Research and Implementation of Instrument System on a Light-Duty Electric Aircraft Simulator

Feng Tian, Yonggang Sun[a], Guanglei Meng and Yongtao Yu

School of Automation, Shenyang Aerospace University, Shenyang 110136, China

Abstract. Considering the demands of the instrument system on a light-duty electric aircraft simulator, a semi-physical simulation instrument system, which is based on virtual reality and microcontroller technologies, is designed and implemented. Meanwhile, some key technologies are discussed and a general development method is put forward in this paper. After being completed, this simulated instrument system is connected with the flight-computing system to test its performances. The results show that it has real effect, stable operation and real time response. In practice, the instrument system not only meets the demands of the light-duty electric aircraft simulator, but also can be seen as a certain reference to develop the instrument system of other aircraft simulators.

1 Introduction

Flight simulator is a ground equipment to simulate the conditions of aircraft's flight. The simulation of instrument is essential to a semi-physical flight simulator or a pure virtual flight simulator because it affects the performance of flight simulation and the coverage of training tasks. The instrument system is also one of important technical indexes to determine the fidelity of flight simulator system [1]. Virtual instrument technology possesses the features of short development cycle, fine fidelity and reusability, so it has distinct advantages in the field of instrument simulation [2]. Meanwhile, microcontroller, with the features of easy design, is used to collect and process signals in this system. Combining the virtual instrument technology and microcontroller technology, this paper discussed the related design methods of a semi-physical simulation instrument system.

2 The design of the instrument system

2.1 The framework of the instrument system

As shown in Figure 1, the system structure mainly includes the software and hardware parts. Specially, the software part includes some aeronautical instruments such as altimeter, airspeed meter, vertical speedometer, side-slip angle indicator, magnetic compass and central integrated instrument because they are all necessary to the light-duty electric aircraft simulator. The hardware mainly implements the collection, processing and transmission of operation signal. To realize data exchange between the software and the hardware parts, serial communication technology is adopted in this system. It is clearly known that the inputs of the instrument system are the flight data and the dashboard operation signals, and the outputs are the display information of the virtual instrument and the status of the panel indicator.

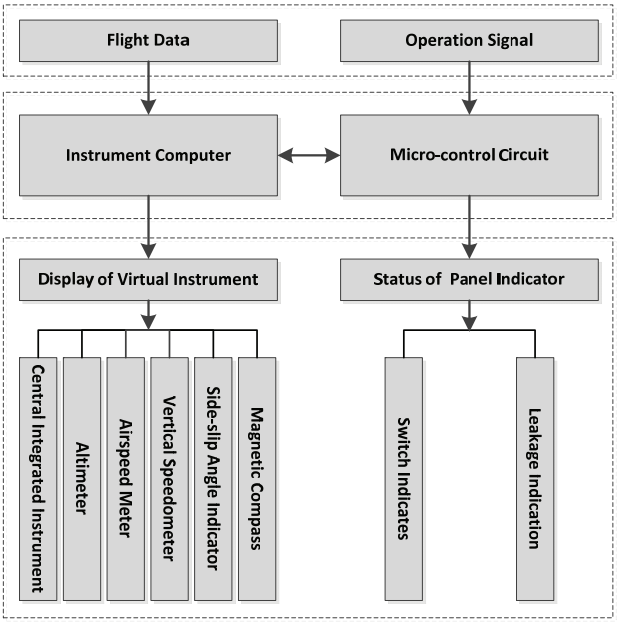

Figure 1. The Framework of Instrument System

2.2 Software development of the instrument system

2.2.1 Introduction to the development platforms

[a] Yonggang Sun: sunyonggangwangyi@163.com

The software of this instrument system is designed with GL Studio and VC++ 6.0 platforms. GL Studio is a professional instrument simulation software launched by United States DISTI company, and it is able to be used to create real-time, two-dimensional or three-dimensional graphical interface on Windows or Linux operating system [3]. The 3.2 version GL Studio includes a design window and a control window. In the design window, users can design instrument models with various graphic elements such as rectangle, circle and sphere. The control window contains coding area, interface setting options and the tree structure of GL Studio objects which created by users [4]. The behaviors of instrument objects can be realized by calling GL Studio library functions. Meanwhile, code generator in the control window can transform a GL Studio file into C++ or Open GL source codes, and the codes can be further compiled and debugged on Visual C++ 6.0 platform to generate an application program.

In addition, GL Studio provides two development modes, namely standalone mode and component mode. The standalone mode file can be used to create an application program. While the component file can only be embedded into a standalone file to take effect [5]. According to the practical requirements, users can chose different modes to design virtual instrument system on GL Studio platform.

