Motion Cueing Implementation for a Low Cost 6 DOF Flight Simulator [⋆]

Matheus Pena, Mauricio Becerra-Vargas [*]

[*] *São Paulo State University (Unesp), Institute of Science and Technology, Sorocaba, CEP 18087-180, Sorocaba, SP, Brazil (e-mail: matheus.pena@unesp.br, mauricio.b.vargas@unesp.br).*

Abstract: This paper presents the results of a study to implement a motion cueing algorithm for a six DOF flight simulator motion base. More specifically, getting specific forces and angular velocity from X-Plane Simulator, communication between X-Plane and dSPACE board 1104, and motion cueing parameter optimization. We evaluate the simulation realism by using an objective performance indicator. The specific forces and angular velocities at the simulator are measured by an inertial measurement unit (IMU). The experimental results revealed that the motion base is capable of producing motion simulation quality comparable to that produced by similar research simulator motion bases.

Keywords: Motion cueing motion; drive algorithm; flight simulator; X-plane; Parallel Robot.

1. INTRODUCTION

Flight simulation has evolved so that all the training can be performed in flight simulators, reducing the cost of training, the impact of aviation on the environment, and the training risk (Allerton, 2010). Besides the training benefits, flight simulation plays a fundamental role in developing new aircraft systems, pilot-vehicle control system analysis, and human perception research.

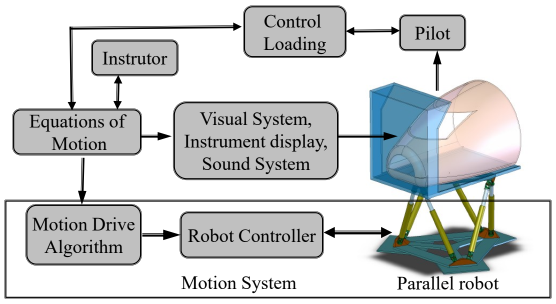


Fig. 1. Block diagram representation of a flight simulator

A flight simulator consists of various engineered subsystems synchronized to provide the pilot in the simulator with motion cues. Fig. 1 shows the main components of a typical flight simulator. The equations of motion determine the actual aircraft motion taking the action of the pilot, instructors inputs, aerodynamics, engine terms, etc. The visual system provides real-time images derived from the aircraft motion equations.

The motion system enhances the fidelity of the simulation by providing inertial cues. A simulator must have “motion" to achieve certain levels of training capabilities; the regulations demand it for all Full Flight Simulator (FFS) levels (FAA, 2012). Flight simulator motion systems consist basically of the motion-base (parallel robot), motion controller, and motion drive algorithm (MDA), as shown in Fig. 1. With the MDA, the trajectory from aircraft motion equations are translated into feasible motion-base trajectories regarding the limited motion envelope of the motion-base.

Since the work published in Conrad and Schmidt (1970) many works have been published on MDAs. Several variations of MDAs exist: classical algorithm (Grant and Reid, 1997; Asadi et al., 2015), adaptive-based algorithm (Nahon et al., 1992), and optimal-based algorithms (Telban and Cardullo, 2005; Dagdelen et al., 2009; Aminzadeh et al., 2012; Aykent et al., 2014; Miunske et al., 2019).

However, no information was presented on the practical implementation of these algorithms, precisely how specific forces are calculated from the aircraft dynamics, how they are sent to the MDA, and how to apply transformations coordinates between the aircraft model, the MDA, and the robot coordinate systems. Even studies related to flight simulators' development and implementation lack this information (Freeman et al., 1995; Baarspul, 1986; Reymond and Kemeny, 2000). This can be because most flight simulator implementation runs on a distributed network where each computer (node) performs a dedicated set of functions, typically with one computer for the flight model, another for the visual system, another for the motion system, etc.(Allerton, 2009).

The flight model computer runs mathematical models consisting of an extensive set of nonlinear differential equations with large aerodynamic data. The visual system computer receives data from the flight model computer, which contains sufficient information to produce a video signal for the projection system (Allerton, 2009).

It is not simple to build a distributed system running a flight model and a visual system managing synchronized

[⋆] The author gratefully acknowledge the financial support provided by grant 2017/14195-9, São Paulo Research Foundation (FAPESP).

sets of data, especially for the capacities of a low-cost simulator. Thus, it is a viable option to use a third-party flight simulator software (X-Plane) and integrate it with the other subsystems.

In this context, the subject of this work is the integration of X–Plane into the motion system of a low-cost motion-base flight simulator. We implemented the real-time control system of the motion-base in a dSPACE DS1104 controller board with ControlDesk, Matlab, and Simulink interfaces.

