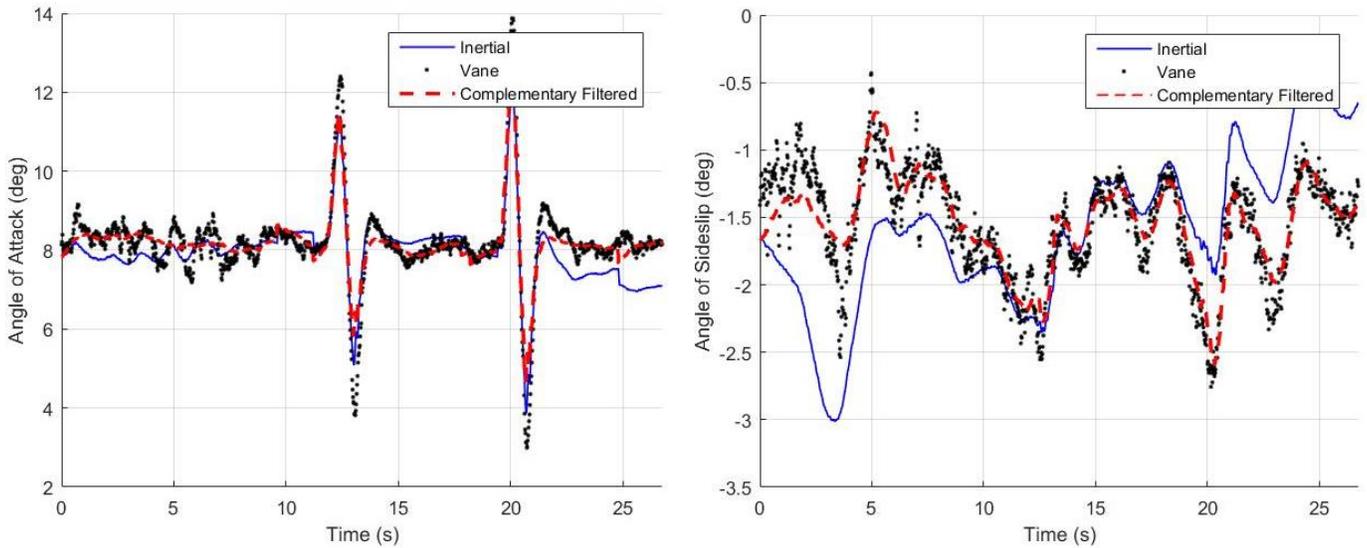
**Figure 4.15 Flight test data for Short Period excitation in Light Atmospheric Turbulence.**

During the Dutch Roll mode, shown in Figure 4.16, the complementary filter worked well in the absence of atmospheric turbulence. However, in such a condition, the complementary filter essentially weighed the vane data much higher than the inertial data to avoid drift. This is the same as if the vane data alone was used.
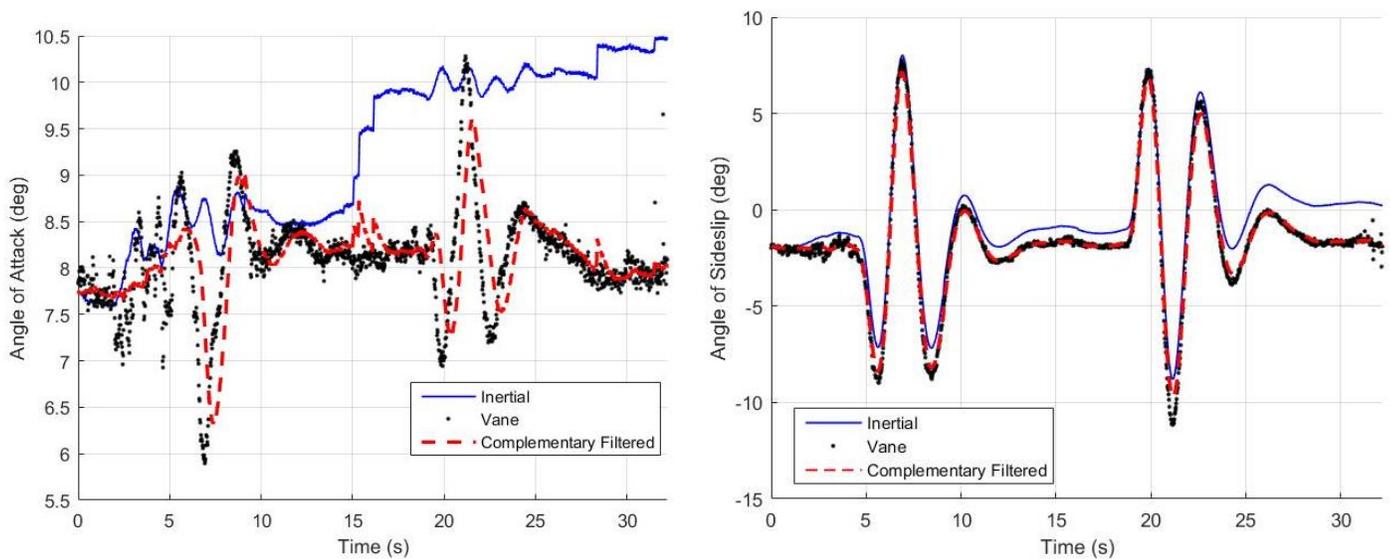
**Figure 4.16 Flight Test Data for Dutch Roll Excitation in a Calm Atmosphere.**

In straight-and-level flight, moderate atmospheric turbulence was present, as shown in Figure 4.17. The complementary filter in this case essentially averaged the vane data. If a "moving" average filter were used in this case, there is a potential for a phase lag in the output, which cannot be used in a real-time application with the EFRC HUD. The results show that the complementary filter works in a straight and level flight with moderate atmospheric turbulence.
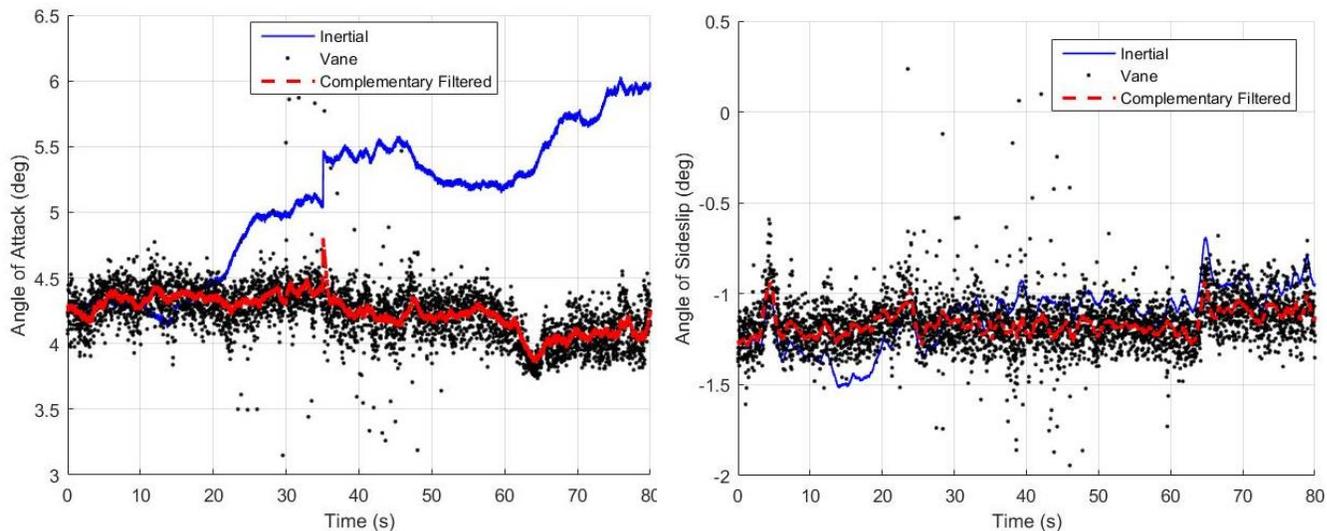
**Figure 4.17 Flight Test Data for Straight-and-Level Flight in Moderate Atmospheric Turbulence.**

Another straight-and-level test point is performed in high atmospheric turbulence, as shown in Figure 4.18. Again, the complementary filter tracked the vane data quite well and avoided the drift in the inertial data, proving that the complementary filter avoids the drift from the inertial data and the noise from the air data vanes.
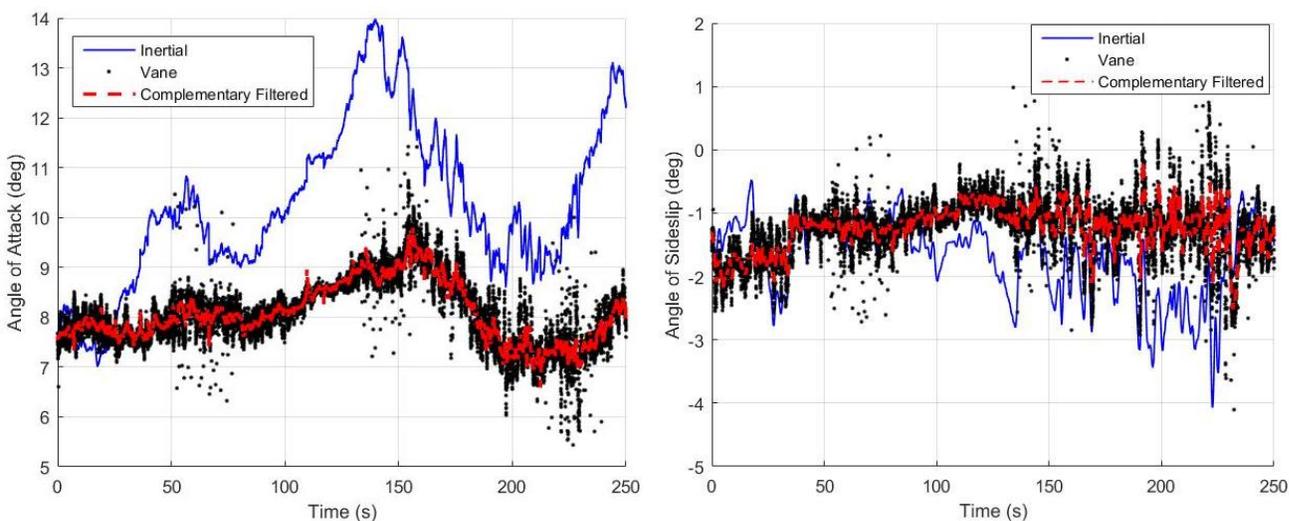


**Figure 4.18 Flight Test Data for Straight-and-Level Flight in High Atmospheric Turbulence.**

# 5. CONCLUSIONS

In summary, a low-cost wearable COTS AR HUD system is designed, successfully reduced to practice, and flight tested. A complementary filter on air data flow angles is proposed and successfully flight tested for straight and level flight, dynamic maneuvering, and atmospheric turbulence, provided that a reasonably accurate lift curve is determined. A novel accelerometer method is proposed for estimating the relative pitch attitude estimation of the pilot's head. The method is evaluated on the ground and in flight. The HUD system is tested, through various test points, to make flying more intuitive and efficient, thereby affecting the GA LOC.

Based on the flight-test results, the EFRC HUD provided a clear advantage in the climb. An inexperienced subject pilot was asked to trim the aircraft to straight and level flight after climbing to a different altitude. The difficulty of the task is proven through the first test point where the inexperienced pilot climbed to a new altitude and trimmed the aircraft without any assistance from the EFRC HUD. In the second task the difference in performing the task and obtaining the desired altitude tolerances was very apparent. In the first task, the pilot was unable to achieve the desired tolerances while in the second task with the EFRC HUD, the subject pilot was able to achieve the altitude tolerances with ease.

The problem of atmospheric turbulence was especially evident throughout flight tests. The results from the complementary filter demonstrate a methodology for dealing with the problem of atmospheric turbulence. In all cases tested, including dynamic mode excitation, the complementary filter calculated the air flow angles correctly. The performance of the filter can be further improved by tuning the filter coefficient in a more extensive flight test program.

While the entire test matrix could not be tested due to funding constraints, the work presented is compelling enough to provide a clear cost benefit to the OOP of the GA sector.

## 5.1 Future Work

Although the experiment demonstrated the benefit of HUD technology for a GA pilot, several improvements are suggested for future experiments.

- Further flight testing must be conducted to complete the suggested test matrix and fully identify the benefit of the HUD.

- An important phase of flight that would benefit from using the HUD is the takeoff phase. Further research must be conducted to evaluate the possibility of using AOA and AOS parameters to perform takeoff instead of relying on airspeed.

- The landing and cruise modes of the HUD must be combined using a logarithmic scale for the $\alpha$-gauge. This results in a finer AOA resolution at the center of the display versus at the ends. This change will allow the pilot to fly at a constant AOA in cruise and use the same instrument to ensure the aircraft does not stall in the landing phase.

# REFERENCES

Ahmed, H., & Tahir, M. (2016). *Accurate Attitude Estimation of a Moving Land Vehicle Using Low-Cost MEMS IMU Sensors.* IEEE.

Avidyne Press. (2017, March 13). *Avidyne Press Release.* Retrieved from Avidyne Website: http://www.avidyne.com/news/press.asp?release=342

Avidyne Press Release. (2017, March 13). *AEA 2017: 356.* Retrieved from Avidyne: http://wwww.avidyne.com/news/press.asp?release=342

Beard, R. W., & McLain, T. W. (2012). *Small Unmanned Aircraft : Theory and Practice.* Princeton University Press.