2.2.2 Development steps

On the GL Studio platform, the development of virtual instrument system is generally divided into two steps, namely the establishment of instrument model and the realization of instrument objects' behaviors. The instrument model can be built by GL Studio graphical editor, or other graphics editing software [6]. The behaviors of instrument objects can be realized by calling GL Studio library functions. The actions of library functions such as rotation, sliding and characters display can meet the general demands of virtual instrument.

In order to maintain the independence, integrity and flexibility in the process of developing instrument objects, all objects are designed with component mode. After all instrument components are completed, they are all loaded in a standalone file to generate the application program. In the standalone file, the position, size, and attitude of all components can be adjusted for layout. This development method is not only easy to realize the reuse of instrument components, but also obtain the aim of multi-person cooperation [7]. In particular, the flow charts for developing virtual instrument on GL Studio platform is shown in Figure 2, in which the (1) is about component mode and the (2) is concerning standalone mode.

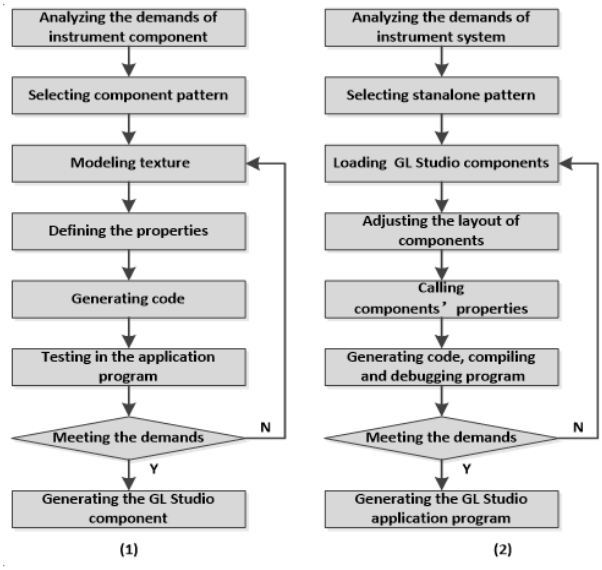

Figure 2. The Flow Chart of Developing Virtual Instrument

2.2.3 Model establishment and behavior realization of the instrument components

The aeronautical instruments required in this system include the central integrated instrument, the altimeter, the airspeed meter, the vertical speedometer, the side-slip angle indicator and the magnetic compass. In this section, three representative instrument components (the altimeter, the airspeed meter and the central integrated instrument) are analyzed and implemented because the principle of designing the others is similar to these.

According to the actual instrument appearance, use the GL Studio tool to build the model of instruments, and then analyse the mathematical features which indicates the movement characteristics of instrument. Generally, to realize the instrument behaviors, the flight data transmitted into instrument components are transformed into angle value or the object's dimension in logical units defined by GL Studio [8].

(1) Altimeter component

The altimeter component is as Figure 3, and it has two pointers and a text box. The long pointer denotes the height range of 0~999 meter, the short pointer represents the height range of 0~5000 meter, and the text box displays the atmospheric pressure.

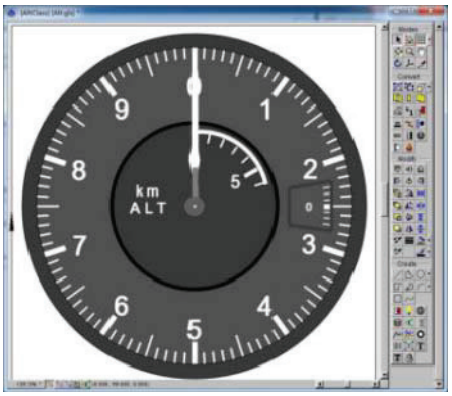

Figure 3. Altimeter Component

For the altimeter component, the long pointer, short pointer and text box (which actually are the member objects of the Altimeter component object) are respectively named Long_Pointer, Short_Pointer and Pressure_Text. The methods which control the behaviors of the altimeter's pointers and the text box are defined as Alt_Pointers(const float & value) and Alt_Text(const float& value), respectively. According to the scale attributes of altimeter shown in Figure 4 and Figure 5, the formula (1) gives the value of rotation angle for the long pointer, and the formula (2) denotes the angle value of the short pointer.