The remainder of the paper is organized as follows: Section 2 briefly describes the parallel robot topology and inverse kinematics. Section 3 describes X-plane inertial and body-aircraft reference frames. Section 4 explains the X-plane's basic plug-in structure to get the X-plane aircraft's specific forces and angular velocity. Section 5 briefly describes the classical motion drive algorithm and its reference frames. Section 6 explains the communication between X-Plane and Dspace 1104 hardware using serial communication. Section 7 describes the configuration and block diagram of the experimental setup and the MDA Simulink configuration and explains the optimization of the MDA parameters. Section 8 describes the experimental results using an objective performance indicator to measure the flight simulator fidelity. Finally, conclusions are addressed in Section 9.

2. MOTION-BASE - ROBOT TOPOLOGY

The 6-U$\underline{P}$RU-type parallel robot (Fig. 2) consists of a moving platform linked to the fixed base by six extensible legs. Each leg is connected to the base platform by a universal joint and to the moving platform by a universal joint and revolute joint. A prismatic joint allows the change of the lengths of the legs. An electromechanical linear actuator drives each prismatic joint.

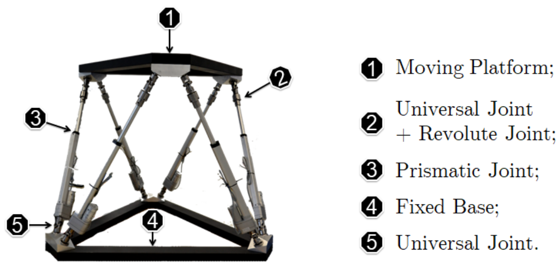


Fig. 2. 6-U$\underline{P}$UR-type Stewart platform

2.1 Inverse Kinematics

The main goal is to control the moving platform in task space while its actuators operate in joint space. Therefore, the desired Cartesian coordinates are transformed to joint space coordinates by inverse kinematic transformation.

To describe the inverse kinematics of the moving platform, we define frame $\{\mathbf{B_s}\}$ fixed with the base and frame $\{\mathbf{P_s}\}$ fixed with the moving platform as shown in Fig. 3. The leg vector $\mathbf{S}$ with respect to frame $\{\mathbf{B}\}$ can be written as:

$$\mathbf{S} = \Re\mathbf{p} + \mathbf{t} - \mathbf{b}, \tag{1}$$

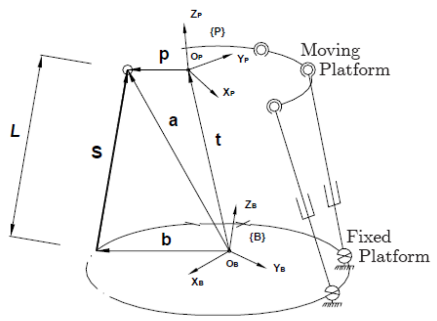


Fig. 3. Kinematic Analysis of One Leg (Becerra-Vargas and Belo, 2015)

where $\mathbf{b}$ is the position vector of the base leg point with respect to frame $\{\mathbf{B}\}$, $\mathbf{p}$ is the position vector of the moving platform leg point with respect to frame $\{\mathbf{P}\}$, $\mathbf{t}$, is the translation vector of the moving platform centroid with respect to frame $\{\mathbf{B}\}$, and, $\Re$, represents the relative orientation of frame $\{\mathbf{P}\}$ to frame $\{\mathbf{B}\}$ by using Euler angle rotations (body-fixed Z, X, Y sequence) leading to:

$$\Re = \begin{bmatrix} C\psi C\phi + S\psi S\theta S\phi & C\psi S\phi - C\phi S\theta S\psi & -C\theta S\psi \\ -C\theta S\psi & C\phi C\theta & -S\theta \\ C\phi S\psi - C\psi S\theta S\phi & C\psi C\phi S\theta + S\psi S\phi & C\theta C\psi \end{bmatrix}, \tag{2}$$

where $S(.)$ means sen(.) and $C(.)$ means cos(.) and where ψ (heading), θ (pitch) and ϕ (roll) are the angles of rotation around the Z-axis, X-axis and Y-axis, respectively.

Equation (1) represents the inverse kinematics problem in the sense one can compute the legs' lengths, i.e., $\|\mathbf{S}\|$, from the given position, $\mathbf{t}$, and orientation, $\Re$, of the moving platform.

3. X-PLANE REFERENCE FRAMES

X-Plane is a flight simulator for personal computers and has a very realistic flight model. It has received certification from the FAA and "can provide credit towards a private pilot's license, recurrence training, hours towards instrument training, and even hours towards an Airline Transport Certificate" (X-Plane, 2021a).

X-Plane employs a local (OpenGL) Cartesian 3D coordinate system for all 3D objects with origin on the earth at sea level at some "reference point"X-Plane (2021c). The positive X-axis points east from the reference point, the positive Y-axis points straight up away from the center of the earth at the reference point, and the positive Z-axis points south from the reference point, as shown in Fig. 4.

