2017 International Conference on Military Technologies (ICMT)
May 31 – June 2, 2017, Brno, Czech Republic

On-line Analysis of Data From the Simulator X-Plane in MATLAB

Rudolf Jalovecký*, Radek Bystřický†
* University of defence, Kounicova 65, 66210 Brno, Czech Republic, e-mail: *rudolf.jalovecky@unob.cz*
† University of defence, Kounicova 65, 66210 Brno, Czech Republic, e-mail: *radek.bystricky@unob.cz*

***Abstract* — The paper deals with the possibility to upgrade the flight simulator X-plane by online analysis of data provided by the simulator itself on another computer. Using the MATLAB and its ability to read UDP protocol data transmitted through the internet connection can be evaluated and displayed or otherwise used. One of its uses can be to display selected data on virtual cockpit gauges known usually as a glass cockpit.**

Keywords - Computer networks; Data transfer; Aerospace simulation; Virtual machine monitors; UDP protocol.

I. INTRODUCTION

Flight simulators play a significant role during the pilot initial training. They not only saves the money needed for the whole teaching process but also increases safety and time effectiveness of the training process. Their level and complexity varies from the simples ones usually referred to cabin crew procedure simulator using only one ordinary computer and one screen to full mission simulators with realistic cabin interiors and aerodynamic profiles and often using movable platforms. Almost all types of simulator allows for some degree of handling unusual situation in flight [6].

The flight simulator X-plane, we use, belong to the middle class of simulators using only one computer to simulate all visual and flight situations. It allows visualizing all necessary flight instrument and the outside situation on as many monitors as the hardware allows it [7]. On top of that, it allows communicating with other computers running the same simulation program and thus creating network of simultaneously flying aircrafts or allowing other people to play a teacher role and to simulate unusual situations in flight without the pilot knowledge.

The X-plane simulator has several possibilities of sharing the data from the active flight. Firstly it can store all selected data on the disk into the standard *.txt file with predefined speed. Secondly, it can independently from the previous possibility transmit all selected data throughout the internet connection using UDP protocol to one or more recipients [2].

Those two possibilities allow for online and offline analysis of gathered data or to create other programmable add-ons allowing to create automatic systems for the flight automation or even to calculate the mathematical model used for the flight aerodynamic [4].

The main advantage of the UDP communication protocol with the internet connectivity is the almost real time ability with only minimal delays. The only delays are caused by the Windows operational system, which cannot be considered as a real time operational system.

II. UDP SETTINGS IN THE X-PLANE SIMULATOR

X-plane allows user to select groups of data sets that are defined in the *Data Input & Output* window as illustrated in the Figure1. Each group of data sets may contain up to 8 specific details about the flight based on the description of the UDP structure defined in the X-plane documentation in the Annex G [1].

The user can specify for each data set where should the data be sent (UDP, file, screen, cockpit) and what should be the rate of transmission, see Figure 1 in the bottom right corner or Figure 2. The transmission rate is by default set to 20 frames per second, which is a reasonable number. The communication speed is of course limited by the quality of the internet connection and the speed of the computer that runs the online MATLAB analysis and the complexity of the analysis as well.

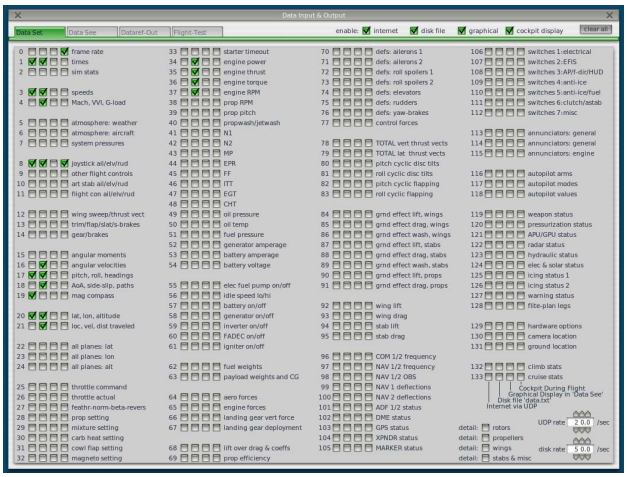

Figure 1. Data Input & Output“ window

Besides the data settings, the proper communication must be set in order to ensure its functionality. Namely, the IP addresses and ports play a key role. These settings can be found on the menu *Settings -> Net Connections -> Data* see Figure 3 and Figure 4. Predefined ports are:

- 49000 for data reception,
- 49001 for data transmission,
- 49002 for iPad application connectivity.