Bertorelli, P. (2013, August 06). *AirVenture Tech Bits.* Retrieved from AeroCross: http://www.aerocross.com/pdfs/News/AVWEB_20130806.pdf

Bertorelli, P. (2016). *What AoA Indicators Don't Do That They Should.* AVWeb.

Bimber, O. (2015, December 21). *Head tracking progress.* Retrieved from https://glass.aero/news/2015/12/head-tracking-progress/

Croft, J. (2013). Finding Focus. *Aviation Week & Space Technology*, 47-48.

*Definition of retrofit in English:.* (n.d.). Retrieved from Oxford Dictionary: https://en.oxforddictionaries.com/definition/retrofit

Dorr, A. D. (2016, March 09). *Press Release - FAA Proposes Rule to Overhaul Safety Certification Standards.* Retrieved from https://www.faa.gov/news/press_releases/news_story.cfm?newsId=20054

Ebit Systems - Aerospace. (2014). *SKYLENS.* Haifa: Eblit Systems.

Elbit Systems. (2016, October 27). *Elbit Systems SKYLENS™ Wearable HUD Begins Flying in Final Configuration Onboard ATR-72/42 Aircraft.* Retrieved from http://elbitsystems.com/pr-new/elbit-systems-skylens-wearable-hud-begins-flying-final-configuration-onboard-atr-7242-aircraft/

Epson. (2014). *Moverio BT-200 Smart Glasses (Developer Version Only).* Retrieved from https://epson.com/For-Work/Wearables/Smart-Glasses/Moverio-BT-200-Smart-Glasses-%28Developer-Version-Only%29/p/V11H560020

FAA. (n.d.). *14 CFR Part 91*. Retrieved from CFR: www.airweb.faa.gov

Federal Aviation Administration. (2017). *Private Pilot - Airplane Airman Certification Standards.* Washington, DC: US Department of Transportation.

Gee, S. W., Gaidsick, H. G., & Enevoldson, E. K. (1971). *Flight Evaluation of Angle of Attack as a Control Parameter in General-Aviation Aircraft.* Washington DC: National Aeronautics and Space Administration.

Global Industry Analysts, Inc. (2015, September 11). *Smart Augmented Reality (AR) Glasses Market Trends.* Retrieved from http://www.strategyr.com/MarketResearch/Smart_Augmented_Reality_AR_Glasses_Market_Trends.asp

Goteman, O., Smith, K., & Dekker, S. (2007). *HUD With a Velocity Vector Reduces Lateral Error Duing Landing in Restricted Visibility.* Stockholm: Lawrence Erlbaum Associates, Inc.

Hefferon, J. (2017). *Linear Algebra.* Vermont: Saint Michael's College.

Jimenez, C. (2013). *COMPARISON OF THREE ANGLE OF ATTACK (AOA) INDICATORS: A.* Daytona Beach: Embry-Riddle Aeronautical University.

Knotts, L. H., & Priest, J. E. (n.d.). *Turbulence Response Matching in the NT-33A In-Flight Simulator.* AIAA.

Motors, C. (2013). *General Aviation Challenges And Opportunities.* AVIC INTERNATIONAL.

Motta, M. (2004). *Competition policy: theory and practice.* Cambridge University Press.

Newman, R. L., & Haworth, L. A. (n.d.). *Flight Displays II: Head-Up and Helmet-Mounted Displays.* AIAA.

News, F. (2014). *FAA Approved the Installation of Angle of Attack Indicator (AOA) in Small Airplanes.* Online: FAA.

NTSB. (2015). *Prevent Loss of Control In Flight In General Aviation.* Washington DC: National Transportation Safety Board.

Rockwell Collins. (January 2016). *Head-Up Guidance System (HGS) for Midsize and Light Business Aircraft.* Cedar Rapids: Rockwell Collins.

Rogers, D. F., Martos, B., & Rodrigues, F. (2015). *Low-Cost Accurate Angle of Attack System.* Renton: US Department of Transportation.

Starr, G. P. (2006). *Introduction to Applied Digital Control.* New Mexico: The University of New Mexico.

Stengel, R. F. (2004). *Flight Dynamics.* New Jersey: Princeton University Press.

Stevenson, A. (2010). *Head-Up Display.* Oxford University Press.

Szondy, D. (2014, May 11). *Skylens wearable HUD gives pilots augmented vision.* Retrieved from http://newatlas.com/skylens-hud-elbit/31945/

Thales. (2015, June 16). *TOPMAX: THE LIGHTEST ALL-IN-ONE EYES-OUT SOLUTION.* Retrieved from http://onboard.thalesgroup.com/2015/06/16/topmax-lightest-one-eyes-solution/

Thurber, M. (2015, November 12). *TopMax Places Head-worn HUD on Headsets.* Retrieved from http://www.ainonline.com/aviation-news/business-aviation/2015-11-12/topmax-places-head-worn-hud-headsets

Vandel, R., & Weener, E. F. (2009). *Head-Up Guidance System Technology - A Clear Path to Increasing Flight Safety.* Flight Safety Foundation.

# APPENDIX A

## Bosch BNO055 Calibration

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <EEPROM.h> // to save the calibration data

/* Set the delay between fresh samples */
#define BNO055_CALIBRATION_DELAY_MS (1000)

/* The following program fetches different parameters of the IMU */
/* Create the object with address corresponding to a high ADR Pin */
Adafruit_BNO055 bno = Adafruit_BNO055(-1, BNO055_ADDRESS_B);

/* Create an offset struct to store all the calibration offsets */
adafruit_bno055_offsets_t bno_offsets;

/* Read/write addresses */
int write_addr = 0;

/****************************************************************************/
/*
    Arduino setup function (automatically called at startup)
*/
/****************************************************************************/
void setup(void)
{

  Serial.begin(9600);

  /* Initialize the unit in NDOF mode */
  if (!bno.begin(Adafruit_BNO055::OPERATION_MODE_NDOF))
  {
    /* There was a problem detecting the BNO055 ... check your connections */
    Serial.println("CONNECTION_ERROR");
  }

  // Create variables to store calibration statuses
  uint8_t calib_sys(0), calib_accel(0), calib_gyro(0), calib_mag(0);

  // 0x03 identifies a fully calibrated sensor
  while(calib_sys < 0x03 || calib_accel < 0x03 || calib_gyro < 0x03 || calib_mag < 0x03)
  {
    bno.getCalibration(&calib_sys, &calib_gyro, &calib_accel, &calib_mag);
    Serial.print("CALIBRATION_STATUS:");
    Serial.print("s:"); Serial.print(calib_sys); Serial.print(";");
    Serial.print("g:"); Serial.print(calib_gyro); Serial.print(";");
    Serial.print("a:"); Serial.print(calib_accel); Serial.print(";");
    Serial.print("m:"); Serial.print(calib_mag); Serial.println(";");
    delay(BNO055_CALIBRATION_DELAY_MS);
  }

  Serial.println("CALIBRATION_COMPLETE");

  // Get calibration offsets
  bool valid_offset = bno.getSensorOffsets(bno_offsets);

  if(valid_offset) {
```

```
  Serial.println("OBTAINED_CALIBRATION_OFFSETS");

  // Convert offsets to lsb and msb
  EEPROM.write(write_addr, bno_offsets.accel_offset_x);
  write_addr++;
  EEPROM.write(write_addr, bno_offsets.accel_offset_x >> 8);
  write_addr++;

  EEPROM.write(write_addr, bno_offsets.accel_offset_y);
  write_addr++;
  EEPROM.write(write_addr, bno_offsets.accel_offset_y >> 8);
  write_addr++;

  EEPROM.write(write_addr, bno_offsets.accel_offset_z);
  write_addr++;
  EEPROM.write(write_addr, bno_offsets.accel_offset_z >> 8);
  write_addr++;


  EEPROM.write(write_addr, bno_offsets.gyro_offset_x);
  write_addr++;
  EEPROM.write(write_addr, bno_offsets.gyro_offset_x >> 8);
  write_addr++;

  EEPROM.write(write_addr, bno_offsets.gyro_offset_y);
  write_addr++;
  EEPROM.write(write_addr, bno_offsets.gyro_offset_y >> 8);
  write_addr++;

  EEPROM.write(write_addr, bno_offsets.gyro_offset_z);
  write_addr++;
  EEPROM.write(write_addr, bno_offsets.gyro_offset_z >> 8);
  write_addr++;


  EEPROM.write(write_addr, bno_offsets.mag_offset_x);
  write_addr++;
  EEPROM.write(write_addr, bno_offsets.mag_offset_x >> 8);
  write_addr++;

  EEPROM.write(write_addr, bno_offsets.mag_offset_y);
  write_addr++;
  EEPROM.write(write_addr, bno_offsets.mag_offset_y >> 8);
  write_addr++;

  EEPROM.write(write_addr, bno_offsets.mag_offset_z);
  write_addr++;
  EEPROM.write(write_addr, bno_offsets.mag_offset_z >> 8);
  write_addr++;

  EEPROM.write(write_addr, bno_offsets.accel_radius);
  write_addr++;
  EEPROM.write(write_addr, bno_offsets.accel_radius >> 8);
  write_addr++;

  EEPROM.write(write_addr, bno_offsets.mag_radius);
  write_addr++;
  EEPROM.write(write_addr, bno_offsets.mag_radius >> 8);
  write_addr++;

  Serial.println("WROTE_CALIBRATION_OFFSETS");
}
else
  Serial.println("Failed to get calibration offsets");
```

```
  delay(1000);
}

/************************************************************************/
/*
    Arduino loop function, called once 'setup' is complete
*/
/************************************************************************/
void loop(void)
{
}
```

# APPENDIX B

# Accelerometer Data from Bosch BNO055

```cpp
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <EEPROM.h> // to obtain the calibration offsets

/* The following program estimates the acceleration of the IMU when placed in the P6
configuration, shown in the BNO055 manual */

/* Set the delay between fresh samples */
#define BNO055_CALIBRATION_DELAY_MS (1000)
#define BNO055_SAMPLERATE_DELAY_MS (0)

/* Create the object with address corresponding to a high ADR Pin */
Adafruit_BNO055 bno = Adafruit_BNO055(-1, BNO055_ADDRESS_B);

/* Create a vector to store acceleration and euler angle values */
imu::Vector<3> accel;

/* Create an offset struct to store all the calibration offsets */
adafruit_bno055_offsets_t bno_offsets;

int read_addr = 0;

/* Create a string object to store the data to be sent over serial port */
String str = "";

/*************************************************************************/
/*
    Arduino setup function (automatically called at startup)
*/
/*************************************************************************/
void setup(void)
{
  Serial.begin(9600);

  /* Initialise the unit in NDOF mode */
  if (!bno.begin(Adafruit_BNO055::OPERATION_MODE_NDOF))
  {
    /* There was a problem detecting the BNO055 ... check your connections */
    Serial.println("CONNECTION_ERROR");
  }

  uint16_t value = EEPROM.read(read_addr + 1) << 8 | EEPROM.read(read_addr);
  read_addr = read_addr + 2;
  bno_offsets.accel_offset_x = value;

  value = EEPROM.read(read_addr + 1) << 8 | EEPROM.read(read_addr);
  read_addr = read_addr + 2;
  bno_offsets.accel_offset_y = value;

  value = EEPROM.read(read_addr + 1) << 8 | EEPROM.read(read_addr);
  read_addr = read_addr + 2;
  bno_offsets.accel_offset_z = value;
```

```
    value = EEPROM.read(read_addr + 1) << 8 | EEPROM.read(read_addr);
    read_addr = read_addr + 2;
    bno_offsets.gyro_offset_x = value;

    value = EEPROM.read(read_addr + 1) << 8 | EEPROM.read(read_addr);
    read_addr = read_addr + 2;
    bno_offsets.gyro_offset_y = value;

    value = EEPROM.read(read_addr + 1) << 8 | EEPROM.read(read_addr);
    read_addr = read_addr + 2;
    bno_offsets.gyro_offset_z = value;

    value = EEPROM.read(read_addr + 1) << 8 | EEPROM.read(read_addr);
    read_addr = read_addr + 2;
    bno_offsets.mag_offset_x = value;

    value = EEPROM.read(read_addr + 1) << 8 | EEPROM.read(read_addr);
    read_addr = read_addr + 2;
    bno_offsets.mag_offset_y = value;

    value = EEPROM.read(read_addr + 1) << 8 | EEPROM.read(read_addr);
    read_addr = read_addr + 2;
    bno_offsets.mag_offset_z = value;

    value = EEPROM.read(read_addr + 1) << 8 | EEPROM.read(read_addr);
    read_addr = read_addr + 2;
    bno_offsets.accel_radius = value;

    value = EEPROM.read(read_addr + 1) << 8 | EEPROM.read(read_addr);
    read_addr = read_addr + 2;
    bno_offsets.mag_radius = value;

    /* READ_CALIBRATION_OFFSETS */

    delay(1000);
}

/***************************************************************************/
/*
    Arduino loop function, called once 'setup' is complete
*/
/***************************************************************************/
void loop(void)
{
  accel = bno.getVector(Adafruit_BNO055::VECTOR_ACCELEROMETER);

  /* Print the different accelerations */
  str = "";
  str = str + accel.x() + " " + accel.y() + " " + accel.z() + "\n";

  Serial.print(str);

  delay(BNO055_SAMPLERATE_DELAY_MS);
}
```

**APPENDIX C**

**EFRC HUD Source Code**

In this section, the entire source code of the EFRC HUD is provided. Note that the application is developed on Android Studio using the Java programming language. While the following source code is an integral part of the application, it is not alone enough to form the application, it requires the corresponding resource files. If you need assistance with compiling the software, please feel free to reach out to the author at cpavankumar993@gmail.com. Each of the file is separated by a solid line, as an alternative to "page breaks".