The X-Plane aircraft body-fixed coordinate frame has its origin at the center of mass with positive X-axis points to the right side, the positive Y-axis points up, and the positive Z-axis points to the tail of the aircraft, as shown in Fig. 4.

Using Euler angle rotations (body-fixed Y, X, Z sequence), aircraft kinematics expressed in the aircraft body-fixed frame can be expressed in the local frame by the following rotation matrix:

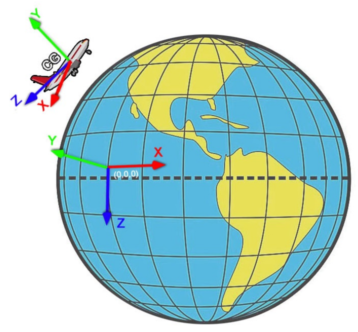


Fig. 4. Xplane OpenGL and aircraft reference frames

$$\Re = \begin{bmatrix} C\psi C\theta & C\psi S\theta S\phi - C\phi S\psi & C\psi C\phi S\theta + S\psi S\phi \\ C\theta S\psi & C\psi C\phi + S\psi S\theta S\phi & C\phi S\psi S\theta - C\psi S\phi \\ -S\theta & C\theta S\phi & C\theta C\phi \end{bmatrix}, \qquad (3)$$

where $S(.)$ means sen(.) and $C(.)$ means cos(.) and where ψ (heading), θ (pitch) and ϕ (roll) are the angles of rotation around the Y-axis, X-axis and Z-axis, respectively.

4. GETTING SPECIFIC FORCES AND ANGULAR VELOCITY FROM X-PLANE

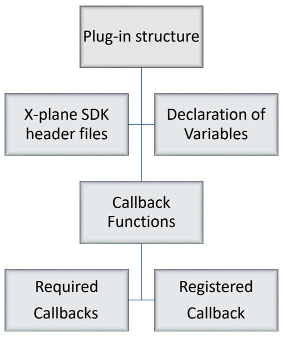


Fig. 5. Basic Plug-in Structure

X-Plane allows accessing flight data by using the X-Plane Plug-in software developer kit (Plug-in SDK) (X-Plane, 2021b). Plug-ins run inside X-plane and can read real-time flight dynamics data and send them over different communication protocols and can be written in various programming languages, e.g., C and C++.

Developing plug-ins in C++ requires a basic structure, as shown in Fig. 5. Plug-ins do not communicate directly with X-plane; instead, it communicates with X-Plane Plug-in Manager (XPLM) by calling functions in the XPLM to read data, create a user interface, etc. These calling functions are defined in the header files that start with XPLM as follow:

```
//SDK header files
#include "XPLMPlugin.h"'
#include "XPLMDataAccess.h"
#include "XPLMProcessing.h"
```

APIs defined in *XPLMDataAccess* and *XPLMProcessing* are used to read data and call registered callbacks during the flight loop, respectively. Global variables must be declared as data reference type (*dataref*) to read data from X-Plane through the *XPLMDataAccess*library. For example:

```
//Declaration of data reference variables
static XPLMDataRef accx_dataref;
static XPLMDataRef roll_dataref;
static XPLMDataRef p_dataref;
```

In this work are declared nine variables: the three linear accelerations relative to the aircraft's center of gravity, the Euler angles, and the three angular velocities of the aircraft. To send these data over the network are used registered callbacks running during the flight loop.

A callback is a function (inside the plug-in) that the XPLM calls to notify the plug-in of events or requirements, e.g., starting the plug-in and accessing the simulator data. There are five required callbacks that a plug-in must implement and that are called by the XLPM :

```
PLUGIN_API int  XPluginStart()  {...}
PLUGIN_API void XPluginStop()  {...}
PLUGIN_API void XPluginDisable(){...}
PLUGIN_API int  XPluginEnable()  {...}
PLUGIN_API void XPluginReceiveMessage()  {...}
```

Once identified a *dataref* in the declaration of data reference variables, they need to be found when the plug-in is first loaded (API XPluginStart) as follows:

```
accx_dataref = XPLMFindDataRef("sim/flightmodel/
    position/local_ax");
```

Then, inside the registered callback (serial communication), the value of that *dataref* can be read:

```
float accx = XPLMGetDataf(accx_dataref);
```

In this case, *XPLMGetDataf* only applies to float-type *datarefs.*

5. CLASSICAL MOTION DRIVE ALGORITHM

The classical MDA, initially developed by Conrad and Schmidt (1970) and later modified by Reid and Nahon (1985), has been widely used as the basis for the MDA in commercial simulators (Hodge et al., 2015). The input signals are the aircraft specific forces and angular velocities, and the output signal is the position and orientation of the parallel robot moving platform.

MDA is divided into three parallel channels, as shown in Fig. 6. The translation and rotational channel filter specific forces and angular velocities, maintaining the moving platform inside its workspace. The tilt coordination channel filters specific forces and transforms them into tilt angles. By tilting slowly, the moving platform creates an illusion of sustained acceleration.