These port numbers can be changed as needed however must be inserted to both computers that we need to connect to ensure proper function.

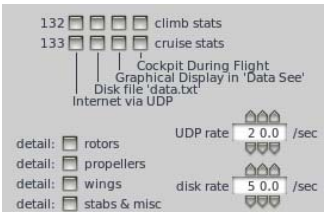

Figure 2. UDP rate and txt file storage setting

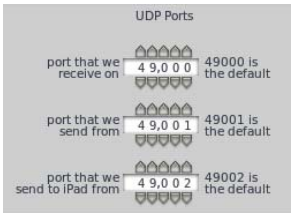

Figure 3. Port settings for UDP protocol

IP addresses are hidden in the next tab *Net Connections -> Data -> IP*. Again, both computers must be set reciprocally.

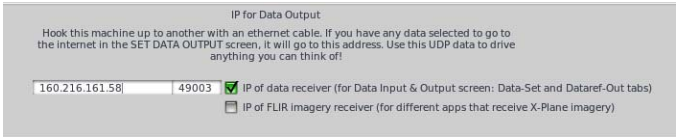

Figure 4. IP address and port settings for a master computer

III. UDP Structure and Data Description

The UDP protocol [2] is relatively simple communication protocol for data sparing. Its advantages and disadvantages both lay in the fact that data are transmitted without any kind of back confirmation and without any CRC checksum.

X-plane has simple format consisting of stream of datasets (segments) and as the Table 1 shows each segment has unique number assigned to it.

The format of UDP follows those rules:

- The length of the message is not constant and varies according to the total number of selected datasets. However, it is known in advance.
- First 5 bytes of the transmitted message is the message header and it can be read from the HEXA code as „DATA@“.
- Each dataset is transmitted via the UDP protocol as strings of bytes.
- First 4 bytes in each segment is dedicated to the segment number that corresponds to the dataset number from the Figure 1.
- Each segment is besides the header composed from 8 numerical information plus header for each segment.
- All data is sent out as 8bit byte in a form of a IEEE standard for floating-point arithmetic, meaning that you need to combine 4 successive numbers to obtain the transmitted numeric value see Table 1.
- All together, the segment has 4 bytes for header plus 8 times 4 bytes for numeric information, which is total of 36 bytes.
- The order of each byte in transmitted in reverse order, meaning that the least significant byte of the floating-point is transmitted first.
- Unused numeric information is filled by 0xC479C000 which corresponds to the number -999. User must handle this number on his own.

The example of the message including commentary about each number can be found in the Table 1 where the time segment and the compass segment are being received.

TABLE I. UDP Frame Structure

Shape of the Reports {Hexa}	importance of data	Value
{44}{41}{54}{41}{40}	message header	DATA@
{01}{00}{00}{00}	segment time {01}	
{05}{8B}{4A}{43}	real, time	202.54305
{DE}{B5}{1A}{40}	totl, time	2.4173503
{70}{87}{16}{40}	missn, time	2.3520164
{00}{00}{00}{00}	timer, time	0.0000000
{CD}{7A}{39}{41}	zulu, time	11.592481
{B0}{7A}{49}{41}	local, time	12.592453
{DA}{F8}{39}{41}	hobbs, time	11.623255
{00}{C0}{79}{C4}	None	-999
{13}{00}{00}{00}	segment compass {13}	
{0D}{76}{89}{42}	mag, comp	68.73057
{3D}{1A}{94}{C0}	mavar, deg	-4.628203
{00}{C0}{79}{C4}	None	-999
{00}{C0}{79}{C4}	None	-999
{00}{C0}{79}{C4}	None	-999
{00}{C0}{79}{C4}	None	-999
{00}{C0}{79}{C4}	None	-999
{00}{C0}{79}{C4}	None	-999

IV. Reception of the UDP Protocol Message in MATLAB

The big advantage for the code development is the fact that despite being paused, the simulator is transmitting the same data even in this state. The flight data are obviously the same while time segment is changing all the time. This offers unique possibility to tune the code perfectly to the needs.