```java
package erau.efrc.getOrientation;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Environment;
import android.os.StrictMode;
import android.view.KeyEvent;
import android.view.Window;
import android.view.WindowManager;

import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.io.PrintStream;
import java.util.Calendar;
import java.util.Date;

import erau.efrc.getOrientation.Gauges.Modes.Cruise;
import erau.efrc.getOrientation.Gauges.Modes.Approach;
import erau.efrc.getOrientation.Gauges.Modes.Takeoff;

import static android.os.Environment.getExternalStorageState;

public class MainActivity extends Activity {

    public static volatile Params hudParams = new Params();

    public static volatile float headPitch = 0.0f;

    // Based on a measured estimate of the pitch attitude of the FRL of 5 deg and 4.32 deg of Bosch
    public static final float headPitchBias = 0f; // degrees
    // public static final float headPitchBias = 2.16f; // degrees

    private float[] CoeffArray = new float[] {0.003f, 0.01f, 0.05f};
```

```java
    private int indCoeff = 1;

    public static volatile float FILTER_COEFFICIENT = 0.1f;

    // Different modes
    private Takeoff takeoff = null;
    private Cruise cruise = null;
    private Approach approach = null;

    // Toggle between hud modes
    public static volatile Modes mode = Modes.takeoff;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        // Creating a File object to store data
        try {

            File file = new File(Environment.getExternalStoragePublicDirectory(
                    Environment.DIRECTORY_DCIM), "/" + Calendar.getInstance().getTimeInMillis() +
".txt");

            System.setOut(new PrintStream(new FileOutputStream(file)));

        }
        catch (Exception e) {
            e.printStackTrace();
        }

        StrictMode.ThreadPolicy policy = new
StrictMode.ThreadPolicy.Builder().permitAll().build();
        StrictMode.setThreadPolicy(policy);

        // Set full-screen
        Window win = getWindow();
        WindowManager.LayoutParams winParams = win.getAttributes();
        winParams.flags |= 0x80000000; // FLAG_SMARTFULLSCREEN = 0x80000000

        switch (mode) {
            case takeoff:
                takeoff = new Takeoff(this);
                setContentView(takeoff);
                break;

            case cruise:
                cruise = new Cruise(this);
                setContentView(cruise);
                break;

            case approach:
                approach = new Approach(this);
                setContentView(approach);
                break;
        }

        // Start the HeadTracker
        Intent i = new Intent(this, HeadTracker.class);
        startService(i);

        // Start the CommChannel
        Intent j = new Intent(this, CommChannel.class);
        startService(j);
    }
```

```java
@Override
public boolean dispatchKeyEvent(KeyEvent event) {
    int action = event.getAction();
    int keyCode = event.getKeyCode();
    switch (keyCode) {
        case KeyEvent.KEYCODE_VOLUME_UP:
            if (action == KeyEvent.ACTION_DOWN) {
                // Ensure the index is bound within the upper limit
                if(indCoeff < CoeffArray.length - 1) {
                    indCoeff = indCoeff + 1;
                    FILTER_COEFFICIENT = CoeffArray[indCoeff];
                }
            }
            return true;
        case KeyEvent.KEYCODE_VOLUME_DOWN:
            if (action == KeyEvent.ACTION_DOWN) {
                // Ensure the index is bound within the lower limit
                if (indCoeff > 0)
                {
                    indCoeff = indCoeff - 1;
                    FILTER_COEFFICIENT = CoeffArray[indCoeff];
                }
            }
            return true;
        default:
            return super.dispatchKeyEvent(event);
    }
}

@Override
public void onPause() {

    super.onPause();

    switch (mode) {
        case takeoff:
            takeoff.pause();
            break;

        case cruise:
            cruise.pause();
            break;

        case approach:
            approach.pause();
            break;
    }
}

@Override
public void onResume() {

    super.onResume();

    switch (mode) {
        case takeoff:
            takeoff.resume();
            break;

        case cruise:
            cruise.resume();
            break;

        case approach:
```

```
                    approach.resume();
                    break;
            }
        }
}
```

```
package erau.efrc.getOrientation;

import android.app.Service;
import android.content.Intent;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.IBinder;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.InetSocketAddress;
import java.net.ServerSocket;
import java.net.Socket;

import static erau.efrc.getOrientation.MainActivity.FILTER_COEFFICIENT;
import static erau.efrc.getOrientation.MainActivity.headPitch;
import static erau.efrc.getOrientation.MainActivity.headPitchBias;
import static java.lang.Float.NaN;

public class HeadTracker extends Service {

    Thread listenThread = null;

    private static final int TCP_SERVER_PORT = 55000;

    @Override
    public void onCreate() {

        super.onCreate();

        listenThread = new Thread(new listenTCP());

        listenThread.start();
    }

    private class listenTCP implements Runnable {

        public void run() {
            ServerSocket serverSocket = null;
            try {

                // If the port has already been used, try re-using
                serverSocket = new ServerSocket();
                serverSocket.setReuseAddress(true);
                serverSocket.bind(new InetSocketAddress(TCP_SERVER_PORT));

                // Look for incoming connections
                Socket socket = serverSocket.accept();

                // Spin a new thread for the incoming connection
                new Thread(new CommTCP(socket)).start();
```

```
                    // Close the serverSocket so it can be re-used without any issues
                    serverSocket.close();

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    private class CommTCP implements Runnable {

        // Sensor Manager
        private SensorManager mSensorManager = null;

        Sensor accel = null;
        Sensor rotate = null;

        // Define SENSOR_DELAY: ensure the data-rate is high enough to ensure a quick convergence
        private int SENSOR_DELAY = 30000;

        private Socket socket;

        private BufferedWriter output;
        private BufferedReader input;

        private sensorEventListener sensorListener;

        CommTCP(Socket socket) {

            this.socket = socket;

            try {
                output = new BufferedWriter(new
OutputStreamWriter(this.socket.getOutputStream()));
                input = new BufferedReader(new InputStreamReader(this.socket.getInputStream()));
            }
            catch (IOException e) {
                e.printStackTrace();
            }

            // Sensor Manager
            mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

            accel = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
            rotate = mSensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);

            sensorListener = new sensorEventListener();
        }

        public void run() {

            // Use the TYPE_ROTATION_VECTOR to calibrate the TYPE_ACCELEROMETER vector
            //mSensorManager.registerListener(sensorListener, accel, SENSOR_DELAY);
            mSensorManager.registerListener(sensorListener, rotate,
SensorManager.SENSOR_DELAY_FASTEST);
        }

        class sensorEventListener implements SensorEventListener {

            public void onAccuracyChanged(Sensor sensor, int accuracy) {
            }

            public void onSensorChanged(SensorEvent event) {

                if(event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
```

```
                        // remap event.values
                        float[] android = new float[3];

                        android[0] = -event.values[2];
                        android[1] = event.values[0];
                        android[2] = -event.values[1];

                        try {
                            // Epson BT-200 requires the following sign convention mapping
                            output.write(' ');
                            output.flush();

                            String Str = input.readLine();

                            estimatePitch(android, Str);

                        } catch (IOException e) {

                            e.printStackTrace();

                            // If an error has occurred, unregister the sensor and start the listening
thread
                            mSensorManager.unregisterListener(sensorListener);
                            listenThread = new Thread(new listenTCP());
                            listenThread.start();
                        }
                    }
                    else if(event.sensor.getType() == Sensor.TYPE_ROTATION_VECTOR) {

                        float q1 = event.values[0];
                        float q2 = event.values[1];
                        float q3 = event.values[2];
                        float q0 = (float) Math.sqrt(1 - q1*q1 - q2*q2 - q3*q3); // Its a unit
quaternion

                        // Formulas are obtained from
https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles
                        float pitch = (float) (Math.toDegrees(Math.atan2(2f * (q0*q1 + q2*q3), 1f - 2f
* (q1*q1 + q2*q2))) - 90);

                        if(!Float.isNaN(pitch)) {
                            headPitch = headPitch + (pitch - headPitch) * FILTER_COEFFICIENT;
                            //System.out.println(headPitch + " " + FILTER_COEFFICIENT);
                        }

                        //System.out.println(pitch);

                    }
                }
            }
        }

    private void estimatePitch(float[] android, String boschStr) {

        // Convert the given string to integers
        String[] valuesArray = boschStr.split(" ");
        float[] bosch = new float[3];

        for(int i = 0; i <= 2; i++) {

            try {

                bosch[i] = Float.parseFloat(valuesArray[i]);
```

```
            } catch (Exception e) {

                return;
            }
        }

        float boschMag = (float) Math.sqrt(bosch[0] * bosch[0] + bosch[1] * bosch[1] + bosch[2] *
bosch[2]);
        float androidMag = (float) Math.sqrt(android[0] * android[0] + android[1] * android[1] +
android[2] * android[2]);

        for(int i = 0; i<= 2; i++) {

            bosch[i] = bosch[i] / boschMag;
            android[i] = android[i] / androidMag;
        }

        float c = bosch[0] * android[0] + bosch[1] * android[1] + bosch[2] * android[2];

        float v1 = bosch[1] * android[2] - bosch[2] * android[1];
        float v2 = bosch[2] * android[0] - bosch[0] * android[2];
        float v3 = bosch[0] * android[1] - bosch[1] * android[0];

        float R32 = v1 + v2 * v3 / (1 + c);
        float R31 = -v2 + v1 * v3 / (1 + c);

        float theta = (float) Math.toDegrees(Math.asin(R31 / Math.cos(Math.asin(-R32)))) +
headPitchBias;

        if(!Float.isNaN(theta)) {
            headPitch = headPitch + (theta - headPitch) * FILTER_COEFFICIENT;
            //System.out.println(headPitch + " " + FILTER_COEFFICIENT);
        }

        return;
    }

    // Ensure CommChannel service is always started using startService() and not bindService()
    @Override
    public IBinder onBind(Intent intent) {

        return null;
    }
}
```

---

```
package erau.efrc.getOrientation;

import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.InetSocketAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Calendar;

import static erau.efrc.getOrientation.MainActivity.FILTER_COEFFICIENT;
import static erau.efrc.getOrientation.MainActivity.headPitch;
import static erau.efrc.getOrientation.MainActivity.hudParams;
```

```java
// Format of incoming messages: START_DATA:airspeed;altitude;roll;pitch;heading;flightpath;aoa;\n
public class CommChannel extends Service {

    Thread listenThread = null;
    private static final int TCP_SERVER_PORT = 50000;
    private final String HEADER = "START_DATA:";

    private final float AIRSPEED_MIN = 0;
    private final float AIRSPEED_MAX = 500;
    private final float ALTITUDE_MIN = -2000;
    private final float ALTITUDE_MAX = 40000;
    private final float ROLL_MIN = -30;
    private final float ROLL_MAX = 30;
    private final float PITCH_MIN = -50;
    private final float PITCH_MAX = 50;
    private final float HEADING_MIN = 0;
    private final float HEADING_MAX = 360;
    private final float FLIGHT_PATH_MIN = -45;
    private final float FLIGHT_PATH_MAX = 45;
    private final float AOA_MIN = -15;
    private final float AOA_MAX = 30;

    @Override
    public void onCreate() {

        super.onCreate();

        listenThread = new Thread(new listenTCP());
        listenThread.start();
    }

    private class listenTCP implements Runnable {

        public void run() {
            ServerSocket serverSocket = null;
            try {

                // If the port has already been used, try re-using
                serverSocket = new ServerSocket();
                serverSocket.setReuseAddress(true);
                serverSocket.bind(new InetSocketAddress(TCP_SERVER_PORT));

                // Look for incoming connections
                Socket socket = serverSocket.accept();

                // Spin a new thread for the incoming connection
                new Thread(new CommTCP(socket)).start();

                // Close the serverSocket so it can be re-used without any issues
                serverSocket.close();

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    private class CommTCP implements Runnable {

        private Socket socket;

        private BufferedReader input;

        CommTCP(Socket socket) {
```

```java
            this.socket = socket;
            try {
                this.input = new BufferedReader(new
InputStreamReader(this.socket.getInputStream())));
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        public void run() {
            while(!Thread.currentThread().isInterrupted()) {
                try {
                    String read = input.readLine(); // readLine returns null only when the
connection is closed

                    if(read != null) {
                        updateHudParams(read, hudParams);
                        System.out.println(Calendar.getInstance().getTimeInMillis() + " " +
headPitch + " " + hudParams.pitch + " " + hudParams.flightPath
                                + " " + FILTER_COEFFICIENT);
                    }
                    else {
                        listenThread = new Thread(new listenTCP());
                        listenThread.start();
                        return;
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    private void updateHudParams(String packet, Params hudParams) {

        int lastIndex = findLastIndex(packet, HEADER);

        if(lastIndex == -1) {
            return;
        }

        // Remove HEADER from the string
        packet = packet.substring(lastIndex);

        int counter = 0;

        while (packet.length() > 0 && counter <= 6)
        {
            lastIndex = packet.indexOf(";");

            float currData;

            try {
                currData = Float.valueOf(packet.substring(0, lastIndex));
            } catch (Exception e) {

                currData = 0;
            }

            packet = packet.substring(lastIndex + 1);

            counter++;

            switch (counter) {
                case 1: {
```

```
                            if (currData >= AIRSPEED_MIN && currData <= AIRSPEED_MAX) {
                                hudParams.airspeed = currData;
                            }
                            break;
                        }
                        case 2: {
                            if (currData >= ALTITUDE_MIN && currData <= ALTITUDE_MAX) {
                                hudParams.altitude = currData;
                            }
                            break;
                        }
                        case 3: {
                            if (currData >= ROLL_MIN && currData <= ROLL_MAX) {
                                hudParams.roll = currData;
                            }
                            break;
                        }
                        case 4: {
                            if (currData >= PITCH_MIN && currData <= PITCH_MAX) {
                                hudParams.pitch = currData;
                            }
                            break;
                        }
                        case 5: {
                            if (currData >= HEADING_MIN && currData <= HEADING_MAX) {
                                hudParams.heading = currData;
                            }
                            break;
                        }
                        case 6: {
                            if (currData >= FLIGHT_PATH_MIN && currData <= FLIGHT_PATH_MAX) {
                                hudParams.flightPath = currData;
                            }
                            break;
                    }
                        case 7: {
                            if(currData >= AOA_MIN && currData <= AOA_MAX) {
                                hudParams.angleOfAttack = currData;
                            }
                            break;
                        }
                    }
                }
            }