5.1 Reference frames

In describing the implementation of the MDA, it is adopted the conventional body-fixed aircraft coordinate system. Reference frames are defined below and are shown in Fig. 7.

Frame $\{\mathbf{A}\}$ is an aircraft body-fixed frame and has its origin at the center of gravity with a positive Y-axis

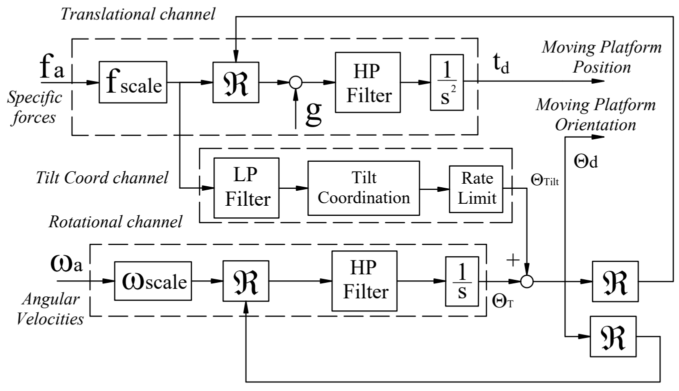


Fig. 6. Classical Motion Drive Algorithm

pointing to the aircraft's right side, the positive Z-axis pointing down, and the positive X-axis pointing to the nose. The transformation from X-plane aircraft frame to conventional aircraft frame is given by:

$$\Re^{\mathbf{A}}_{\mathbf{A_{xplane}}} = \begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix} \qquad (4)$$

The aircraft body-fixed frame $\{\mathbf{P_a}\}$ has the same orientation with respect frame $\{\mathbf{A}\}$. It is located in the same relative cockpit location (pilot station) as the simulator reference frame $\{\mathbf{P_s}\}$.

In the X-Plane-based simulation, the aircraft dynamics is calculated relative to aircraft's center of gravity. Then, total acceleration at the pilot station is given by:

$$\mathbf{a}_p = \mathbf{a}_{CG} + \boldsymbol{\alpha}_a \times \mathbf{P}_a + \boldsymbol{\omega}_a \times (\boldsymbol{\omega}_a \times \mathbf{P}_a), \qquad (5)$$

where $\boldsymbol{\omega}_a$ and $\boldsymbol{\alpha}_a$ are the angular velocity and angular acceleration of the aircraft, respectively.

The simulator reference frame $\{\mathbf{P_s}\}$ is fixed to the robot moving platform, and it is located at the centroid of the moving platform and has the same orientation with respect frame $\{\mathbf{P_a}\}$, i.e., the positive X-axis points forward, the positive Y-axis points toward the simulator pilot right hand and the positive Z-axis points downward. Frame $\{\mathbf{B_s}\}$ is the inertial reference frame fixed to the fixed-base platform of the parallel robot.

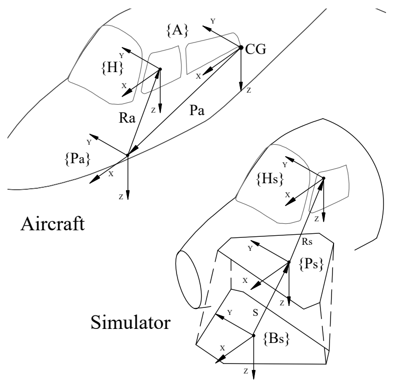


Fig. 7. Simulator reference frames

6. COMMUNICATION BETWEEN X-PLANE AND DSPACE BOARD 1104

The easiest way to get data from X-plane is using *The Data Input & Output* screen interface (X-Plane, 2021a) that enable sending data through the UDP network protocol to the address assigned for the user. Nevertheless, DS1104 does not support communication via UDP socket. The DS1104 I/O connector panel only contains connectors for Serial Interfacing. Therefore, we decided to use serial communication.

The implementation of serial communication is part of registered callbacks (Fig. 5), and it is declared as a function prototype:

```
float SerialComunication(float, float, int, void
    *);
```

We defined serial communication parameters inside of *XPluginStart* required callback. To call the serial communication function (registered callback) once every flight loop cycle, we used the following statement (inside of *XPluginStart*):

```
XPLMRegisterFlightLoopCallback(SerialComunication
    , -1.0, NULL);
```

To send the data via serial communication, we adopted to convert a floating-point data (getting by *XPLMGetDataf*) into an array of characters (*char* data type) as follows:

```
float SerialComunication(float, float, int, void
    *)
{
char lpBuffer[81]; // 72+9=81
char AccxFinal[8];
float accx = XPLMGetDataf(accx_dataref);
snprintf(AccxFinal, sizeof(AccxFinal), "%f",accx);
}
```

All variables are grouped in a buffer char array (*lpBuffer*) through a *for* loop statement. Identifier letters ranging from A to I are included between each variable.

dSPACE RTI block library provides Simulink Serial block for graphical configuration: The *DS1104SER_SETUP* block sets the serial connection parameter (Table 1), and the *DS1104SER_RX* block reads bytes from the serial interface, and its parameters are shown in Table 2.

