Appendix B: Descriptions of Virtual Instruments (vis) Implemented

Overview of vis Implemented

This appendix contains a brief description of each vi implemented in this project. Labview implements functions as virtual instruments, vis. Each vi was created and saved with the appropriate filename. Necessary input and output nodes were then added to the vis. Labview's constants were used whenever possible, especially for the value of pi. The schematics for all the vis described in this appendix can be found in Appendix C. The inputs and outputs of each function contain the measurement unit in parentheses. Note that the notation NM represents International Nautical Miles.

RadToDeg.vi

This function converts the input, Radians, from radians to degrees using Labview constant definition for PI. Note that the returned value, Degrees, is not necessarily 0 < Degrees < 360. The function was created this way in order to promote function reuse.

AddRadians.vi

This function adds two numbers, Number 1 (rads) and Number 2 (rads), in radians. The function then calls RadMod.vi to guarantee that result is 0 <= result < 2PI. The result is returned in radians as Result (rads).

CourseBetTwoPts.vi

This vi requires the input latitude and longitude to follow established sign conventions. It is also mandatory that the Magnetic Variation follow the same established sign conventions. The vi begins by calling RadBetTwoPts.vi with Position 1 Latitude (rads), Position 1 Longitude (rads), Position 2 Latitude (rads), and Position 2 Longitude (rads) to determine the radian difference of the two positions. Formulas from Aviation Formulary are then utilized to determine the True Course. For accuracy, two different calculations are required - one for paths starting at the poles and one for all other starting points. Once the course, the True Course, is calculated, Magnetic Variation is used to determine Magnetic Course. The vi then calls SubRadians.vi to ensure result is >=0 and < 2PI. The result is returned as Course (rads).

The following algorithm from *Aviation Formulary v1.22* is used to calculate the true course (tc):

```
IF sin(lon2-lon1)<0
        tc1=acos((sin(lat2)-sin(lat1)*cos(d))/(sin(d)*cos(lat1)))

ELSE
        tc1=2*pi-acos((sin(lat2)-sin(lat1)*cos(d))/(sin(d)*cos(lat1)))

ENDIF
```
[1]

where (lat1,long1) are the Geodetic coordinates of the first position, (lat2, long2) are the Geodetic coordinates of the second position, and d is the radial distance between the two points calculated by RadsBetTwoPts.vi.

DegToRad.vi

This vi converts the input, Degrees, from degrees to radians. There is no limit set for the output of Radians. The formula used is:

```
rad=deg/180 * PI.
```

DegToDecimalDeg.vi

This function converts a coordinate specified in Degrees, Minutes, and Seconds into a decimal value, DegsInDecimal, using the following formula:

```
DegDec = Deg + Min/60 + Sec/3600
```

NMEAToPos.vi

The vi begins by searching StringRead for the NMEA sentence code $GPGLL. Once the code is found, the vi strips the entire line. The various position data are then extracted from the line. Using offsets, subsets of the line are converted to decimal numbers with fixed widths to maximize accuracy.

The vi accounts for the HHMM.MM latitude data format by converting HHMM.MM to a decimal in degrees and dividing by 100 to convert the decimal number to HH.MMMM. The vi then extracts the MM.MM number from the original string. This number is converted to degrees by dividing by 60. Further, to extract the degree value from the first number, the second number is subtracted from the first. masking high __XX to determine the degrees and then calculating the minutes using rest of string. Note that minutes are in decimal format. The minutes are then divided by 60 to get a fraction in degrees, which is

added to the degrees. The latitude is then multiplied by 1 or -1 to compensate for North or South respectively. The longitude is then multiplied by 1 or -1 to compensate for West or East respectively.

The following formula, which is described in Bennett's article, is implemented in the function:

Longitude = (±1) * (DDDMM.MM/100 - MM.MM/100) + MM.MM
Latitude = (±1) * (DDMM.MM/100 - MM.MM/100) + MM.MM[2]

NMBetTwoPts.vi

RadsBetTwoPts.vi is first called to determine the radians between two pts, specified by Positon 1 Latitude (rads), Position 1 Longitude (rads), Position 2 Latitude (rads), and Position 2 Longitude (rads). The radians are then converted to degrees. The degree measurement is then multiplied by 60 NM per degree to get the result, Distance (NM), in NM.

NMEAToAltitude.vi

Begins by searching StringRead with Match Pattern.vi for the NMEA sentence code $PGRMZ, which is for. Once code is altitude. Once the string is found, the string after the code is stripped and first number is converted to altitude. This number is returned as Altitude (ft).