    // Find the position after a substring in a string
    private int findLastIndex(String str, String pattern) {

        if(str.contains(pattern)) {
            return str.indexOf(pattern) + pattern.length();
        }
        else {
            return -1;
        }
    }

    // Ensure CommChannel service is always started using startService() and not bindService()
    @Override
    public IBinder onBind(Intent intent) {

        return new Binder();
    }
}
```

```
package erau.efrc.getOrientation;

public enum Modes {
    takeoff,
    cruise,
    approach
}
```

```
package erau.efrc.getOrientation;

public class Params {
    public float airspeed = 0,
            altitude = 0,
            roll = 0,
            pitch = 0,
            heading = 0,
            angleOfAttack = 0,
            flightPath = 0;
}
```

```
package erau.efrc.getOrientation.Gauges;

import android.graphics.Canvas;
import android.graphics.PointF;

public interface Gauge {

    void draw(Canvas canvas, PointF currLocation, float... currVals);
}
```

```
package erau.efrc.getOrientation.Gauges;

public enum Directions {
    LEFT,
    RIGHT,
    DOWN,
    HORIZONTAL
}
```

```
package erau.efrc.getOrientation.Gauges.Symmteric;

import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.DashPathEffect;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.PointF;
import android.graphics.Rect;

import erau.efrc.getOrientation.Gauges.Gauge;

abstract class SymmetricGauge implements Gauge {

    float UNITS_PER_GRADUATION = 3;
    float OVERALL_VALUE;                                    // Overall value the gauge can
display in one frame
    float LARGER_MARGIN_VAL = 3;                            // Larger graduation value

    float GAUGE_HEIGHT;                                     // Overall gauge height (px)
```

```
    float LARGER_MARGIN_LEN = 75;                              // Larger graduation length,
(px)
    float HORIZON_LEN = 200;                                   // Horizon length (px)
    float CENTER_GAP = 20;
    float DASH_FILL_LEN = 10;                                  // Dash length in dashed line,
for negative value (px)
    float DASH_GAP_LEN = 20;                                   // Gap in dashed line, for
nagative value (px)
    float CROSS_HAIR_LEN = 10;
    float FLIGHT_PATH_RAD = 5;

    String MAX_TEXT = "XXX";                                   // Maximum number of digits the
gauge can represent
    int STROKE_WIDTH = 1;
    int TEXT_SIZE = 20;

    private PointF GRAD_DIR = new PointF();
    private PointF HORIZON_DIR = new PointF();
    private PointF LADDER_DIR = new PointF();

    private final float UNITS_PER_PIXEL;
    private final float MARGIN_SPACING;          // Spacing between any two adjacent graduations on
the gauge (px)
    private final Paint pathPaint;
    private final Paint textPaint;
    private final Paint negativePaint;
    private final float textHeight;
    private final float textWidth;

    SymmetricGauge() {

        // Define all the characteristics of the derived classes
        // NOTE: Ensure this is done right at the beginning of the constructor method, to
        // get a correct customization of the gauge
        defineGaugeChars();

        /* Initialize paint brushes */
        pathPaint = new Paint();
        pathPaint.setColor(Color.GREEN);
        pathPaint.setStrokeWidth(STROKE_WIDTH);
        pathPaint.setStyle(Paint.Style.STROKE);

        textPaint = new Paint();
        textPaint.setColor(Color.GREEN);
        textPaint.setStyle(Paint.Style.FILL);
        textPaint.setTextSize(TEXT_SIZE);
        textPaint.setTextAlign(Paint.Align.CENTER);

        negativePaint = new Paint();
        negativePaint.setColor(Color.GREEN);
        negativePaint.setStrokeWidth(STROKE_WIDTH);
        negativePaint.setStyle(Paint.Style.STROKE);

        float[] intervals = new float[]{DASH_FILL_LEN, DASH_GAP_LEN};
        float phase = 0;

        DashPathEffect dashPathEffect =
                new DashPathEffect(intervals, phase);

        negativePaint.setPathEffect(dashPathEffect);

        // Get height of a generic text
        Rect rect = new Rect();
        textPaint.getTextBounds("0123456789", 0, 10, rect);
        textHeight = rect.height();
```

```
        // Assume the value never goes higher than MAX_TEXT
        textWidth = textPaint.measureText(MAX_TEXT);

        // Initialize all the dependent variables
        UNITS_PER_PIXEL = OVERALL_VALUE / GAUGE_HEIGHT;

        // MARGIN_SPACING must be an integer, else, there will be round-off error in the final
calculations
        MARGIN_SPACING = GAUGE_HEIGHT / (OVERALL_VALUE/UNITS_PER_GRADUATION);

        // The horizon direction is always leveled
        HORIZON_DIR = new PointF((float) Math.cos(0),
                (float) Math.sin(0));
    }

    // Must be implemented by the derived class to customize the gauge
    abstract void defineGaugeChars();

    public void draw(Canvas canvas, PointF drawLocation, float... currVals) {

        float theta = currVals[0];

        // Estimate GRAD_DIR and LADDER_DIR
        GRAD_DIR = new PointF((float) Math.cos(Math.toRadians(theta)),
                (float) Math.sin(Math.toRadians(theta)));

        LADDER_DIR = new PointF((float) Math.cos(Math.toRadians(theta + 90)),
                (float) Math.sin(Math.toRadians(theta + 90)));

        float centerVal = currVals[1];

        // Estimate the number of units to the nearest valid value (HIGHER than or equal to the
current value)
        // NOTE: A valid value is one that can be represented on a graduation
        float unitsAway = centerVal % UNITS_PER_GRADUATION == 0 ? 0 : UNITS_PER_GRADUATION -
centerVal % UNITS_PER_GRADUATION;

        // Estimate the nearest valid value (HIGHER than or equal to the current value)
        float tempVal = centerVal + unitsAway;

        // Estimate the number of pixels to the nearest valid value, e.g. 3 units is 3 units /
UNITS_PER_PIXEL away
        float pixelsAway = unitsAway / UNITS_PER_PIXEL;

        // Estimate location of the nearest valid value
        PointF i = new PointF();
        i.x = drawLocation.x - LADDER_DIR.x * pixelsAway;
        i.y = drawLocation.y - LADDER_DIR.y * pixelsAway;

        // Calculate length of the gauge sketched so far
        float gaugeLen = calcDistance(i, drawLocation);

        Path positivePath = new Path();
        Path negativePath = new Path();

        // Draw the first portion of the gauge
        while (gaugeLen <= GAUGE_HEIGHT / 2)
        {
            if(tempVal >= 0) {

                // Draw the larger margin, if the current value is a multiple of LARGER_MARGIN_VAL
                drawMargins(canvas, positivePath, i, tempVal);

            }
```

```
        else {

            // Draw the larger margin, if the current value is a multiple of LARGER_MARGIN_VAL
            drawMargins(canvas, negativePath, i, -tempVal);
        }

        tempVal = updateCounter(i, tempVal, -1);

        gaugeLen = calcDistance(i, drawLocation);
    }

    // Estimate the nearest valid value (LOWER than or equal to the current value)
    unitsAway = centerVal % UNITS_PER_GRADUATION == 0 ? 0 : centerVal % UNITS_PER_GRADUATION;

    pixelsAway = unitsAway / UNITS_PER_PIXEL;

    tempVal = centerVal - unitsAway;

    // Reset the value of i
    i.x = drawLocation.x + LADDER_DIR.x * pixelsAway;
    i.y = drawLocation.y + LADDER_DIR.y * pixelsAway;

    // Ensure the center graduation is NOT drawn twice
    if(pixelsAway == 0) {

        tempVal = updateCounter(i, tempVal, +1);
    }

    gaugeLen = calcDistance(i, drawLocation);

    // Draw the second portion of the gauge
    while (gaugeLen <= GAUGE_HEIGHT / 2) {
        if(tempVal >= 0) {

            // Draw the larger margin, if the current value is a multiple of LARGER_MARGIN_VAL
            drawMargins(canvas, positivePath, i, tempVal);

        }
        else {

            // Draw the larger margin, if the current value is a multiple of LARGER_MARGIN_VAL
            drawMargins(canvas, negativePath, i, -tempVal);
        }

        // Update the counter
        tempVal = updateCounter(i, tempVal, +1);

        // Calculate length of the gauge sketched so far
        gaugeLen = calcDistance(i, drawLocation);
    }

    // Draw cross-hairs at the appropriate location
    float pitch = currVals[2];
    pixelsAway = (pitch - centerVal) / UNITS_PER_PIXEL;

    if(Math.abs(pixelsAway) <= GAUGE_HEIGHT / 2) {

        drawCrossHairs(positivePath, new PointF(drawLocation.x - LADDER_DIR.x * pixelsAway,
                drawLocation.y - LADDER_DIR.y * pixelsAway));
    }

    // Draw flight-path marker at the appropriate location
    float flightPath = currVals[3];
    pixelsAway = (flightPath - centerVal) / UNITS_PER_PIXEL;
```

```java
    if(Math.abs(pixelsAway) <= GAUGE_HEIGHT / 2) {

        drawFlightPath(positivePath, new PointF(drawLocation.x - LADDER_DIR.x * pixelsAway,
                drawLocation.y - LADDER_DIR.y * pixelsAway));
    }

    canvas.drawPath(positivePath, pathPaint);

    canvas.drawPath(negativePath, negativePaint);
}

private void drawCrossHairs(Path path, PointF i) {

    // In positive GRAD direction
    path.moveTo(i.x, i.y);
    path.lineTo(i.x + GRAD_DIR.x * CROSS_HAIR_LEN,
            i.y + GRAD_DIR.y * CROSS_HAIR_LEN);

    // In negative GRAD direction
    path.moveTo(i.x, i.y);
    path.lineTo(i.x - GRAD_DIR.x * CROSS_HAIR_LEN,
            i.y - GRAD_DIR.y * CROSS_HAIR_LEN);

    // In positive LADDER direction
    path.moveTo(i.x, i.y);
    path.lineTo(i.x + LADDER_DIR.x * CROSS_HAIR_LEN,
            i.y + LADDER_DIR.y * CROSS_HAIR_LEN);

    // In negative LADDER direction
    path.moveTo(i.x, i.y);
    path.lineTo(i.x - LADDER_DIR.x * CROSS_HAIR_LEN,
            i.y - LADDER_DIR.y * CROSS_HAIR_LEN);
}

private void drawFlightPath(Path path, PointF i) {

    // In positive GRAD direction
    path.moveTo(i.x + GRAD_DIR.x * FLIGHT_PATH_RAD,
            i.y + GRAD_DIR.y * FLIGHT_PATH_RAD);
    path.lineTo(i.x + GRAD_DIR.x * (FLIGHT_PATH_RAD + CROSS_HAIR_LEN),
            i.y + GRAD_DIR.y * (FLIGHT_PATH_RAD + CROSS_HAIR_LEN));

    // In negative GRAD direction
    path.moveTo(i.x - GRAD_DIR.x * FLIGHT_PATH_RAD,
            i.y - GRAD_DIR.y * FLIGHT_PATH_RAD);
    path.lineTo(i.x - GRAD_DIR.x * (FLIGHT_PATH_RAD + CROSS_HAIR_LEN),
            i.y - GRAD_DIR.y * (FLIGHT_PATH_RAD + CROSS_HAIR_LEN));

    // In negative LADDER direction
    path.moveTo(i.x - LADDER_DIR.x * FLIGHT_PATH_RAD,
            i.y - LADDER_DIR.y * FLIGHT_PATH_RAD);
    path.lineTo(i.x - LADDER_DIR.x * (FLIGHT_PATH_RAD + CROSS_HAIR_LEN),
            i.y - LADDER_DIR.y * (FLIGHT_PATH_RAD + CROSS_HAIR_LEN));

    // Center circle
    path.addCircle(i.x, i.y, FLIGHT_PATH_RAD, Path.Direction.CCW);
}

private void drawMargins(Canvas canvas, Path path, PointF i, float tempVal) {

    // Prevent any loss in precision upto 3 decimal places
    tempVal = Math.round(tempVal * 1000)/1000;