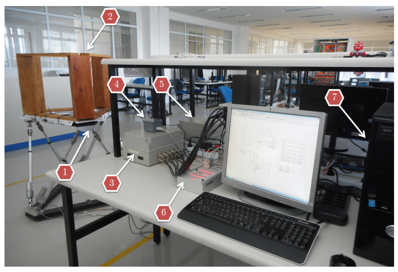
Table 1. DS1104SER_SETUP block parameters

Baud Rate	115200	Block size	81 bytes
Data Bits	8	FIFO size	1024 bytes
Stop bits	1	Copy after reception	14 bytes
Parity	Even	Overwrite	Replace old data

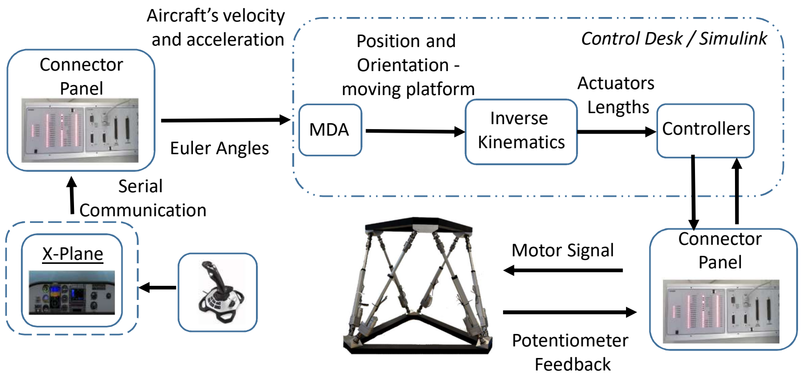
Table 2. DS1104SER_RX block parameters

Number of bytes to be received	81 bytes
Reception mode	Skip read operation

A Matlab routine is developed to unpack the byte streams transmitted by serial communication into separate variables. The size of each variable is 9 bytes (*char* type): the first byte is the *id* letter and the remaining 8 bytes correspond to the data itself.



(a) Configuration



(b) Block diagram

Fig. 8. The experimental setup

In writing the code routine, we highlighted the following points: if the number is negative, the sign byte is the second byte and from the third to the ninth byte is the numerical value; if the number is positive, the second to the eighth byte is the numerical value, and the ninth byte is filled with a NULL byte. The decimal-point character (ASCII "." character, *Dec* 46) must be identified in converting *char* to numeric value.

To validate the data received by serial communication, a code was written in the X-Plane plug-in to send the data to a text file (Data.txt) as follows:

```
#include "XPLMUtilities.h"
static FILE * gOutputFile;
PLUGIN_API int XPluginStart(){
char outputPath[255];
XPLMGetSystemPath(outputPath);
strcat(outputPath, "Data.txt");
gOutputFile = fopen(outputPath, "w");}
float SerialComunication(){
fprintf(gOutputFile, "Time=%f, ax=%f, \n ", time
    , accx);}
```

7. EXPERIMENTAL SETUP

The experimental setup is shown in Fig. 8. The system consists of a parallel manipulator (1) driven by electromechanical actuators, motor drives box(2), dSPACE Connector Panel (6), and dSPACE DS1104 R&D controller board (7). The experimental setup is implemented utilizing a PC equipped with a dSPACE DS1104 R&D controller board inserted in one PCI slot of the CPU. Simulink/Dspace Control Desk real-time program is used to executed the proposed algorithms in real-time. Analog voltage signals obtained from the potentiometers are sent via A/D converter (connector panel) to PC, where they are processed, and the control signals for the actuators motors are sent through D/A converter.

You can find more practical details about implementing the independent-joint control scheme in Gonçalves et al. (2019). We designed and implemented two independent-joint control schemes: a PID control strategy and an observer-based pole placement controller. The observer-based controller presented better performance and a better performance indicator (Eq. (7)).

7.1 MDA configuration

The MDA simulink implementation is shown in Fig. 9. Inputs to the MDA are scaled to keep the moving platform inside the parallel robot workspace. This produces the desired acceleration vector of the moving platform, which is filtered to extract only its high-frequency component to assure that the simulator will remain near its neutral position. Finally, acceleration is integrated twice to obtain the required displacements of the platform. We choose a high-pass filter of the translational channel as a second-order filter.

The rotational channel (bottom channel) is similar to the translational channel but acts upon angular velocities. We choose a low-pass filter of this channel as a second-order filter.