From Bennett's article on NMEA, the NMEA-0183 format for reading the altitude is:

$PGRMZ,xxxx,f,y*21[3]

where xxx,f is the altitude in feet and y represents the position fix dimensions (2 = user altitude, 3 = GPS altitude). Note that the altitude returned is in feet.

InputFileIO.vi

The vi begins by calling Open/Create/Replace File.vi to open the file specified by Filename. Once file is opened, ReadFile.vi is called to read all characters in file until the end of file, EOF, is detected. The string read from the file is returned as Data String.

NMToRads.vi

This vi converts Distance (NM), a distance in nautical miles, to Distance (rads), a distance in radians, using the fact that 1 deg = 60 NM.[4] This is the radial distance in NM converted to radians based on the earth's radius.

GetStationIndex.vi

The vi first calls SearchDataBase.vi to determine the record number of the last occurrence of Station Frequency in the string specified by File Data. A list of all stations with that frequency is generated and presented to the user in StationsAvailable and in the pull down list StationLd. Each station in the list corresponds to an enumerated value. The stations in the list are enumerated from the last occurrence in the data string. Therefore, the higher the enumerated value, the further back in the data file the record is to be found. The Record Index is calculated by subtracting the enumerated value from the last occurrence. The vi checks for an invalid station choice and defaults to 0, causing the last occurrence of the station to be returned in Record Index if an invalid station choice is selected.

OutputFileIO.vi

This vi begins by calling Open/Create/Replace File.vi to open the file specified in Filename. Write File.vi is then called to write the contents of Data String to the file. Note that Write File.vi is called with the options to convert the End of Line Characters and the tabs to spaces. Once the string has been written to the file, Close File.vi is called to close the file.

SimPosition.vi

The vi begins by finding the True Heading by adding the Heading (rads) and the Magnetic Variance (rads). The next task is to convert the Distance Flown (NM)/Cycle to radians using NMToRads.vi. The converted distance, True Heading (rads), Initial Latitude (rads), and Initial Longitude (rads) are used by DistAlongRadial.vi to generate the Final Latitude (rads) and Final Longitude (rads).

SimFlight.vi

This vi begins by calling SimAcceleration.vi to calculate the Final Speed (kts) using the Throttle level and the Initial Speed (kts). The Final Speed is then used to calculate the Distance Flown (NM)/Cycle. The vi then calls AddRadians.vi to add the Initial Heading (rads) Bank Angle (rads) to find the Final Heading (rads). The Final Heading (rads), Initial Longitude (rads), Initial Latitude (rads), Distance Flown (NM)/Cycle, and

Magnetic Variation (rads) are used as parameters for calling SimPosition.vi to calculate the Final Latitude (rads) and Final Longitude (rads). SimAltitude.vi is also called with the Initial Altitude (ft), Elevator (deg), and Distance Flown (NM)/Cylce in order to calculate the Final Altitude. The three vital outputs of this vi are Final Latitidue (rads), Final Longitude (rads), and Final Altitude since all the instruments rely on these calculations.

SimAltitude.vi

This vi first converts the Distance Flown (NM)/Cycle to Distance Flown (ft)/Cycle. This value multiplied by the sine of the Elevator (deg) to determine the Altitude Change (ft)/Cycle. Since this value would result in the aircraft ascending at a high ascent rate for a small elevator angle, the value is divided by 10 to yield a more realistic effect. The Altitude Change is then added to the Initial Altitude to generate the Final Altitude. The Altitude Change is dependent on the aircraft's airspeed and on the sine of the Elevator angle.

SimAcceleration.vi

This vi multiplies the throttle by 5 kts to determine the maximum airspeed for that throttle level. Since the aircraft acceleration is fixed at -5 kts/cycle, 0, or 5 kts/cycle, comparisons are then conducted to determine if 5 knots should be added or subtracted from the Init Speed (kts) in order to simulate Speed (kts). The maximum speed is limited to 500 kts, which should be sufficient for this project.

SearchDatabase.vi

This vi uses a while loop structure to linearly search a database. The vi searches the Station Data string for the Desired Number, which represents the frequency. If an exact match is found in the Station Data string, the following line is extracted from the Station Data string and sent to a case statement. The case statement adds the Station Name to the global array Station Names. The Number of options register is also incremented. Finally, Valid Station register is set to true to indicate that a valid station was found. Once all occurrences of the Desired Number are found in the Station Data string or the end of the string is detected, the while loop exits. The Valid Station register is then used to set the File Index (end of options) to the Number of Options for a valid station or -1 for an invalid station. The array of Station Names is also returned to the calling vi.