    // Draw horizon line, if the current value is 0
    if(tempVal == 0) {
```

```
            // Draw on positive side
            path.moveTo(i.x + HORIZON_DIR.x * CENTER_GAP, i.y + HORIZON_DIR.y * CENTER_GAP);
            path.lineTo(i.x + HORIZON_DIR.x * (CENTER_GAP + HORIZON_LEN),
                    i.y + HORIZON_DIR.y * (CENTER_GAP + HORIZON_LEN));

            // Draw on negative side
            path.moveTo(i.x - HORIZON_DIR.x * CENTER_GAP, i.y - HORIZON_DIR.y * CENTER_GAP);
            path.lineTo(i.x - HORIZON_DIR.x * (CENTER_GAP + HORIZON_LEN),
                    i.y - HORIZON_DIR.y * (CENTER_GAP + HORIZON_LEN));
        }
        // Draw larger margin, if the current value is a multiple of LARGER_MARGIN_VAL
        else if (tempVal % LARGER_MARGIN_VAL == 0)
        {
            // Draw on positive side
            path.moveTo(i.x + GRAD_DIR.x * CENTER_GAP, i.y + GRAD_DIR.y * CENTER_GAP);
            path.lineTo(i.x + GRAD_DIR.x * (CENTER_GAP + LARGER_MARGIN_LEN),
                    i.y + GRAD_DIR.y * (CENTER_GAP + LARGER_MARGIN_LEN));

            // Draw on negative side
            path.moveTo(i.x - GRAD_DIR.x * CENTER_GAP, i.y - GRAD_DIR.y * CENTER_GAP);
            path.lineTo(i.x - GRAD_DIR.x * (CENTER_GAP + LARGER_MARGIN_LEN),
                    i.y - GRAD_DIR.y * (CENTER_GAP + LARGER_MARGIN_LEN));

            // Draw a new line to properly place the text
            placeText(canvas, i, Integer.toString((int) tempVal));
        }
    }

    private void placeText(Canvas canvas, PointF i, String str) {

        Path textPath = new Path();

        textPath.moveTo(i.x + GRAD_DIR.x * (CENTER_GAP + LARGER_MARGIN_LEN) + LADDER_DIR.x *
textHeight / 2,
                i.y + GRAD_DIR.y * (CENTER_GAP + LARGER_MARGIN_LEN) + LADDER_DIR.y * textHeight /
2);

        textPath.lineTo(i.x + GRAD_DIR.x * (CENTER_GAP + LARGER_MARGIN_LEN + textWidth) +
LADDER_DIR.x * textHeight / 2,
                i.y + GRAD_DIR.y * (CENTER_GAP + LARGER_MARGIN_LEN + textWidth) + LADDER_DIR.y *
textHeight / 2);

        canvas.drawTextOnPath(str, textPath, 0, 0, textPaint);

        textPath.rewind();

        textPath.moveTo(i.x - GRAD_DIR.x * (CENTER_GAP + LARGER_MARGIN_LEN + textWidth) +
LADDER_DIR.x * textHeight / 2,
                i.y - GRAD_DIR.y * (CENTER_GAP + LARGER_MARGIN_LEN + textWidth) + LADDER_DIR.y *
textHeight / 2);

        textPath.lineTo(i.x - GRAD_DIR.x * (CENTER_GAP + LARGER_MARGIN_LEN) + LADDER_DIR.x *
textHeight / 2,
                i.y - GRAD_DIR.y * (CENTER_GAP + LARGER_MARGIN_LEN) + LADDER_DIR.y * textHeight /
2);

        canvas.drawTextOnPath(str, textPath, 0, 0, textPaint);
    }

    // Update the counter location variable and return the current value
    private float updateCounter(PointF i, float tempVal, int sign) {

        i.x = i.x + sign * LADDER_DIR.x * MARGIN_SPACING;
        i.y = i.y + sign * LADDER_DIR.y * MARGIN_SPACING;
```

```
            return(tempVal - sign * (MARGIN_SPACING * UNITS_PER_PIXEL));
    }

    // Calculate distance between two points using distance formula
    private float calcDistance(PointF start, PointF end) {
        return (float) Math.hypot(start.x - end.x, start.y - end.y);
    }
}
```

```
package erau.efrc.getOrientation.Gauges.Symmteric;

public class Pitch extends SymmetricGauge {

    public Pitch() {  super(); }

    void defineGaugeChars() {

        float PIXELS_PER_DEGREE = 540 / (23 * 9 /16); // pixels / field_of_view -> vertical

        OVERALL_VALUE = 12.f;
        GAUGE_HEIGHT = OVERALL_VALUE * PIXELS_PER_DEGREE;
    }
}
```

```
package erau.efrc.getOrientation.Gauges.Modes;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.PointF;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

import erau.efrc.getOrientation.Gauges.Curvelinear.Roll;
import erau.efrc.getOrientation.Gauges.FlightPathMarker.FlightPathMarkerGauge;
import erau.efrc.getOrientation.Gauges.Gauge;
import erau.efrc.getOrientation.Gauges.Linear.Airspeed;
import erau.efrc.getOrientation.Gauges.Linear.Altitude;
import erau.efrc.getOrientation.Gauges.Linear.Heading;

import static erau.efrc.getOrientation.MainActivity.headPitch;
import static erau.efrc.getOrientation.MainActivity.hudParams;

public class Approach extends SurfaceView implements Runnable {

    Thread thread = null;
    boolean canDraw = false;

    Canvas canvas;
    SurfaceHolder surfaceHolder;

    private Gauge airspeed;
    private Gauge altitude;
    private Gauge heading;
    private Gauge fpm;
    private Gauge roll;

    public Approach(Context context) {

        super(context);

        surfaceHolder = getHolder();
```

```java
        airspeed = new Airspeed();

        altitude = new Altitude();

        roll = new Roll();

        heading = new Heading();

        fpm = new FlightPathMarkerGauge(context);
    }

    @Override
    public void run() {

        while(canDraw) {
            // Carry out some drawing
            if(surfaceHolder.getSurface().isValid()) {

                canvas = surfaceHolder.lockCanvas();

                canvas.drawColor(Color.BLACK);

                // Add head-tracker - calibrated for head position of -3.5f degrees
                this.drawGauges(hudParams.airspeed, hudParams.altitude, hudParams.roll,
hudParams.heading,
                        headPitch + hudParams.pitch, hudParams.flightPath,
hudParams.angleOfAttack);

                surfaceHolder.unlockCanvasAndPost(canvas);
            }
        }
    }

    private void drawGauges(float speedV, float altitudeV, float rollV, float headingV, float
headV, float flightPathV,
                            float angleOfAttackV) {

        airspeed.draw(canvas, new PointF(getWidth() * 0.15f, getHeight() * 0.5f), speedV);

        altitude.draw(canvas, new PointF(getWidth() * 0.85f, getHeight() * 0.5f), altitudeV);

        heading.draw(canvas, new PointF(getWidth() * 0.5f, 0), headingV);

        roll.draw(canvas, new PointF(getWidth() * 0.5f, getHeight() * 0.8f), rollV);

        fpm.draw(canvas, new PointF(getWidth() * 0.5f, getHeight() * 0.425f), rollV, headV,
flightPathV, angleOfAttackV);
    }

    public void pause() {

        canDraw = false;

        while(thread != null) {
            try {
                thread.join();
                thread = null;
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public void resume() {
```

```
            canDraw = true;
            thread = new Thread(this);
            thread.start();
        }
}
```

---

```
package erau.efrc.getOrientation.Gauges.Modes;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.PointF;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

import erau.efrc.getOrientation.Gauges.Curvelinear.Roll;
import erau.efrc.getOrientation.Gauges.FlightPathMarker.FlightPathMarkerGauge;
import erau.efrc.getOrientation.Gauges.Gauge;
import erau.efrc.getOrientation.Gauges.Linear.Altitude;
import erau.efrc.getOrientation.Gauges.Linear.Heading;

import static erau.efrc.getOrientation.MainActivity.headPitch;
import static erau.efrc.getOrientation.MainActivity.hudParams;

public class Cruise extends SurfaceView implements Runnable {

    Thread thread = null;
    boolean canDraw = false;

    Canvas canvas;
    SurfaceHolder surfaceHolder;

    private Gauge altitude;
    private Gauge heading;
    private Gauge fpm;
    private Gauge roll;

    public Cruise(Context context) {

        super(context);

        surfaceHolder = getHolder();

        altitude = new Altitude();

        roll = new Roll();

        heading = new Heading();

        fpm = new FlightPathMarkerGauge(context);
    }

    @Override
    public void run() {

        while(canDraw) {
            // Carry out some drawing
            if(surfaceHolder.getSurface().isValid()) {

                canvas = surfaceHolder.lockCanvas();

                canvas.drawColor(Color.BLACK);
```

```
                // Add head-tracker
                this.drawGauges(hudParams.altitude, hudParams.roll, hudParams.heading,
                        headPitch + hudParams.pitch, hudParams.flightPath,
hudParams.angleOfAttack);

                surfaceHolder.unlockCanvasAndPost(canvas);
            }
        }
    }

    private void drawGauges(float altitudeV, float rollV, float headingV, float headV, float
flightPathV,
                            float angleOfAttackV) {

        altitude.draw(canvas, new PointF(getWidth() * 0.85f, getHeight() * 0.5f), altitudeV);

        heading.draw(canvas, new PointF(getWidth() * 0.5f, 0), headingV);

        roll.draw(canvas, new PointF(getWidth() * 0.5f, getHeight() * 0.8f), rollV);

        fpm.draw(canvas, new PointF(getWidth() * 0.5f, getHeight() * 0.425f), rollV, headV,
                flightPathV, angleOfAttackV);
    }

    public void pause() {

        canDraw = false;

        while(thread != null) {
            try {
                thread.join();
                thread = null;
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public void resume() {

        canDraw = true;
        thread = new Thread(this);
        thread.start();
    }
}
```

```
package erau.efrc.getOrientation.Gauges.Modes;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.PointF;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

import erau.efrc.getOrientation.Gauges.Curvelinear.Roll;
import erau.efrc.getOrientation.Gauges.Gauge;
import erau.efrc.getOrientation.Gauges.Linear.Airspeed;
import erau.efrc.getOrientation.Gauges.Linear.Altitude;
import erau.efrc.getOrientation.Gauges.Linear.Heading;
import erau.efrc.getOrientation.Gauges.Symmteric.Pitch;
import erau.efrc.getOrientation.Gauges.AngleOfAttack.NormalAoA;
```

```java
import static erau.efrc.getOrientation.MainActivity.headPitch;
import static erau.efrc.getOrientation.MainActivity.hudParams;

public class Takeoff extends SurfaceView implements Runnable {

    Thread thread = null;
    boolean canDraw = false;

    Canvas canvas;
    SurfaceHolder surfaceHolder;

    private Gauge airspeed;
    private Gauge altitude;
    private Gauge heading;
    private Gauge roll;
    private Gauge pitch;
    private Gauge aoa;

    public Takeoff(Context context) {
        super(context);

        surfaceHolder = getHolder();

        airspeed = new Airspeed();

        altitude = new Altitude();

        heading = new Heading();

        roll = new Roll();

        pitch = new Pitch();

        aoa = new NormalAoA(context);
    }

    @Override
    public void run() {

        while(canDraw) {
            // Carry out some drawing
            if(surfaceHolder.getSurface().isValid()) {

                canvas = surfaceHolder.lockCanvas();
                canvas.drawColor(Color.BLACK);

                // Add head-tracker - calibrated for head position of -3.5f degrees
                this.drawGauges(hudParams.airspeed, hudParams.altitude, hudParams.roll,
                        hudParams.heading, headPitch + hudParams.pitch, hudParams.pitch,
hudParams.flightPath,
                        hudParams.angleOfAttack);

                surfaceHolder.unlockCanvasAndPost(canvas);
            }
        }
    }

    private void drawGauges(float speedV, float altitudeV, float rollV, float headingV, float
headV,
                            float pitchV, float flightPathV, float angleOfAttackV) {

        airspeed.draw(canvas, new PointF(getWidth() * 0.15f, getHeight() * 0.5f), speedV);

        altitude.draw(canvas, new PointF(getWidth() * 0.85f, getHeight() * 0.5f), altitudeV);
```

```
        //roll.draw(canvas, new PointF(getWidth() * 0.5f, getHeight() * 0.8f), rollV);

        //heading.draw(canvas, new PointF(getWidth() * 0.5f, 0), headingV);

        pitch.draw(canvas, new PointF(getWidth() * 0.5f, getHeight() * 0.425f), rollV, headV,
pitchV, flightPathV);

        aoa.draw(canvas, new PointF(getWidth() * 0.15f, getHeight() * 0.85f), angleOfAttackV);
    }

    public void resume() {

        canDraw = true;
        thread = new Thread(this);
        thread.start();
    }

    public void pause() {

        canDraw = false;

        while(thread != null) {
            try {
                thread.join();
                thread = null;
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
package erau.efrc.getOrientation.Gauges.Linear;

import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.Point;
import android.graphics.PointF;
import android.graphics.Rect;

import erau.efrc.getOrientation.Gauges.Directions;
import erau.efrc.getOrientation.Gauges.Gauge;

abstract class LinearGauge implements Gauge {

    float UNITS_PER_GRADUATION = 1;
    float OVERALL_VALUE = 10;                               // Overall value the gauge can
display in one frame
    float MIN_VAL = 0;                                      // Minimum value shown on the
gauge
    float MAX_VAL = 99999;                                  // Maximum value shown on the
gauge
    float LARGER_MARGIN_VAL = 5;                            // Larger graduation value

    float GAUGE_HEIGHT = 200;                               // Overall gauge height (px)
    float LARGER_MARGIN_LEN = 20;                           // Larger graduation length,
(px)
    float SMALLER_MARGIN_LEN = 10;                          // Smaller graduation length,
(px)
```