The UDP channel settings for the data reception in the MATLAB environment is done by this series of commands where ipB is the IP address and portA and portB are port numbers.

```
udpA = udp(ipB,portB,'LocalPort',portA);
fopen(udpA);
```

The reading of bytes from the frame ensures the following cycle of commands.

```
while get(udpA,'BytesAvailable') == 0
    data = fscanf(udpA, '%d');
end
```

When the reading loop finishes all necessary data are stored in a memory and the whole message can be parsed and decoded. Before that the UDP port must be closed otherwise the communication line may crash. UDP port is closed by the *fclose* command.

An example demonstrating the processing of received message can be found in the flowchart in Figure 5. The initialization of the UDP protocol and communication line must be done at first followed by the data reception. Next step in the displayed by the flowchart is the header detection and then parsing of the segments.

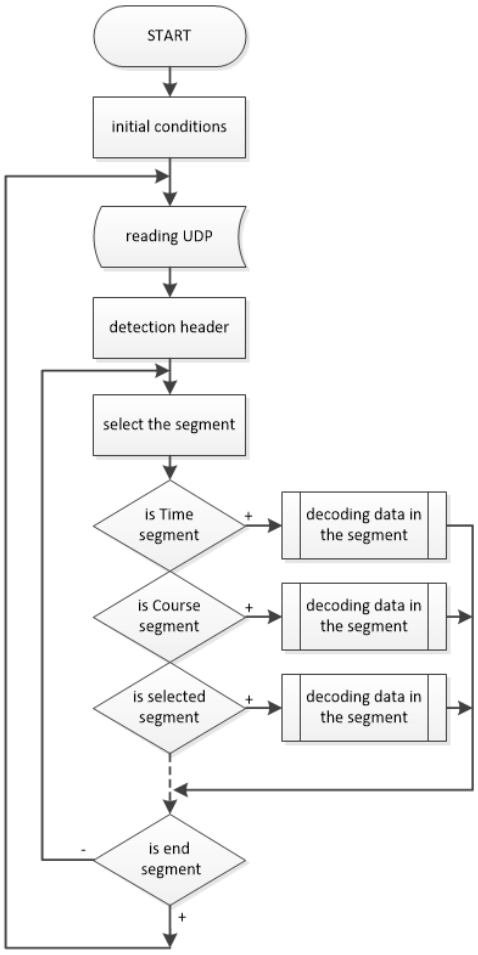

Figure 5. Flowchart of the UDP message reception parsing in MATLAB

Firstly, the header must be decoded in order to ascertain the segment type and then information is parsed and stored according to the segment type. As mentioned before each segment type contains different type of data. After the parsing step is done each 4byte, data information must be reassembled according to the above-mentioned IEEE standard to form the floating-point information. The program must contain as many segment subroutines as the number of segments in the message used by the application respectively as many as selected in the *Data Input & Output window*.

The decoding of the data in the segment is shown in the flowchart in the Figure 6. The input of this subroutine is the 36-byte long segment. The problem of the segment is that data are transmitted in reverse order meaning that the least significant byte of the floating-point is transmitted first. The subroutine has to pick 4 bytes out of the segment rearrange their order and decode the number into its floating-point representation according to the IEEE standard. This must be done for all 8 numeric information. The outputs of the subroutine are 8 numbers stored in their respective variable.

```
for i= 1:8
     index= 5+(i-1)*4;
     LetParChar =  [InputData(index+3) ...
                    InputData(index+2) ...
                    InputData(index+1) ...
                    InputData(index+0)];
     xx= unicode2native(LetParChar);
     dd= [dec2hex(xx(1),2) dec2hex(xx(2),2)…
          dec2hex(xx(3),2) dec2hex(xx(4),2)];
     LetParam(i) =  ieeesingle2num(hex2dec(dd));
end
```