SaveDebugData.vi

This vi was created for saving data only for debugging purposes.

ReadGPSData.vi

This vi begins by calling SerialPortRead.vi to read a set number of bytes, Number Bytes To Read, from the port is specified by Com Port. SerialPortRead.vi returns a data string to the calling function. This data string is then sent to NMEAToAltitude.vi and NMEAToPosition.vi to calculate the GPS Latitude (rads), GPS Longitude (rads), GPS Time, and GPS Altitude (ft).

RadToDeg.vi

This vi converts a value, Radians, from radians to Degrees.

RadsBetTwoPts.vi

This vi uses the formula from *Aviation Formulary* to calculate the Distance (rads) between two positions specified by Lat 1 (rads), Long 1 (rads), Lat 2 (rads), and Long 2 (rads). The formula used is:

$d = 2*\text{asin}(\text{sqrt}((\sin((\text{lat1}-\text{lat2})/2))^2 + \cos(\text{lat1})*\cos(\text{lat2})*(\sin((\text{long1}-\text{long2})/2))^2))$[5]

RadMod.vi

This vi executes one iteration of the modulus operation for the given input, Input Value (rads). The vi uses comparisons to determine if Input Value is 0 <= Input Value < 2*PI. If Input Value > 2*PI, then 2*PI is subtracted from the Input Value.

SubRadians.vi

This vi subtracts Number 2 (rads) from Number 1 (rads) and calls RadMod.vi once before outputting Result (rads). This tends to ensure that Result is 0<= Result < 2 PI

VORCalc.vi

This vi Calls NMBetTwoPts.vi, CourseBetTwoPts.vi, VORStatus.vi to calculate all necessary VOR calculations: VOR Status (to/from/off), Distance (NM), and Needle Deflection (rads). If the VOR Status is OFF, then the Distance returned is 0 NM.

WritePosData.vi

This vi converts the Latitude (degs), Longitude (degs), Altitude (ft) and Time (s) values into strings. The strings are then concatenated together into to from one large string, with spaces separating the substrings. The string is then sent to OutputFileIO.vi to be written to the specified file.

VOR.vi

This vi begins by calling GetStationIndex.vi to retrieve the index for the desired VOR station. This index value and the String Data File string structure are sent to GetStaionInfo.vi, which produces the Station ID, station latitude values, and station longitude values. These values are converted from degrees to radians. VORCalc.vi is called with the aircraft latitude/longitude and the station latitude/longitude values. VORCalc.vi then produces the Distance (NM), Course (degrees), and a Needle Deflection (degs).

VORDisplayDriver.vi

This vi calculates the degree values for 90 degrees left, 90 degrees right, and 180 degrees of the OBS Heading (deg). The OBS Heading (deg) is first converted to radians. AddRadians is called to determine East Heading (deg), which is 90 degrees to the right of OBS Heading (deg). AddRadians is then called again to determine South Heading (deg), which is 180 degrees to the right of OBS Heading (deg). Finally, Subradians is called to determine West Heading (deg), which is 90 degrees to the left of OBS Heading (deg).

VORStatus.vi

The vi calls NMBetTwoPts.vi and CourseBetTwoPts.vi to calculate the distance and course from the aircraft's position, specified by Aircraft Latitude (rads) and Aircraft Longitude (rads), and the VOR station's position, specified by VOR Latitude (rads) and VOR Longitude (rads). The distance is compared to the maximum distance of 200 NM, which will cause the VOR Status (to/from/off) to display "Off" if the VOR Override is "Off" and the distance between the aircraft and VOR station is > 200 NM. The VOR OBS (rads) and course are compared to determine if the VOR Status should indicate TO or FROM using 3 formula boxes for all possible combinations. The course and VOR OBS are also used to determine the absolute needle deflection. An additional formula box compares the course and VOR OBS to determine if the Needle Deflection (rads) should be in the negative or positive direction. One final step is to set the Needle Deflection (rads) to -10 or +10 if the VOR Status (to/from/off) is OFF.