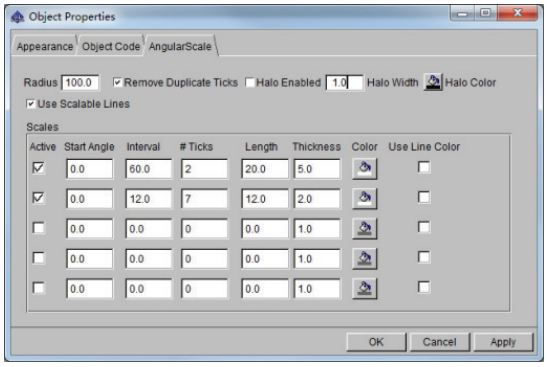

Figure 4. The Scale Attribute for the Altimeter's Long Pointer

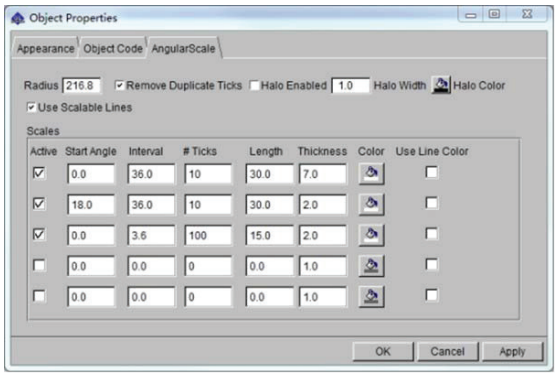

Figure5. The Scale Attribute for the Altimeter's Short Pointer

$$Long\_Angle = Alt\_Value \times 0.36 \quad (1)$$

$$Short\_Angle = Alt\_Value \times 0.012 \quad (2)$$

Where Alt_Value represents the real height value, Long_Angle and Short_Angle respectively denotes the rotation angle of the long pointer and short pointer. Considering the two formulas above, the codes to realize the altimeter's behaviors are added as follows.

```
//controlling the movement of the altimeter's pointers
Alt_Pointers(const float & value)
{
   float   Long_Angle, Short_Angle;
   Long_Angle=value *0.36;
   Short_Angle=value*0.012;
   // controlling the rotary movement of the long pointer
   Long_Pointer->DynamicRotate(Long_Angle,Z_AXIS);
   //controlling the rotary movement of the short pointer
   Short_Pointer->DynamicRotate(Short_Angle,Z_AXIS);
}
```

```
//controlling the display of the text box
Alt_Text (const float& value)
{
   //defining a character array to store string
   Char AirPressureChar[64];
   //transforming value into string
   sprintf(AirPressureChar,"%d",value);
   //displaying the string to in the text box
   Pressure_Text ->String(AirPressureChar);
}
```

Where DynamicRotate(float angle, axis) is to set rotation around X, Y or Z axis by angle, and String(char* s) is used to set the ASCII string that is to be displayed in the text. However, it should be noted that the two methods are all from GL Studio library.

(2) Airspeed meter component and its behavioral realization

As shown in Figure 6, the airspeed meter has only one pointer to denote the speed range of 0~220 (Km/s). The pointer is named as Pointer and the method is defined as Speed_Pointer(const float& value). Figure 7 displays the airspeed meter's scale attribute, which determines the rotation angle of the pointer as formula (3).

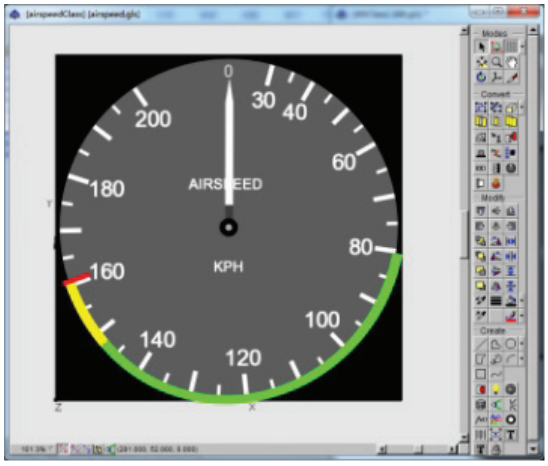

Figure 6. Airspeed Meter Component

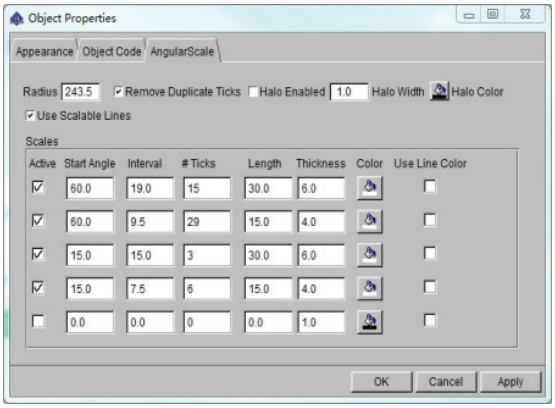

Figure 7. The Scale Attribute for the Airspeed Meter's Pointer

$$SpeedAngle = \begin{cases} Value \times 0.5 & 0 \le Value < 30 \\ Value \times 1.5 - 30 & 30 \le Value < 60 \\ Value \times 0.9 - 54 & 60 \le Value \le 220 \end{cases} \quad (3)$$