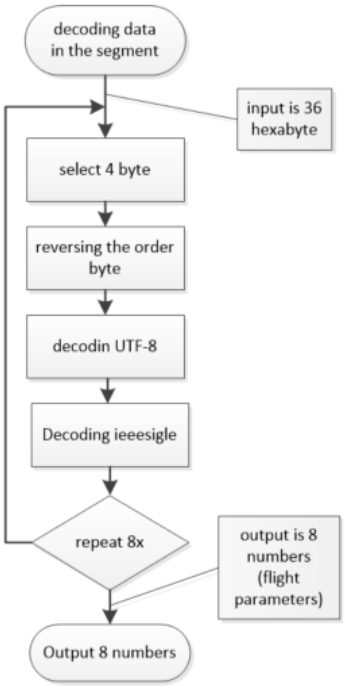

Figure 6. Flowchart of the segment decoding

V. EXPERIMENTAL DATA ANALYSIS

It is incredibly convenient during the initial stage of program development for data reception and data parsing to display the message data in some kind of intelligent way. In our case, we have chosen to show the data in several charts showing speed, altitude, heading and geographic coordinates of the flight see Figure 7. Some additional information as time needed for one loop of decoding are displayed in the main MATLAB windows as well for debugging purposes.

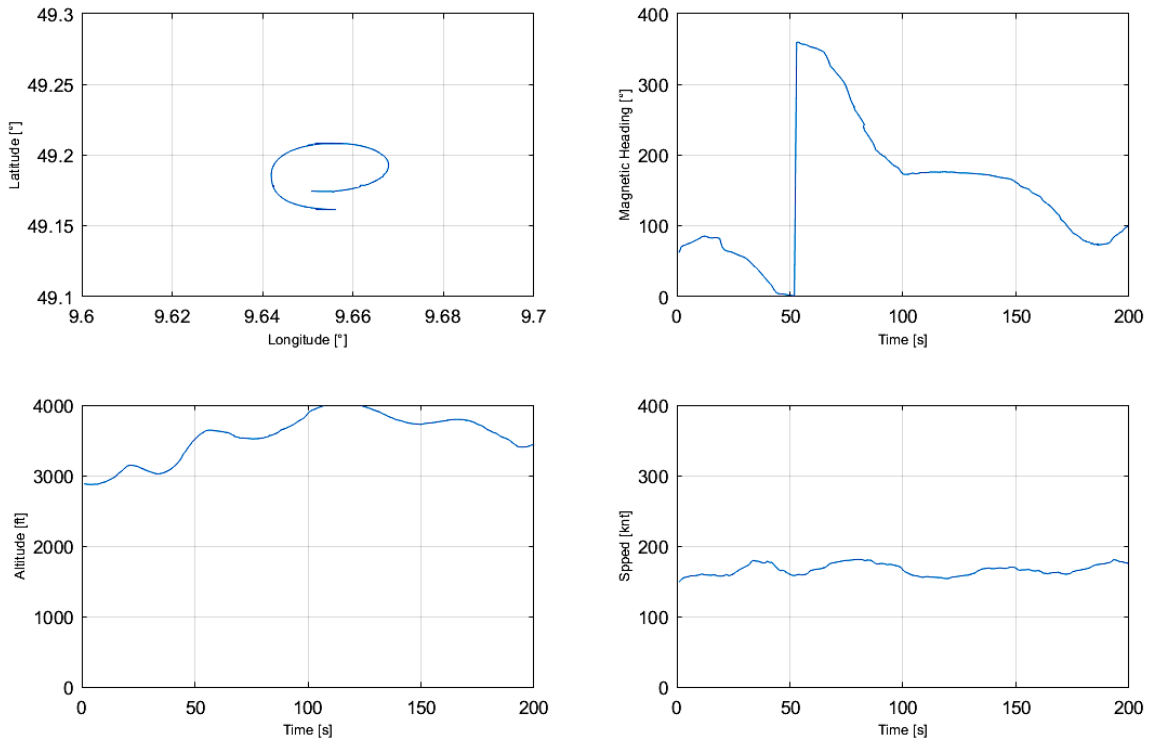

Figure 7. Sample of received data displayed in the MATLAB environment

Additional improvements can be done. For example, information about the pilot's reactions can be displayed as well in order to visually evaluate his reaction to random or preplanned events see Figure 8.