DistAlongRadial.vi

This vi uses equations from *Aviation Formulary* to calculate the Final Lat (rads) and Final Long (rads) from Distance (rads) along the Course (rads) with initial point, Init Lat (rads) and Init Long (rads). The following algorithm from *Aviation Formulary* is implemented in this function:

```
lat=asin(sin(lat1)*cos(d)+cos(lat1)*sin(d)*cos(tc))

IF (cos(lat)=0)
        lon=lon1      // endpoint a pole
ELSE
        lon=mod(lon1-asin(sin(tc)*sin(d)/cos(lat))+pi,2*pi)-pi

ENDIF
```
[6]

GPS Project.vi

This is the main vi from which the GPS project is executed from. The program executes in a continuous loop of cycles. Since Labview is hardware oriented, registers must be implemented to preserve data values from cycle to cycle. The vi contains registers to save File Record Index, Pos Time (sec), Latitude (rads), Longitude (rads), Aircraft Heading (rads), Simulated Airspeed (kts), Simulated Heading (kts), and ADF Heading (kts). The File Record Index is used File Mode to track the current position in the data file. Pos Time (sec), Latitude (rads), and Longitude (rads) are used to preserve the respective values in the current cycle for use in the next cycle by dynamic and Simulation calculations. Aircraft Heading (rads) and ADF Heading (kts) are used solely for formatting and appearance. Simulated Airspeed (kts) and Simulated Heading (kts) are used for the Simulation

The vi is executed by the user activating Labview's RUN button. The vi begins by calling the Initialization Module. The Initialization Module sets up the serial port for communication, converts the Simulation Mode initial values to the required formats, clears the array for Track Record, reads the VOR station data, and reads the ADF station data. The vi then transfers control to the Main Module, which is implemented in a continuous loop.

VOR.vi is called with the appropriate arguments to calculate the ADF and VOR data. For the VOR instruments, VORDisplayDriver.vi is called to create the necessary outputs to enhance the VOR instrument displays. For the ADF instrument, SubRadians.vi is called to determine the bearing of the station from the aircraft's current course. An additional block of code is added to eliminate displaying invalid data for the ADF instrument.

For the position data source, two separate case blocks are created. One block causes a beep to be emitted if the program is in File Mode or Simulation Mode. The same block is used to create a one second delay in File Mode and Simulation Mode. The delay is necessary to allow the user to view the data on the instruments. The Elapsed Time Per Cycle input is used by this block to increase or decrease the delay to create an effect of speeding up or slowing down time.

Another block describes the source of the position data. For Simulation Mode, the Simulation Mode Interface Module calls SimFlight.vi to generate position data. For GPS Mode, the GPS Mode Interface Module calls NMEAToAltitude.vi and NMEToPos.vi to generate the position data. File Mode, which was implemented as an additional feature, calls the various functions required to extract the position data from the data string.

The module also calls CourseBetTwoPts.vi, using the current position and the aircraft's previous position to determine the aircraft's course. The aircraft's position is saved in registers during each cycle since it needs to be accessed in the following cycle. An additional block has also been added to eliminate displaying invalid data.

A global array is used to store the position data for each cycle. This array is linked to the x-y graph, which displays the Track Record. Some formatting and sign conversions are necessary to display the current position on the graph. The data from this array is also used by WritePosData.vi when the Write Position Data option is activated.

The GPS/File Status, which indicates whether the GPS unit is correctly tracking or the data from the file is being read correctly, is implemented by comparing the position latitude and longitude. A latitude or longitude of 0 degrees will cause the signal to be activated, therefore alerting the user to a possible problem. 0 degrees for latitude or longitude was implemented due to the rare occurrence that the aircraft will be flown exclusively on the Prime Meridian or the Equator. The multiplication operator is used for the comparison to eliminate possible round off errors.

The final region involves the airspeed calculation. NMBetTwoPts.vi is called to determine the distance flown between the aircraft's current position and the position on the last cycle. For Simulation Mode and File Mode, the airspeed is found by dividing the distance by time delay which was implemented. In GPS Mode, however, the airspeed is found by diving the distance by the difference in seconds. This method is necessary since no set delay is implemented in GPS Mode, allowing the system to execute as quickly as possible, maximizing the frequency of updating the instruments' data.

The Main Module is terminated by the user activating Labview's STOP feature.

---

[1] Williams, Ed. *Aviation Formulary v1.22*, http://www.best.com/~williams/avform.html, p. 3
[2] Bennett, Peter. *NMEA FAQ*, http://vancouver-webpages.com/peter/nmeafaq.txt, p. 7
[3] Bennett, p. 10
[4] Lethan, Lawrence, *GPS Made Easy*, Mountaineers, Seattle, Washington, 1995, p. 75
[5] Williams, p. 3
[6] Williams, p. 4