```java
    Directions GRADUATIONS_DIRECTION = Directions.LEFT;          // Graduations direction
    String MAX_TEXT = "XXX";                                     // Maximum number of digits the
gauge can represent
    int STROKE_WIDTH = 1;
    int TEXT_SIZE = 20;

    private final Point LADDER_DIR;                   // Convert Ladder Direction (left, right, up,
down) to a unit vector
    private final Point GRAD_DIR;                     // Convert Graduations Direction (left,
right, up, down) to a unit vector
    private final float UNITS_PER_PIXEL;
    private final float MARGIN_SPACING;               // Spacing between any two adjacent
graduations on the gauge (px)
    private final Paint pathPaint;
    private final Paint textPaint;
    private final Paint centerPaint;
    private final float textHeight;
    private final float textWidth;

    LinearGauge() {

        // Define all the characteristics of the derived classes
        // NOTE: Ensure this is done right at the beginning of the constructor method, to
        // get a correct customization of the gauge
        defineGaugeChars();

        /* Initialize paint brushes */
        pathPaint = new Paint();
        pathPaint.setColor(Color.GREEN);
        pathPaint.setStrokeWidth(STROKE_WIDTH);
        pathPaint.setStyle(Paint.Style.STROKE);

        textPaint = new Paint();
        textPaint.setColor(Color.GREEN);
        textPaint.setStyle(Paint.Style.FILL);
        textPaint.setTextSize(TEXT_SIZE);
        textPaint.setTextAlign(Paint.Align.CENTER);

        // Create a new paintbrush for filling center rectangles
        centerPaint = new Paint();
        centerPaint.setColor(Color.BLACK);
        centerPaint.setStyle(Paint.Style.FILL);

        // Get height of a generic text
        Rect rect = new Rect();
        textPaint.getTextBounds("0123456789", 0, 10, rect);
        textHeight = rect.height();

        // Assume the value never goes higher than MAX_TEXT
        textWidth = textPaint.measureText(MAX_TEXT);

        // Initialize all the dependent variables
        LADDER_DIR = getVector(getLADDER_DIRECTION(GRADUATIONS_DIRECTION));
        GRAD_DIR = getVector(GRADUATIONS_DIRECTION);
        UNITS_PER_PIXEL = OVERALL_VALUE / GAUGE_HEIGHT;
        MARGIN_SPACING = GAUGE_HEIGHT / (OVERALL_VALUE/UNITS_PER_GRADUATION);
    }

    // Must be implemented by the derived class to customize the gauge
    abstract void defineGaugeChars();

    public void draw(Canvas canvas, PointF drawLocation, float... currVals) {

        // Get the first value
        float currVal = currVals[0];
```

```
// Estimate the number of units to the nearest valid value (HIGHER than or equal to the
current value)
// NOTE: A valid value is one that can be represented on a graduation
float unitsAway = currVal % UNITS_PER_GRADUATION == 0 ? 0 : UNITS_PER_GRADUATION - currVal
% UNITS_PER_GRADUATION;

// Estimate the nearest valid value (HIGHER than or equal to the current value)
float tempVal = currVal + unitsAway;

// Estimate the number of pixels to the nearest valid value, e.g. 3 units is 3 units /
UNITS_PER_PIXEL away
float pixelsAway = unitsAway / UNITS_PER_PIXEL;

// Estimate location of the nearest valid value
PointF i = new PointF();
i.x = drawLocation.x - LADDER_DIR.x * pixelsAway;
i.y = drawLocation.y - LADDER_DIR.y * pixelsAway;

// Calculate length of the gauge sketched so far
float gaugeLen = calcDistance(i, drawLocation);

Path path = new Path();

// Draw the first portion of the gauge
while (gaugeLen <= GAUGE_HEIGHT / 2 && tempVal <= MAX_VAL)
{
    // Draw the larger margin, if the current value is a multiple of LARGER_MARGIN_VAL
    drawMargins(canvas, path, i, tempVal);

    tempVal = updateCounter(i, tempVal, -1);

    gaugeLen = calcDistance(i, drawLocation);
}

// Estimate the nearest valid value (LOWER than or equal to the current value)
unitsAway = currVal % UNITS_PER_GRADUATION == 0 ? 0 : currVal % UNITS_PER_GRADUATION;

pixelsAway = unitsAway / UNITS_PER_PIXEL;

tempVal = currVal - unitsAway;

// Reset the value of i
i.x = drawLocation.x + LADDER_DIR.x * pixelsAway;
i.y = drawLocation.y + LADDER_DIR.y * pixelsAway;

// Ensure the center graduation is NOT drawn twice
if(pixelsAway == 0) {

    tempVal = updateCounter(i, tempVal, +1);
}

gaugeLen = calcDistance(i, drawLocation);

// Draw the second portion of the gauge, ensure all values are greater than the MIN_VAL
while (gaugeLen <= GAUGE_HEIGHT / 2 && tempVal >= MIN_VAL) {
    // Draw larger/smaller margins
    drawMargins(canvas, path, i, tempVal);

    // Update the counter
    tempVal = updateCounter(i, tempVal, +1);

    // Calculate length of the gauge sketched so far
    gaugeLen = calcDistance(i, drawLocation);
}
```

```
        canvas.drawPath(path, pathPaint);

        // Draw the current value bordered with a rectangle
        drawCenterRect(canvas, drawLocation);

        // Draw a new line to properly place the center text
        placeText(canvas, drawLocation, Integer.toString((int) currVal));
    }

    // Draw larger/smaller margins based on the current value
    private void drawMargins(Canvas canvas, Path path, PointF i, float tempVal) {

        // Prevent any loss in precision upto 3 decimal places
        tempVal = Math.round(tempVal * 1000)/1000;

        // Draw larger margin, if the current value is a multiple of LARGER_MARGIN_VAL
        if (tempVal % LARGER_MARGIN_VAL == 0)
        {
            path.moveTo(i.x, i.y);
            path.lineTo(i.x + GRAD_DIR.x * LARGER_MARGIN_LEN, i.y + GRAD_DIR.y *
LARGER_MARGIN_LEN);

            // Draw a new line to properly place the text
            placeText(canvas, i, Integer.toString((int) tempVal));
        }
        // Draw the smaller margin, for all other cases
        else
        {
            path.moveTo(i.x, i.y);
            path.lineTo(i.x + GRAD_DIR.x * SMALLER_MARGIN_LEN, i.y + GRAD_DIR.y *
SMALLER_MARGIN_LEN);
        }
    }

    // Place text at the specified location
    void placeText(Canvas canvas, PointF i, String str) {

        Path textPath = new Path();

        switch(GRADUATIONS_DIRECTION) {

            case LEFT:
                textPath.moveTo(i.x + GRAD_DIR.x * (LARGER_MARGIN_LEN + textWidth),
                        i.y + LADDER_DIR.y * textHeight / 2);

                textPath.lineTo(i.x + GRAD_DIR.x * LARGER_MARGIN_LEN,
                        i.y + LADDER_DIR.y * textHeight / 2);
                break;

            case RIGHT:
                textPath.moveTo(i.x + GRAD_DIR.x * LARGER_MARGIN_LEN,
                        i.y + LADDER_DIR.y * textHeight / 2);

                textPath.lineTo(i.x + GRAD_DIR.x * (LARGER_MARGIN_LEN + textWidth),
                        i.y + LADDER_DIR.y * textHeight / 2);

            case DOWN:
                textPath.moveTo(i.x + LADDER_DIR.x * textWidth / 2,
                        i.y + GRAD_DIR.y * (LARGER_MARGIN_LEN + textHeight));

                textPath.lineTo(i.x - LADDER_DIR.x * textWidth / 2 ,
                        i.y + GRAD_DIR.y * (LARGER_MARGIN_LEN + textHeight));
                break;
```

```
            default:
                return;
        }

    canvas.drawTextOnPath(str, textPath, 0, 0, textPaint);
}

// Draw rectangle at the specified location, fill is ALWAYS black_fill_paintbrush
private void drawCenterRect(Canvas canvas, PointF i) {

    float left = 0, top = 0, right = 0, bottom = 0;

    switch(GRADUATIONS_DIRECTION) {

        case LEFT:
        case RIGHT:
            left = i.x + GRAD_DIR.x * LARGER_MARGIN_LEN;
            top = i.y - LADDER_DIR.y * textHeight/2;
            right = i.x + GRAD_DIR.x * LARGER_MARGIN_LEN + GRAD_DIR.x * textWidth;
            bottom = i.y + LADDER_DIR.y * textHeight/2;
            break;

        case DOWN:
            left = i.x - LADDER_DIR.x * textWidth / 2;
            top = i.y + GRAD_DIR.y * LARGER_MARGIN_LEN;
            right = i.x + LADDER_DIR.x * textWidth / 2;
            bottom = i.y + GRAD_DIR.y * LARGER_MARGIN_LEN + GRAD_DIR.y * textHeight;
            break;

        default:
            return;
    }

    canvas.drawRect(left, top, right, bottom, centerPaint);
    canvas.drawRect(left, top, right, bottom, pathPaint);
}

// Update the counter location variable and return the current value
private float updateCounter(PointF i, float tempVal, int sign) {

    i.x = i.x + sign * LADDER_DIR.x * MARGIN_SPACING;
    i.y = i.y + sign * LADDER_DIR.y * MARGIN_SPACING;
    return(tempVal - sign * (MARGIN_SPACING * UNITS_PER_PIXEL));
}

// Calculate distance between two points using distance formula
private float calcDistance(PointF start, PointF end) {
    return (float) Math.hypot(start.x - end.x, start.y - end.y);
}

// Get LADDER_DIRECTION based on the Graduations direction
private Directions getLADDER_DIRECTION(Directions gradDirection) {
    switch (gradDirection) {
        case LEFT:
        case RIGHT:
        case HORIZONTAL:
            return Directions.DOWN;

        case DOWN:
            return Directions.LEFT;

        default:
            return Directions.DOWN;
    }
}
```

```
        // Convert direction (left, right, up, down) to a unit vector
        private Point getVector(Directions direction) {

            switch (direction) {
                case LEFT:
                case HORIZONTAL:
                    return new Point(-1, 0);

                case RIGHT:
                    return new Point(1, 0);

                case DOWN:
                    return new Point(0, 1);

                default:
                    return new Point();
            }
        }
}
```

```
package erau.efrc.getOrientation.Gauges.Linear;

import erau.efrc.getOrientation.Gauges.Directions;

public class Heading extends LinearGauge {

    public Heading() { super(); }

    void defineGaugeChars() {

        float PIXELS_PER_DEGREE = 960 / 23; // pixels / field_of_view -> horizontal

        GRADUATIONS_DIRECTION = Directions.DOWN;
        OVERALL_VALUE = 15;
        GAUGE_HEIGHT = OVERALL_VALUE * PIXELS_PER_DEGREE;
    }
}
```

```java
package erau.efrc.getOrientation.Gauges.Linear;

import erau.efrc.getOrientation.Gauges.Directions;

public class Altitude extends LinearGauge {

    public Altitude() { super(); }

    protected  void defineGaugeChars() {

        UNITS_PER_GRADUATION = 100;
        OVERALL_VALUE = 1000;
        MIN_VAL = -500;
        MAX_VAL = 10000;
        LARGER_MARGIN_VAL = 500;

        GAUGE_HEIGHT = 200;
        LARGER_MARGIN_LEN = 20;
        SMALLER_MARGIN_LEN = 10;

        GRADUATIONS_DIRECTION = Directions.LEFT;

        MAX_TEXT = "XXXXX";
        STROKE_WIDTH = 1;
        TEXT_SIZE = 20;
    }
}
```

```java
package erau.efrc.getOrientation.Gauges.Linear;

import erau.efrc.getOrientation.Gauges.Directions;

public class Airspeed extends LinearGauge {

    public Airspeed() { super(); }

    protected void defineGaugeChars() {

        UNITS_PER_GRADUATION = 1;
        OVERALL_VALUE = 10;
        MIN_VAL = 0;
        MAX_VAL = 200;
        LARGER_MARGIN_VAL = 5;

        GAUGE_HEIGHT = 200;
        LARGER_MARGIN_LEN = 20;
        SMALLER_MARGIN_LEN = 10;

        GRADUATIONS_DIRECTION = Directions.RIGHT;

        MAX_TEXT = "XXX";
        STROKE_WIDTH = 1;
        TEXT_SIZE = 20;
    }
}
```

```java
package erau.efrc.getOrientation.Gauges.FlightPathMarker;

import erau.efrc.getOrientation.Gauges.AngleOfAttack.AngleOfAttack;
import erau.efrc.getOrientation.Gauges.AngleOfAttack.ApproachAoA;
import erau.efrc.getOrientation.Gauges.AngleOfAttack.CruiseAoA;
import erau.efrc.getOrientation.Gauges.AngleOfAttack.NormalAoA;
```