The central channel represents tilt coordination. It receives the scaled specific forces, which filter to extract low-frequency components that are then transformed into tilt angles. Finally, tilt angles are passed through a rate limiter to ensure that the tilt coordination will occur slowly enough to prevent the sensation of the angular rotation rate associated with the tilting. We choose a high-pass filter of this channel as a first-order filter.

MDA Tuning We optimized the algorithm parameters to find the most suitable parameters while respecting moving platform physical limitations and minimizing human perception error between the actual and simulator pilots. In this work, the optimization problem is written as:

$$min\ f(x)\ such\ that\ \ c(x) \leq 0, \tag{6}$$

where the cost function, $f(x)$, is a function of the following performance indicator:

$$F = \frac{\lambda_f}{a_{max}} + \frac{\lambda_\omega}{\omega_{max}}, \tag{7}$$

where

$$\lambda_f = \frac{1}{N}\sum_{i=1}^{N}\sqrt{e_{fi}} \quad , \tag{8}$$

and

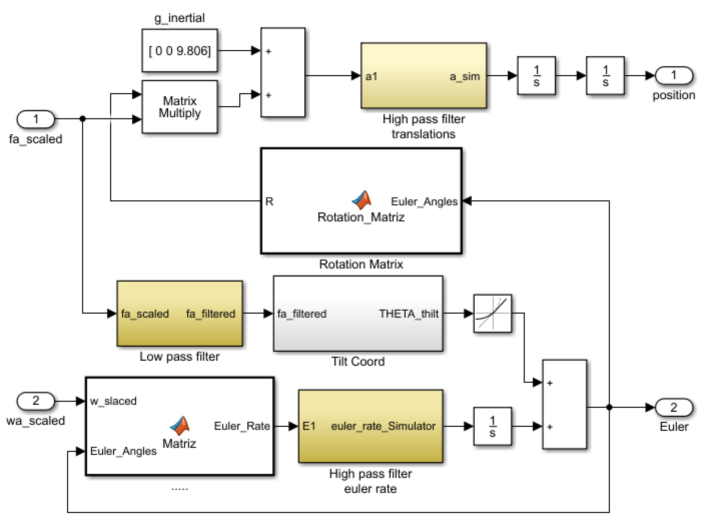


Fig. 9. MDA simulink configuration

$$e_{fi} = \left(f_{xi} - f_{xi}^{sim}\right)^2 + \left(f_{yi} - f_{yi}^{sim}\right)^2 + \left(f_{zi} - f_{zi}^{sim}\right)^2, \quad (9)$$

and where

$$\lambda_\omega = \frac{1}{N}\sum_{i=1}^{N}\sqrt{e_{\omega i}} \quad , \qquad (10)$$

and

$$e_{\omega i} = \left(\omega_{xi} - \omega_{xi}^{sim}\right)^2 + \left(\omega_{yi} - \omega_{yi}^{sim}\right)^2 + \left(\omega_{zi} - \omega_{zi}^{sim}\right)^2, \qquad (11)$$

and where, f is the desired specific forces (aircraft), and f^{sim}, is the achieved specific forces at the simulator. In the same way, ω is the desired angular velocity (aircraft), and ω^{sim}, is the achieved specific forces at the simulator.

The constraint function, $c(x)$, is used to place bounds on minimum and maximum actuator length of the parallel robot.

8. EXPERIMENTAL RESULTS AND DISCUSSION

An objective performance indicator is used to measure the fidelity of the motion system. This indicator (Eq. (7)) is intended to yield a single numerical value describing the average error between motion cues generated in the aircraft and those produced in a simulator (Pouliot and Gosselin, 1998).

The specific forces and angular velocities at the simulator, f^{sim} and ω^{sim}, are measured by an inertial measurement unit (IMU) located at centroid of the moving platform (Fig 10). A transformation coordinates must be realized to match the moving platform reference frame.

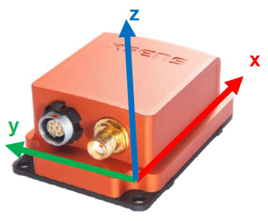


Fig. 10. MTi-700 - IMU

The inputs to the MDA are simulated from a takeoff rejection maneuver of a Boeing 747 performed in the X-plane software. In this maneuver, the turbines are activated. The brakes are disabled, allowing the aircraft to accelerate after a short period, then remove the power of the turbines and activate the brake to impose an acceleration in the opposite direction until the plane is entirely stopped.