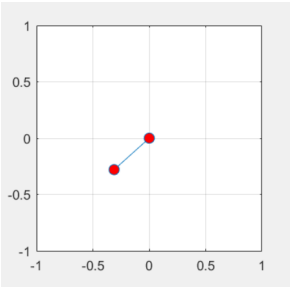

Figure 8. The control lever position indicator

Figure 9. Magnetic heading gauge indicator

One this data reception process is mastered the ability can be greatly improved by creation of realistically looking flight gauges as the one displayed in Figure 9.

Since the MATLAB GUI does not have any intelligent way to display data in the gauge form it is necessary in order to achieve some realistic look create our own flight gauges. The rest of this paper deals with the creation of such one. In order to show the process in its simplicity only the basic magnetic heading gauge will be shown.

The overall look was created and prepared in Microsoft Visio using several layers each incorporating different part of the gauge, see Figure 10. This approach allows separation of different parts as well as easy modification in the future. The left side of the picture shows stationary parts superimposed on top of each other in the right order and the right side of the picture show the moving (rotating) part of the instrument. The scale is based on the data obtained from the simulator rotated and subsequently inserted into the already existing picture as an overlay. This is the reason to have two separate files that must be used to achieve the right and realistic look. The white (or black) background in the middle of the picture is used to determine which part of the image should be transparent.

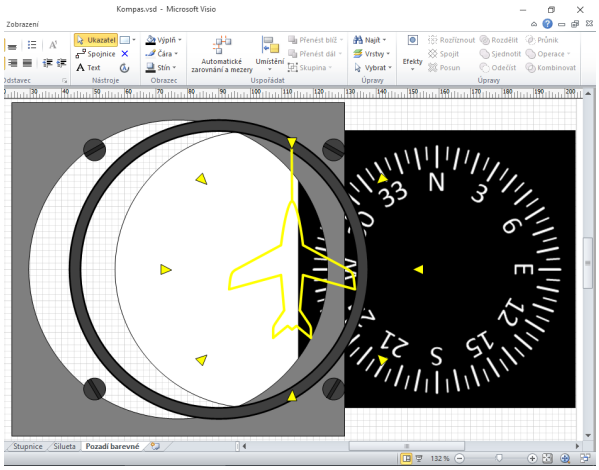

Figure 10. Separation of the heading instrument to its corresponding layers

Working with transparent images in MATLAB environment can be tricky. That not only the picture must be read and stored in a variable but also the transparency information, which is in MATLAB called *alpha channel*, must be stored as well. The alpha channel is essentially telling which pixel should be transparent. The user must decide whether he wants use the transparency or not. Next set of commands shows how to load an image and its transparency into the MATLAB.

```
% Loading of the rotating part of the gauge
A,map_A,transparency_A] = imread(Gauge.png');
imshow(A,map_A,'Border','tight');
axis image;
hold on;
% Loading of the stationary part of the gauge
[B,map_B,transparency_B] = imread('Overlay.png');
B_h= imshow(B,map_B,'Border','tight');

% Nastavení průhlednosti
set(B_h, 'AlphaData', (transparency_B));
hold off;
```

The command *imread* [3] can be used for that purpose where individual parameters (picture and alpha channel) are stored in the variables *A* and *transparency_A*. The final image is then displayed using the command *imshow* [3] where the parameters are set to disable the frame borders and axis. The last command *hold on* is there to assure that the next picture inserted into the same figure will keep the previous picture and thus superimpose both pictures on top of each other.

The process for the second picture is essentially the same as for the first one. The only difference is that the axes are already set and that the second picture must contain some transparent parts. To achieve the transparency for the second part the alpha channel must be set for the picture using the command set. In order to apply the set command the handle of the second picture must be known and it is done by storing the handle in the *B_h* variable. Once the indicator is displayed, the image-holding feature may be dropped.

While working with the alpha channel it is extremely important to note whether the loaded picture has the alpha channel filled or not. It will save many hours of work to notice that fact at the beginning. The empty alpha channel variable means that the picture does not have the transparency at all. Some picture formats does not support the transparency at all.