```
import erau.efrc.getOrientation.Gauges.Gauge;
import erau.efrc.getOrientation.MainActivity;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.PointF;

public class FlightPathMarkerGauge implements Gauge {

    float UNITS_PER_GRADUATION = 3;
    float OVERALL_VALUE = 8.f;                              // Overall value the gauge can
display in one frame

    float GAUGE_HEIGHT;                                    // Overall gauge height (px)
    float HORIZON_LEN = 200;                               // Horizon length (px)
    float CENTER_GAP = 20;                                 // Gap in horizon line (px)
    float AOA_GAP = 15;                                    // Gap between flight-path
marker and AoA gauge
    float FPM_LINE_LEN = 25;
    float FPM_CIRCLE_RAD = 5;

    int STROKE_WIDTH = 1;

    private PointF GRAD_DIR = new PointF();
    private PointF LADDER_DIR = new PointF();

    private AngleOfAttack aoa;

    private final float UNITS_PER_PIXEL;
    private final float MARGIN_SPACING;                    // Spacing between any two
adjacent graduations on the gauge (px)
    private final Paint pathPaint;

    public FlightPathMarkerGauge(Context context) {

        /* Initialize paint brushes */
        pathPaint = new Paint();
        pathPaint.setColor(Color.GREEN);
        pathPaint.setStrokeWidth(STROKE_WIDTH);
        pathPaint.setStyle(Paint.Style.STROKE);

        switch (MainActivity.mode) {
            case takeoff:
                aoa = new NormalAoA(context);
                break;

            case cruise:
                aoa = new CruiseAoA(context);
                break;

            case approach:
                aoa = new ApproachAoA(context);
                break;
        }

        float PIXELS_PER_DEGREE = 540 / (23 * 9 /16); // pixels / field_of_view -> vertical; fov:
23 deg

        GAUGE_HEIGHT = OVERALL_VALUE * PIXELS_PER_DEGREE;

        // Initialize all the dependent variables
        UNITS_PER_PIXEL = OVERALL_VALUE / GAUGE_HEIGHT;
```

```
        // MARGIN_SPACING must be an integer, else, there will be round-off error in the final
calculations
        MARGIN_SPACING = GAUGE_HEIGHT / (OVERALL_VALUE/UNITS_PER_GRADUATION);
    }

    // Need currVals to be an array of 4 elements: theta, center-value, flight path angle, angle
of attack
    public void draw(Canvas canvas, PointF drawLocation, float... currVals) {

        float theta = currVals[0];

        // Estimate GRAD_DIR and LADDER_DIR
        GRAD_DIR = new PointF((float) Math.cos(Math.toRadians(theta)),
                (float) Math.sin(Math.toRadians(theta)));

        LADDER_DIR = new PointF((float) Math.cos(Math.toRadians(theta + 90)),
                (float) Math.sin(Math.toRadians(theta + 90)));

        float centerVal = currVals[1];

        // Estimate the number of units to the nearest valid value (HIGHER than or equal to the
current value)
        // NOTE: A valid value is one that can be represented on a graduation
        float unitsAway = centerVal % UNITS_PER_GRADUATION == 0 ? 0 : UNITS_PER_GRADUATION -
centerVal % UNITS_PER_GRADUATION;

        // Estimate the nearest valid value (HIGHER than or equal to the current value)
        float tempVal = centerVal + unitsAway;

        // Estimate the number of pixels to the nearest valid value, e.g. 3 units is 3 units /
UNITS_PER_PIXEL away
        float pixelsAway = unitsAway / UNITS_PER_PIXEL;

        // Estimate location of the nearest valid value
        PointF i = new PointF();
        i.x = drawLocation.x - LADDER_DIR.x * pixelsAway;
        i.y = drawLocation.y - LADDER_DIR.y * pixelsAway;

        Path path = new Path();

        // Calculate length of the gauge sketched so far
        float gaugeLen = calcDistance(i, drawLocation);

        boolean horizonDrawn = false;

        // Draw the first portion of the gauge
        while (gaugeLen <= GAUGE_HEIGHT / 2 && !horizonDrawn)
        {
            if(tempVal == 0) {

                // Draw the larger margin, if the current value is a multiple of LARGER_MARGIN_VAL
                drawMargins(path, i, tempVal);
                horizonDrawn = true;
            }

            tempVal = updateCounter(i, tempVal, -1);
            gaugeLen = calcDistance(i, drawLocation);
        }

        // Estimate the nearest valid value (LOWER than or equal to the current value)
        unitsAway = centerVal % UNITS_PER_GRADUATION == 0 ? 0 : centerVal % UNITS_PER_GRADUATION;

        pixelsAway = unitsAway / UNITS_PER_PIXEL;
```

```
        tempVal = centerVal - unitsAway;

        // Reset the value of i
        i.x = drawLocation.x + LADDER_DIR.x * pixelsAway;
        i.y = drawLocation.y + LADDER_DIR.y * pixelsAway;

        gaugeLen = calcDistance(i, drawLocation);

        // Draw the second portion of the gauge
        while (gaugeLen <= GAUGE_HEIGHT / 2 && !horizonDrawn) {
            if(tempVal == 0) {

                // Draw the larger margin, if the current value is a multiple of LARGER_MARGIN_VAL
                drawMargins(path, i, tempVal);
                horizonDrawn = true;
            }

            // Update the counter
            tempVal = updateCounter(i, tempVal, +1);

            // Calculate length of the gauge sketched so far
            gaugeLen = calcDistance(i, drawLocation);
        }

        // Draw flight-path marker at the appropriate location
        float flightPath = currVals[2];
        pixelsAway = (flightPath - centerVal) / UNITS_PER_PIXEL;

        if(Math.abs(pixelsAway) <= GAUGE_HEIGHT / 2) {

            drawFlightPath(path, new PointF(drawLocation.x - LADDER_DIR.x * pixelsAway,
                    drawLocation.y - LADDER_DIR.y * pixelsAway));

            PointF aoaLocation = new PointF(drawLocation.x - LADDER_DIR.x * pixelsAway -
GRAD_DIR.x * AOA_GAP,
                    drawLocation.y - LADDER_DIR.y * pixelsAway - GRAD_DIR.y * AOA_GAP);

            // Update theta for aoa; and, draw aoa
            float aoaV = currVals[3];
            aoa.setTheta(90 + theta, theta);
            aoa.draw(canvas, aoaLocation, aoaV);
        }

        canvas.drawPath(path, pathPaint);
    }

    private void drawFlightPath(Path path, PointF i) {

        // In positive GRAD direction
        path.moveTo(i.x + GRAD_DIR.x * FPM_CIRCLE_RAD,
                i.y + GRAD_DIR.y * FPM_CIRCLE_RAD);
        path.lineTo(i.x + GRAD_DIR.x * (FPM_CIRCLE_RAD + FPM_LINE_LEN),
                i.y + GRAD_DIR.y * (FPM_CIRCLE_RAD + FPM_LINE_LEN));

        // In negative GRAD direction
        path.moveTo(i.x - GRAD_DIR.x * FPM_CIRCLE_RAD,
                i.y - GRAD_DIR.y * FPM_CIRCLE_RAD);
        path.lineTo(i.x - GRAD_DIR.x * (FPM_CIRCLE_RAD + FPM_LINE_LEN),
                i.y - GRAD_DIR.y * (FPM_CIRCLE_RAD + FPM_LINE_LEN));

        // In negative LADDER direction
        path.moveTo(i.x - LADDER_DIR.x * FPM_CIRCLE_RAD,
                i.y - LADDER_DIR.y * FPM_CIRCLE_RAD);
        path.lineTo(i.x - LADDER_DIR.x * (FPM_CIRCLE_RAD + FPM_LINE_LEN),
                i.y - LADDER_DIR.y * (FPM_CIRCLE_RAD + FPM_LINE_LEN));
```

```java
        // Center circle
        path.addCircle(i.x, i.y, FPM_CIRCLE_RAD, Path.Direction.CCW);
    }

    private void drawMargins(Path path, PointF i, float tempVal) {

        // Prevent any loss in precision upto 3 decimal places
        tempVal = Math.round(tempVal * 1000)/1000;

        // Draw horizon line, if the current value is 0
        if(tempVal == 0) {
            // Draw on positive side
            path.moveTo(i.x + GRAD_DIR.x * CENTER_GAP, i.y + GRAD_DIR.y * CENTER_GAP);
            path.lineTo(i.x + GRAD_DIR.x * (CENTER_GAP + HORIZON_LEN),
                    i.y + GRAD_DIR.y * (CENTER_GAP + HORIZON_LEN));

            // Draw on negative side
            path.moveTo(i.x - GRAD_DIR.x * CENTER_GAP, i.y - GRAD_DIR.y * CENTER_GAP);
            path.lineTo(i.x - GRAD_DIR.x * (CENTER_GAP + HORIZON_LEN),
                    i.y - GRAD_DIR.y * (CENTER_GAP + HORIZON_LEN));
        }
    }

    // Update the counter location variable and return the current value
    private float updateCounter(PointF i, float tempVal, int sign) {

        i.x = i.x + sign * LADDER_DIR.x * MARGIN_SPACING;
        i.y = i.y + sign * LADDER_DIR.y * MARGIN_SPACING;

        return(tempVal - sign * (MARGIN_SPACING * UNITS_PER_PIXEL));
    }

    // Calculate distance between two points using distance formula
    private float calcDistance(PointF start, PointF end) {
        return (float) Math.hypot(start.x - end.x, start.y - end.y);
    }
}
```

```java
package erau.efrc.getOrientation.Gauges.Curvelinear;

import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.PointF;

import erau.efrc.getOrientation.Gauges.Directions;
import erau.efrc.getOrientation.Gauges.Gauge;

import static java.lang.Math.abs;

abstract class CurvelinearGauge implements Gauge {

    float UNITS_PER_GRADUATION = 5;
    float LARGE_MARGIN_VAL = 10;
    float MIN_VAL = 0;                              // Absolute units
    float MAX_VAL = 30;                             // Absolute units

    Directions GAUGE_DIR = Directions.DOWN;         // Gauge direction

    float GAUGE_WIDTH = 150;
    float LARGE_MARGIN_LENGTH = 20;                 // Larger graduation length, (px)
```

```
    float SMALL_MARGIN_LENGTH = 10;                        // Smaller graduation length, (px)
    float TRIANGLE_HEIGHT = 5;                       // Triangle height, to indicate current
value (px)
    float TRIANGLE_THETA = 6;                         // Triangle apex angle, to indicate
current values (px)
    int STROKE_WIDTH = 1;

    private final float[] THETA_LIMITS;

    private final Paint pathPaint;

    CurvelinearGauge() {

        // Ensure this is the first statement always
        defineChars();

        /* Initialize paint brushes */
        pathPaint = new Paint();
        pathPaint.setColor(Color.GREEN);
        pathPaint.setStrokeWidth(STROKE_WIDTH);
        pathPaint.setStyle(Paint.Style.STROKE);

        THETA_LIMITS = getMIN_MAX_VAL(GAUGE_DIR);
    }

    protected abstract void defineChars();

    public void draw(Canvas canvas, PointF drawLocation, float... currVals) {

        // Get the first value
        float currVal = currVals[0];

        // arc_length = pi * r
        float gaugeR = GAUGE_WIDTH / 2;
        float largeMarginR = gaugeR - LARGE_MARGIN_LENGTH;
        float smallMarginR = gaugeR - SMALL_MARGIN_LENGTH;

        // Keep track of the angle being drawn
        float positiveTheta = 0;
        float negativeTheta = 0;

        Path path = new Path();

        // sketchR = largeMarginR or smallMarginR, depending on the currnt angle
        float sketchR;

        // Define the line constants: a and b (y = ax + b)
        float a = (THETA_LIMITS[1] - THETA_LIMITS[0]) / (MAX_VAL - MIN_VAL);
        float b = THETA_LIMITS[0] - a * MIN_VAL;

        // Draw the entire gauge
        while(positiveTheta <= MAX_VAL) {

            if(positiveTheta % LARGE_MARGIN_VAL == 0) {
                sketchR = largeMarginR;
            }
            else {
                sketchR = smallMarginR;
            }

            PointF positiveVector = new PointF((float) Math.cos(Math.toRadians(a * positiveTheta +
b)),
                    (float) Math.sin(Math.toRadians(a * positiveTheta + b)));

            path.moveTo(drawLocation.x + positiveVector.x * sketchR,
```

```
                drawLocation.y + positiveVector.y * sketchR);

        path.lineTo(drawLocation.x + positiveVector.x * gaugeR,
                drawLocation.y + positiveVector.y * gaugeR);

        if(positiveTheta != negativeTheta) {
            PointF negativeVector = new PointF((float) Math.cos(Math.toRadians(a *
negativeTheta + b)),
                    (float) Math.sin(Math.toRadians(a * negativeTheta + b)));

            path.moveTo(drawLocation.x + negativeVector.x * sketchR,
                    drawLocation.y + negativeVector.y * sketchR);

            path.lineTo(drawLocation.x + negativeVector.x * gaugeR,
                    drawLocation.y + negativeVector.y * gaugeR);
        }

        positiveTheta += UNITS_PER_GRADUATION;
        negativeTheta -= UNITS_PER_GRADUATION;
    }

    // Draw the triangular pointer, iff the current roll angle is legally bounded
    if(abs(currVal) <= MAX_VAL) {

        // Scale the current value to the gauge coordinate system
        float theta = a * currVal + b;

        // Calculate the unit vector along the current value
        PointF unitVector = new PointF((float) Math.cos(Math.toRadians(theta)),
                (float) Math.sin(Math.toRadians(theta)));

        PointF apex = new PointF(drawLocation.x + unitVector.x * gaugeR, drawLocation.y +
unitVector.y * gaugeR);

        // Get the angular value at the left vertex of the triangle
        theta -= TRIANGLE_THETA / 2;

        unitVector = new PointF((float) Math.cos(Math.toRadians(theta)),
                (float) Math.sin(Math.toRadians(theta)));