Where Value is the speed value transmitted to the airspeed meter and SpeedAngle represents the angle value. For the above mathematical features, the codes is added in the airspeed meter's method as follows.

```
//controlling the movement of the airspeed meter's
//pointer
Speed_Pointer (const float& value)
{
  //rotation angle of the pointer
  float SpeedAngle;
  //transforming the speed value into the rotation angle
  if(0<=value&&value<=30)
  {
    SpeedAngle=value*0.5;
  }
  if(30<value&&value<60)
  {
    SpeedAngle=value*1.5-30;
  }
  if(60<=value)
  {
    SpeedAngle=value*1.9-54;
  }
  // controlling the clockwise movement of the pointer
  Pointer->DynamicRotate(
  -SpeedAngle, Z_AXIS);
}
```

(3) Central integrated instrument component

The central integrated instrument component is shown in Figure 8, and it contains many subordinate components. Here only two subordinate indicators (the throttle indicator and the voltage indicator) are analyzed and implemented.

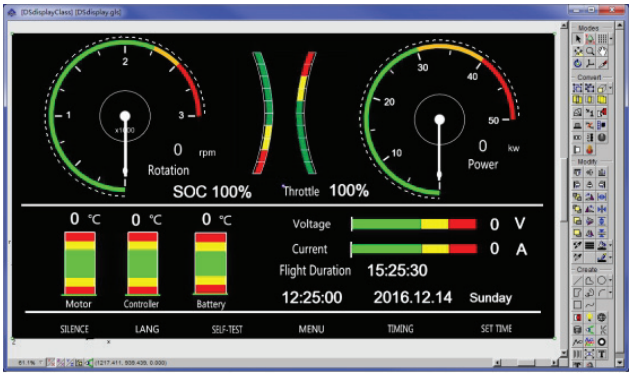

Figure 8. Central Integrated Instrument Component

It can be shown from Figure 9 that the throttle indicator consists of a cambered progress bar and a text box. The progress bar is composed with ten different parallelogram objects. The green part implies that the throttle's state is normal, the yellow part indicates prompt state and the red part represents warning state. For a created GL Studio object, its member method Visibility() can control the visibility of itself, so it is utilized to control the visibility of the respective parallelogram objects in this component. Meanwhile, the throttle value is used to define the number of the parallelogram objects whose state are to be visible. The object names of the ten parallelograms from bottom to top are respectively named Green1, Green2, Green3, Green4, Green5, Green6, Yellow7, Yellow8, Red9 and Red10, and the method that controls the visibility of the parallelograms is defined as Throttle_Display(const oat& value). The parts of the codes are given as follows, and the "..." represents the omitted codes which can be obtained via the similar principle.

```
//Control the throttle indicator
Throttle_Display (const float& value)
{
  ...
  //according to the value, Green1, Green2, Green3 and
  //Green4 are visible. Green5, Green6, Yellow7,
  //Yellow8, Red9 and Red10 are invisible.
  If(30<value&&value<=40)
   {
     Green1->Visibility(true);
      Green2->Visibility(true);
     Green3->Visibility(true);
      Green4->Visibility(true);
     Green5->Visibility(false);
      Green6->Visibility(false);
     Yellow7->Visibility(false);
      Yellow8>Visibility(false);
     Red9->Visibility(false);
      Red10->Visibility(false);
   }
   ...
}
```

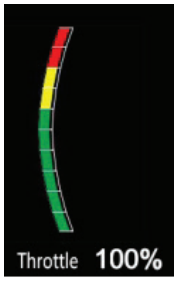

Figure 9. Throttle Indicator

Figure 10. Voltage Indicator

As shown in Figure 10, the voltage indicator mainly consists of a slide bar, a pointer and a text box. Its specifications are given in Table 1.