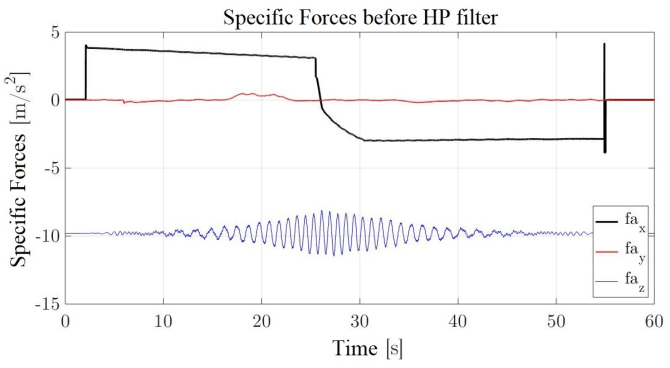


Fig. 11. Specific forcer before HP filter

Figure 11 shows the specific forces (x, y, and z coordinates) represented in the local moving platform reference frame before the HP filter. One observes a more significant component on the x-axis and a small value on the y-axis due to the accelerations imposed by the takeoff maneuver. An oscillation around the gravity value is observed on the z-component. Because of the characteristics of the maneuver, specific force on the x-axis has the typical shape of a step input.

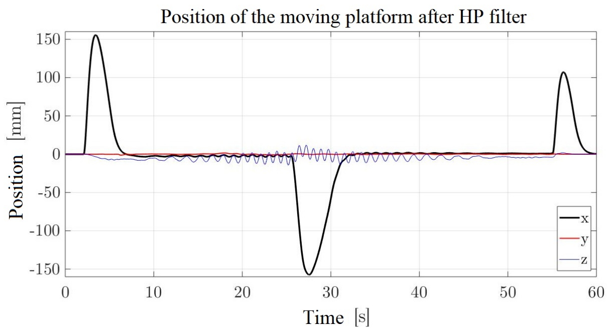


Fig. 12. Position of the moving platform after HP filter

From Fig. 12, the MDA algorithm carefully brings back the platform toward its neutral position without causing sensory conflicts (it is known as "washout").

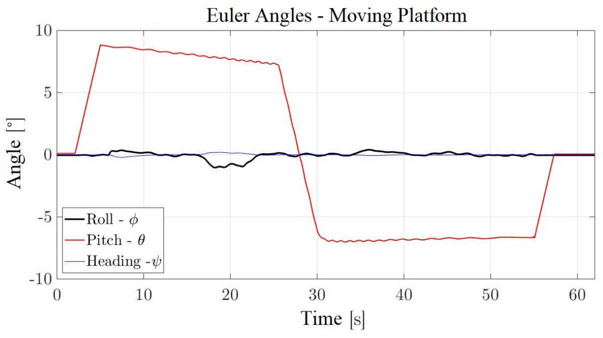


Fig. 13. Euler Angles of the moving platform

We also observed from Fig. 13 that sustained forward acceleration (x acceleration component) is represented through tilt coordination (θ angle), and the tilt angular velocity is limited to prevent the sensation of the angular rotation rate associated with the tilting (3 deg/s).

The MDA parameter optimization guarantees that the actuator stroke is kept inside the actuator stroke limits, as shown in Fig. 14. The stroke of the actuator is 300 mm, which corresponds to ± 150 mm from the middle position.

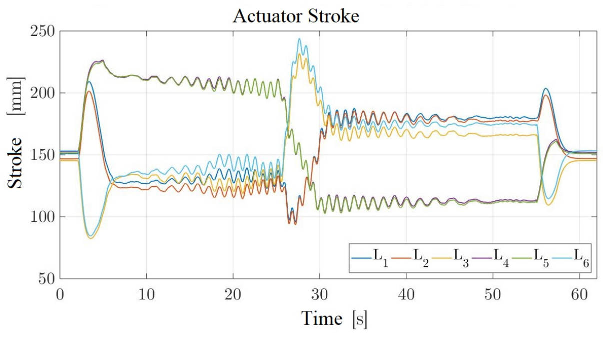


Fig. 14. Actuator Stroke

Finally, Fig. 15 shows the X-Plane's specific force and angular velocity and the measurement from the inertial sensor for the x-axis. The MDA parameter optimization matches the aircraft's linear acceleration better than the not optimized MDA.

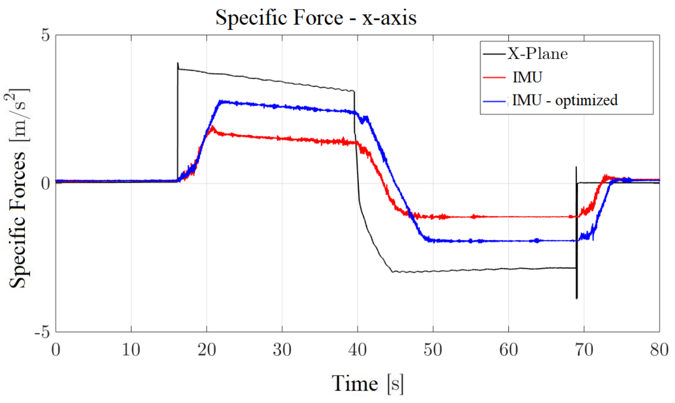


Fig. 15. Simulator Fidelity

On the other hand, we get a performance indicator value of $F = \%10,098$. Compared to smoothers simulation maneuvers, that is an excellent result, as shown in Pouliot and Gosselin (1998).