        PointF left = new PointF(drawLocation.x + unitVector.x * (gaugeR + TRIANGLE_HEIGHT),
                drawLocation.y + unitVector.y * (gaugeR + TRIANGLE_HEIGHT));

        // Get the angular value at the right vertex of the triangle
        theta += TRIANGLE_THETA;

        unitVector = new PointF((float) Math.cos(Math.toRadians(theta)),
                (float) Math.sin(Math.toRadians(theta)));

        PointF right = new PointF(drawLocation.x + unitVector.x * (gaugeR + TRIANGLE_HEIGHT),
                drawLocation.y + unitVector.y * (gaugeR + TRIANGLE_HEIGHT));

        // Draw the pointer triangle, if the current theta is bounded
        path.moveTo(apex.x, apex.y);
        path.lineTo(left.x, left.y);
        path.lineTo(right.x, right.y);
        path.lineTo(apex.x, apex.y);
    }

    canvas.drawPath(path, pathPaint);
}

private float[] getMIN_MAX_VAL(Directions direction) {

    float[] array = new float[2];
```

```java
        switch (direction) {

            // Values along the left half are positive and right half are negative
            default:
            case DOWN:
                array[0] = 90;
                array[1] = 180;
        }

        return array;
    }

}
```

---

```java
package erau.efrc.getOrientation.Gauges.Curvelinear;

public class Roll extends CurvelinearGauge {

    public Roll() { super(); }

    protected void defineChars() {
    }
}
```

---

```java
package erau.efrc.getOrientation.Gauges.AngleOfAttack;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.PointF;

import erau.efrc.getOrientation.Gauges.Gauge;
import erau.efrc.getOrientation.R;

public abstract class AngleOfAttack implements Gauge {

    float POSITIVE_STALL_END;
    float POSITIVE_STALL_START;

    float NEGATIVE_STALL_START;
    float NEGATIVE_STALL_END;

    float OVERALL_VALUE; // Overall value the gauge can display in one frame

    float POSITIVE_CAUTION;
    float NEGATIVE_CAUTION;

    float GAUGE_HEIGHT;                                  // Overall gauge height (px)
    float LARGER_MARGIN_LEN;                             // Larger graduation length, (px)

    int STROKE_WIDTH;
    int TEXT_SIZE;

    float LADDER_THETA = 90;                             // degrees
    float GRADUATION_THETA = 0;                          // degrees

    float BOLT_ANGLE = 7;
```

```
    float GLIDER_ANGLE = 10;
    float LEAF_ANGLE = 12;

    private PointF LADDER_DIRECTION;
    private PointF GRADUATION_DIRECTION;

    private final Paint regularPaint;
    private final Paint cautionPaint;
    private final Paint stallPaint;
    final Paint textPaint;

    private final float UNITS_PER_PIXEL;

    private final Bitmap bolt;
    private final Bitmap glider;
    private final Bitmap leaf;

    AngleOfAttack(Context context) {

        // Ensure all values are initialized
        defineChars();

        /* Initialize paint brushes */
        regularPaint = new Paint();
        regularPaint.setColor(Color.GREEN);
        regularPaint.setStrokeWidth(STROKE_WIDTH);
        regularPaint.setStyle(Paint.Style.STROKE);

        cautionPaint = new Paint();
        cautionPaint.setColor(Color.YELLOW);
        cautionPaint.setStrokeWidth(STROKE_WIDTH);
        cautionPaint.setStyle(Paint.Style.STROKE);

        stallPaint = new Paint();
        stallPaint.setColor(Color.RED);
        stallPaint.setStrokeWidth(STROKE_WIDTH);
        stallPaint.setStyle(Paint.Style.STROKE);

        textPaint = new Paint();
        textPaint.setColor(Color.GREEN);
        textPaint.setStyle(Paint.Style.FILL);
        textPaint.setTextSize(TEXT_SIZE);
        textPaint.setTextAlign(Paint.Align.LEFT);

        // Initialize all the dependent variables
        UNITS_PER_PIXEL = OVERALL_VALUE / GAUGE_HEIGHT;

        LADDER_DIRECTION = new PointF((float) Math.cos(Math.toRadians(LADDER_THETA)),
                (float) Math.sin(Math.toRadians(LADDER_THETA)));

        GRADUATION_DIRECTION = new PointF((float) Math.cos(Math.toRadians(GRADUATION_THETA)),
                (float) Math.sin(Math.toRadians(GRADUATION_THETA)));

        // Load the images, adjusting the target density as per the image
        BitmapFactory.Options bmpOptions = new BitmapFactory.Options();
        bmpOptions.inScaled = true;
        bmpOptions.inTargetDensity = 15;
        bolt = BitmapFactory.decodeResource(context.getResources(), R.drawable.bolt, bmpOptions);

        bmpOptions.inTargetDensity = 7;
        glider = BitmapFactory.decodeResource(context.getResources(), R.drawable.glider,
bmpOptions);

        bmpOptions.inTargetDensity = 5;
        leaf = BitmapFactory.decodeResource(context.getResources(), R.drawable.leaf, bmpOptions);
```

```java
    }

    protected abstract void defineChars();

    @Override
    public void draw(Canvas canvas, PointF currLocation, float... currVals) {

        float currVal = currVals[0];

        LADDER_DIRECTION.x = (float) Math.cos(Math.toRadians(LADDER_THETA));
        LADDER_DIRECTION.y = (float) Math.sin(Math.toRadians(LADDER_THETA));

        GRADUATION_DIRECTION.x = (float) Math.cos(Math.toRadians(GRADUATION_THETA));
        GRADUATION_DIRECTION.y = (float) Math.sin(Math.toRadians(GRADUATION_THETA));

        // Ensure the aoa value is bounded
        if(currVal > POSITIVE_STALL_END) {
            currVal = POSITIVE_STALL_END;
        }
        else if(currVal < NEGATIVE_STALL_END) {
            currVal = NEGATIVE_STALL_END;
        }

        // Create all path objects
        Path regularPath = new Path();
        Path cautionPath = new Path();
        Path stallPath = new Path();

        // Estimate the start and end positions of the regular region
        float startLen = (currVal - POSITIVE_CAUTION) * 1 / UNITS_PER_PIXEL;
        float endLen = (currVal - NEGATIVE_CAUTION) * 1 / UNITS_PER_PIXEL;
        moveTolineTo(regularPath, currLocation, startLen, endLen);

        // Estimate the start and end positions of the positive caution region
        startLen = (currVal - POSITIVE_STALL_START) * 1 / UNITS_PER_PIXEL;
        endLen = (currVal - POSITIVE_CAUTION) * 1 / UNITS_PER_PIXEL;
        moveTolineTo(cautionPath, currLocation, startLen, endLen);

        // Estimate the start and end positions of the negative caution region
        startLen = (currVal - NEGATIVE_CAUTION) * 1 / UNITS_PER_PIXEL;
        endLen = (currVal - NEGATIVE_STALL_START) * 1 / UNITS_PER_PIXEL;
        moveTolineTo(cautionPath, currLocation, startLen, endLen);

        // Estimate the start and end positions of the positive stall region
        startLen = (currVal - POSITIVE_STALL_START) * 1 / UNITS_PER_PIXEL;
        endLen = (currVal - POSITIVE_STALL_END) * 1 / UNITS_PER_PIXEL;
        moveTolineTo(stallPath, currLocation, startLen, endLen);

        // Draw the end points of the positive stall region
        float xLocation = currLocation.x + LADDER_DIRECTION.x * endLen + GRADUATION_DIRECTION.x *
LARGER_MARGIN_LEN;
        float yLocation = currLocation.y + LADDER_DIRECTION.y * endLen + GRADUATION_DIRECTION.y *
LARGER_MARGIN_LEN;

        if(xLocation > 0 && yLocation > 0) {
            stallPath.lineTo(xLocation, yLocation);
        }

        // Estimate the start and end positions of the negative stall region
        startLen = (currVal - NEGATIVE_STALL_START) * 1 / UNITS_PER_PIXEL;
        endLen = (currVal - NEGATIVE_STALL_END) * 1 / UNITS_PER_PIXEL;

        moveTolineTo(stallPath, currLocation, startLen, endLen);

        // Draw the end points of the negative stall region
```

```
        xLocation = currLocation.x + LADDER_DIRECTION.x * endLen + GRADUATION_DIRECTION.x *
LARGER_MARGIN_LEN;
        yLocation = currLocation.y + LADDER_DIRECTION.y * endLen + GRADUATION_DIRECTION.y *
LARGER_MARGIN_LEN;

        if(xLocation > 0 && yLocation > 0) {
            stallPath.lineTo(xLocation, yLocation);
        }

        canvas.drawPath(regularPath, regularPaint);
        canvas.drawPath(cautionPath, cautionPaint);
        canvas.drawPath(stallPath, stallPaint);

        // Estimate the location of the bolt - ensure the location is calculated to the center of
the bolt
        float len = (currVal - BOLT_ANGLE) * 1 / UNITS_PER_PIXEL;
        xLocation = currLocation.x + LADDER_DIRECTION.x * (len - bolt.getHeight() / 2)
                + GRADUATION_DIRECTION.x * (- bolt.getWidth() / 2);
        yLocation = currLocation.y + LADDER_DIRECTION.y * (len - bolt.getHeight() / 2)
                + GRADUATION_DIRECTION.y * (- bolt.getWidth() / 2);
        // Draw the bolt
        canvas.drawBitmap(bolt, xLocation, yLocation, null);

        // Repeat for other bitmaps
        len = (currVal - GLIDER_ANGLE) * 1 / UNITS_PER_PIXEL;
        xLocation = currLocation.x + LADDER_DIRECTION.x * (len - glider.getHeight() / 2)
                + GRADUATION_DIRECTION.x * (- glider.getWidth() / 2);
        yLocation = currLocation.y + LADDER_DIRECTION.y * (len - glider.getHeight() / 2)
                + GRADUATION_DIRECTION.y * (- glider.getWidth() / 2);
        canvas.drawBitmap(glider, xLocation, yLocation, null);

        len = (currVal - LEAF_ANGLE) * 1 / UNITS_PER_PIXEL;
        xLocation = currLocation.x + LADDER_DIRECTION.x * (len - leaf.getHeight() / 2)
                + GRADUATION_DIRECTION.x * (- leaf.getWidth() / 2);
        yLocation = currLocation.y + LADDER_DIRECTION.y * (len - leaf.getHeight() / 2)
                + GRADUATION_DIRECTION.y * (- leaf.getWidth() / 2);
        canvas.drawBitmap(leaf, xLocation, yLocation, null);
    }

    void moveTolineTo(Path path, PointF location, float startLen, float endLen) {

        float xLocation = location.x + LADDER_DIRECTION.x * startLen;

        if(xLocation < 0) {
            xLocation = 0;
        }

        float yLocation = location.y + LADDER_DIRECTION.y * startLen;

        if(yLocation < 0) {
            yLocation = 0;
        }

        path.moveTo(xLocation, yLocation);

        xLocation = location.x + LADDER_DIRECTION.x * endLen;

        if(xLocation < 0) {
            xLocation = 0;
        }

        yLocation = location.y + LADDER_DIRECTION.y * endLen;

        if(yLocation < 0) {
            yLocation = 0;
```

```
        }

        path.lineTo(xLocation, yLocation);
    }

    public void setTheta(float ladderTheta, float gradTheta) {
        LADDER_THETA = ladderTheta;
        GRADUATION_THETA = gradTheta;
    }
}
```

```
package erau.efrc.getOrientation.Gauges.AngleOfAttack;

import android.content.Context;

public class ApproachAoA extends AngleOfAttack {

    public ApproachAoA(Context context) {
        super(context);
    }

    protected void defineChars() {
        POSITIVE_STALL_END = 20;
        POSITIVE_STALL_START = 17;

        NEGATIVE_STALL_START = -12;
        NEGATIVE_STALL_END = -15;

        OVERALL_VALUE = POSITIVE_STALL_END - NEGATIVE_STALL_END;

        POSITIVE_CAUTION = 15;
        NEGATIVE_CAUTION = -10;

        GAUGE_HEIGHT = 200;
        LARGER_MARGIN_LEN = 25;

        STROKE_WIDTH = 4;
    }

}
```

```
package erau.efrc.getOrientation.Gauges.AngleOfAttack;

import android.content.Context;

public class CruiseAoA extends AngleOfAttack {

    public CruiseAoA(Context context) {
        super(context);
    }

    protected void defineChars() {
        POSITIVE_STALL_END = 15;
        POSITIVE_STALL_START = 15;

        NEGATIVE_STALL_START = 0;
        NEGATIVE_STALL_END = 0;

        OVERALL_VALUE = POSITIVE_STALL_END - NEGATIVE_STALL_END;

        POSITIVE_CAUTION = 15;
        NEGATIVE_CAUTION = 0;
```

```
            GAUGE_HEIGHT = 250;
            LARGER_MARGIN_LEN = 0;

            STROKE_WIDTH = 3;
        }
}

package erau.efrc.getOrientation.Gauges.AngleOfAttack;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.PointF;

public class NormalAoA extends AngleOfAttack {

    public NormalAoA(Context context) {
        super(context);
    }

    protected void defineChars() {

        TEXT_SIZE = 20;
    }

    @Override
    public void draw(Canvas canvas, PointF drawLocation, float... currVals) {

        canvas.drawText(Float.toString(currVals[0]), drawLocation.x, drawLocation.y, textPaint);
    }
}
```