Table 1. The Specifications of Voltage Indicator

Color	Green	Yellow	Red
Voltage(V)	0~220	220~320	320~420

The width of the slide bar is 420. Obviously, the voltage specifications shown in the above table matches the width of the slide bar, and the mathematical characteristic of the movement for the voltage indicator's pointer can be obtained as the formula (4).

$$X\_Position = Voltage\_Value \quad (4)$$

Where X_Position represents the pointer's position in the X-axis direction of the slide bar and Voltage_Value is the real voltage value which is to be displayed on the

slide bar. The pointer is named Voltage_Pointer and the method that controls the pointer is defined as Voltage_Display(const float& value). The codes are as follows.

```
//Control the voltage indicator
Voltage_Display(const float& value)
{
  //Controlling the pointer moves in the positive X axis
  //direction of the sliding bar
  Voltage_Pointer>DynamicTranslate(value,0,0,false);
}
```

Where DynamicTranslate(float x, float y, float z, bool relative = false) can be utilized to set the dynamic translation to the specified value for all axes (in logical units), and parameter x, y and z respectively means the new dynamic translation for the X axis, Y axis and Z axis.

2.3 Hardware design of the instrument system

The hardware circuit structure shown in Figure 11 mainly include a signal conversion module, a microprocessor module and a serial communication module.

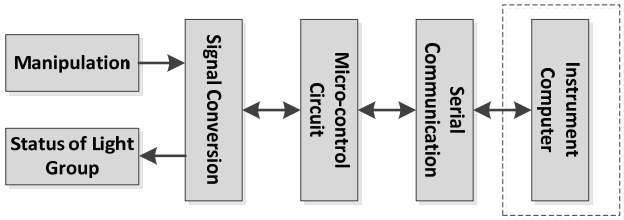

Figure 11. Hardware Structure of the Instrument System

Considering the demands of this instrument system. The STC90C52 microcontroller is used to collect and process the operation signals. Meanwhile, the serial communication module is designed with RS-232 standard to realize the data transmission between instrument computer and microcontroller. In fact, the hardware circuit not only meets the needs, but also reduces the development costs.

3 System integration and test

After all instrument components is finished, the primary task is to create a standalone file, then all the components can be loaded into it. Meanwhile, the position, size, and attitude of all the components can be adjusted for layout. Moreover, in the built-in method caclulate(double time), all component objects' behaviors can be realized by calling library functions.

In the standalone file, the central integrated instrument component object is named CentralPanel, the altimeter object is AltPanel and the airspeed meter object is AirspeedPanel, respectively. The parts of the codes are as follows, and there INFO is a struct variable saving the flight data.

```
caclulate(double time)
{
  // Do not remove (for normal operations)
  objects->Group::Calculate(time);
  ...
  //transmitting altitude value to the altimeter
  AltPanel-> Alt_Pointers (INFO.Altitude);
  //transmitting atmospheric pressure value to the
  //altimeter
  AltPanel-> Alt_Text (INFO.AirPressure);
  //transmitting speed value to the airspeed meter
  AirspeedPanel->Speed_Pointer (INFO.AirSpeed);
  //transmitting throttle value to the central integrated
  //instrument component
  CentralPanel-> Throttle_Display(INFO.Throttle);
  //transmitting voltage value to the central integrated
  //instrument component
  CentralPanel-> Voltage_Display(INFO.Voltage);
  ...
}
```

To test the system's performance, the simulated instrument system is connected with flight-computing computer. The UDP protocol is adopted to realize network communication between the instrument system and the flight-computing system because it has the remarkable advantages for its fast transmission speed, and fine real-time [9]. Through repeatedly debugging and testing, the requirements of the instrument system is reached well, and the operating effect is shown in Figure 12.

Figure 12. The Operating Effect of the Instrument System

4 Conclusions

After the instrument system is connected with the flight-computing computer, the performance of the system is tested and the results show that it not only possess real effect, stable operation, but also has fine fidelity and good reusability. The practice has showed that this instrument system meets the demands of the light-duty electric aircraft simulator.

References

1. H.Wang, Y.Qiu, Microcomputer Information, **25**, 180, (2011).

2. Y.Wu, T.Yu, J.Zhang, F.Yu, J.Liu, Measurement & Control Technology, **l**, 140, (2015).
3. Q.Xu, Z.Ouyang, R.Zhu, Ship Electronic Engineering, **28**, 152, (2008).
4. Y.Zhao, Z.Li, Electronic Design Engineering, **24**, 35, (2016).
5. Y.Xu, F.Yang, Instrumentation and Measurement, **27**, 76, (2008).
6. Y.Sun, L.Wang, J.Chen, System Simulation Technology, **11**,151, (2015).
7. J.Xu, G.Li, H.Zhao, S.Fan, Aerospace Control, **28**, 70, (2010).
8. G.Liu, Z.Li, Microcomputer Information, **26**, 110, (2010).
9. H.Chen, J.Wu and X.Huang, Measurement & Control Technology, **32**, 89, (2013).