However to achieve some functionality the scale must be rotated before being used. To achieve the rotation according to the data send by the simulator the command *imrotate* [3] may be used.

The situation in case of the heading gauge is extremely simple since we rotate inside a full 360°degrees range, which simplifies the case. For other flight instrument, some transformation is needed. The motor RPM gauge for example has a 315° span for 3500 revolution per minute.

However, every time a picture is rotated it not only rotates but also changes its size according to the since of the rotational angle. Square positioned on its tip will be higher than positioned on its side. For the proper function of the overlay, keeping the same size of the picture is essential. Therefore, some parameters must be used to keep the image size all the time constant. This can be achieved by using the command crop, which will splice the redundant pieces created by the rotation of the image.

Figure 11. Created flight instruments

This is of course not the best method for picture display however it is the first approach for our purposes. Better solution would be to use some Open GL approaches [7]. In the future program Beaugauge instruments suite may be used instead of this somewhat primitive way of dealing with pictures. Despite using this simple approach relatively high refresh rate can be achieved. The *tic toc* command was used to measure the time needed to refresh the instrument.

```
Elapsed time is 0.165108 seconds.
```

This example has been for simplicity created by reading the source image over and over again which is not necessary. The pictures must be read only once at the beginning of the program, which can reduce, the time needed to draw the figure in MATLAB. Adopting this strategy may increase the speed so the refresh rate may go up to approximately 10Hz which is speed sufficient for the flight simulator, where usually the indicated value doesn't change that fast.

This approach was used to create several essential flight gauges like those shown in the Figure 11. The ultimate goal is to create cockpit similar to the Z-142 which is training aircraft used to train military pilots in the Czech army. To achieve this goal many more instrument ought to be designed.

VI. CONCLUSION

This paper concluded work that is trying to read and decode data from the flight simulator X-Plane using UDP protocol capability of the MATLAB environment. This approach allows simulator designers to incorporate more computers and visualization capabilities. It will serve two main purposes. One is to create glass cockpit for the simulator and the second is to measure flight data parameters as well as pilot reactions during unusual flight situations.

This can in future allow to enhance control feedback loops of the airplane and to incorporate simple types of automatic flight control systems based on selected parameters. Such approach allows students to directly influence airplane's behavior and to see the impact of their changes in the system. This approach however needs to control automatically the control surfaces of the airplane [6], which is something that we still did not master.

Nevertheless, this paper is more focused on the measurement and visualization of data produced by the simulator. Several flight gauges shown in the paper represents only the beginning of the process. It shows the tedious process used to create one flight gauge. Main portion of the work must be still done in order to achieve stable communication and fast display rates.

ACKNOWLEDGMENT

The work presented in this article has been supported by the Ministry of Defence of the Czech Republic (UoD development program "Research of sensor and control systems to achieve battlefield information superiority").

REFERENCES

[1] X-Plane 10. *Manual* [online] [cit. 2017-01-30]. Available from: http://www.x-plane.com/support/manuals/

[2] Nuclear Projects. *Communicating with X-Plane using C# and UDP* [online] [cit. 2017-01-30]. Available from: http://www.nuclearprojects.com/xplane/info.shtml

[3] Matlab Functions. *MathWorks documentation* [online]. [cit. 2017-01-30]. Available from: https://www.mathworks.com/help/matlab/functionlist.html

[4] ALLERTON, David. *Principles of flight simulation*. Chichester, U.K.: Wiley, 2009. Aerospace series (Chichester (England)). ISBN 0470754362.

[5] Beaugauge instruments suite. BeauGauge Inc. [online]. 2016 [cit. 2017-01-30]. Available from: http://www.beaugauge.com

[6] P. Paces, R. Jalovecky, E. Blasch and J. Stanek, "Pilot controller design using the CTU flight simulator for shared situation awareness," 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC), Prague, 2015, pp. 3E3-1-3E3-10.

[7] P. Mazurek, P. Paces and J. Filla, "High-fidelity terrain landscape EFIS visualization in comparative navigation to solve disorientation," 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC), Prague, 2015, pp. 3A3-1-3A3-6.