9. CONCLUSIONS

The proposed motion cueing implementation is a general method and can be used in many simulators with different topologies: serial, parallel, or hybrid kinematic structure. The investigation shows a third-party flight simulator software (X-Plane) integration with a flight simulator motion system using a simple serial communication. We suggest a performance index to conveniently quantify the motion system's efficiency as the reference for fidelity (realism) and to optimize the motion cueing algorithm. We performed practical tests, demonstrating that the proposed motion cueing algorithm yields much more realistic motion than the smoother simulation maneuvers in other research.

REFERENCES

Allerton, D. (2009). *Principles of flight simulation*. John Wiley & Sons.

Allerton, D. (2010). The impact of flight simulation in aerospace. *Aeronautical Journal*, 114(1162), 747–756.

Aminzadeh, M., Mahmoodi, A., and Sabzehparvar, M. (2012). Optimal motion-cueing algorithm using motion system kinematics. *European Journal of Control*, 18(4), 363–375.

Asadi, H., Mohamed, S., and Nahavandi, S. (2015). Incorporating human perception with the motion washout filter using fuzzy logic control. *IEEE/ASME Transactions on Mechatronics*, 20(6), 3276–3284.

Aykent, B., Merienne, F., Paillot, D., and Kemeny, A. (2014). Influence of a new discrete-time lqr-based motion cueing on driving simulator. *Optimal Control Applications and Methods*, 35(4), 454–467.

Baarspul, M. (1986). Flight simulation techniques with emphasis on the generation of high fidelity 6 dof motion cues. *Delft University of Technology, Department of Aerospace Engineering, memorandum m-553.*

Becerra-Vargas, M. and Belo, E. (2015). Dynamic modeling of a 6 dof flight simulator motion base. *Journal of Computational and Nonlinear Dynamics (doi 10.1115/1.4030013)*. doi:10.1115/1.4030013. URL http://dx.doi.org/10.1115/1.4030013.

Conrad, B. and Schmidt, S. (1970). Motion drive signals for piloted flight simulators. Technical report, USA. NASA CR-1601.

Dagdelen, M., Reymond, G., Kemeny, A., Bordier, M., and Maïzi, N. (2009). Model-based predictive motion cueing strategy for vehicle driving simulators. *Control Engineering Practice*, 17(9), 995–1003.

FAA (2012). *14 CFR FAR Part 60*. FAA-Federal Aviation Administration, Washington D.C, US.

Freeman, J., Watson, G., Papelis, Y., Lin, T., Tayyab, A., Romano, R., and Kuhl, J. (1995). The iowa driving simulator: An implementation and application overview. Technical report, SAE Technical Paper.

Gonçalves, A.H., Becerra-Vargas, M., and Silveira, M. (2019). Estudo experimental do controle de movimento de uma plataforma stewart do tipo 6- upur.

Grant, P. and Reid, L. (1997). Motion washout filter tuning: Rules and requirements. *Journal of Aircraft*, 34(2), 356–362.

Hodge, S.J., Manso, S., and White, M.D. (2015). Challenges in roll-sway motion cueing fidelity: a view from academia. In *Conference on 'Challenges in Flight Simulation*, volume 9, 10.

Miunske, T., Pradipta, J., and Sawodny, O. (2019). Model predictive motion cueing algorithm for an overdetermined stewart platform. *Journal of Dynamic Systems, Measurement, and Control*, 141(2), 021006.

Nahon, M., Reid, L., and Kirdeikis, J. (1992). Adaptive simulator motion software with supervisory control. *Journal of Guidance, Control, and Dynamics*, 15(2), 376–383.

Pouliot, N. and Gosselin, C. (1998). Motion simulation capabilities of three degree of freedom flight simulator. *Journal of Aircraft*, 35(1), 9–17.

Reid, L. and Nahon, M. (1985). Flight simulation motiobase drive algorithms: Part 1 - developing and testing the equation. Technical report, University of Toronto -

UTIAS, Toronto-Canada. Report No. 296.

Reymond, G. and Kemeny, A. (2000). Motion cueing in the renault driving simulator. *Vehicle System Dynamics*, 34(4), 249–259.

Telban, R. and Cardullo, F. (2005). Motion cueing algorithm development human-centered linear and nonlinear approaches. Technical report, NASA Langley Research Center. Technical Report CR-2005-213747.

X-Plane (2021a). *X-Plane 11 Desktop Manual*. URL `https://www.x-plane.com/manuals/desktop/`.

X-Plane (2021b). *X-Plane SDK Documentation*. URL `https://developer.x-plane.com/sdk/`.

X-Plane (2021c). *XPLMGraphics*. URL `https://developer.x-plane.com/sdk/XPLMGraphics/`.