APRICOT: Aerospace PRototypIng COntrol Toolbox. A Modeling and Simulation Environment for Aircraft Control Design

Andrea Ferrarelli[1], Danilo Caporale[2], Alessandro Settimi[2,3(✉)], and Lucia Pallottino[2]

[1] Vehicle Engineering, Università di Pisa, Largo Lucio Lazzarino 1, 56122 Pisa, Italy
[2] Centro di ricerca "E. Piaggio", Università di Pisa, Largo Lucio Lazzarino 1, 56122 Pisa, Italy
alessandro.settimi@for.unipi.it
[3] Department of Advanced Robotics, Istituto Italiano di Tecnologia, via Morego, 30, 16163 Genova, Italy

Abstract. A novel MATLAB/Simulink based modeling and simulation environment for the design and rapid prototyping of state-of-the-art aircraft control systems is proposed. The toolbox, named APRICOT, is able to simulate the longitudinal and laterodirectional dynamics of an aircraft separately, as well as the complete 6 degrees of freedom dynamics. All details of the dynamics can be easily customized in the toolbox, some examples are shown in the paper. Moreover, different aircraft models can be easily integrated. The main goal of APRICOT is to provide a simulation environment to test and validate different control laws with different aircraft models. Hence, the proposed toolbox has applicability both for educational purposes and control rapid prototyping. With respect to similar software packages, APRICOT is customizable in all its aspects, and has been released as open source software. An interface with Flightgear Simulator allows for online visualization of the flight. Examples of control design with simulation experiments are reported and commented.

Keywords: Aircraft control design · Aircraft dynamics simulation · Linear and nonlinear control

1 Introduction

Aircraft planning and control design requires a simulation environment that is highly configurable based on specific aircraft characteristics or mission objectives. Simulation environments are also necessary to design control systems and validate flight planning strategies. There are several solutions available as commercial or open source software packages. Among these, most are coded in programming languages as Java or C++ and focus on the simulation of specific vehicles as multirotor aircrafts or on generic aircrafts, see [1,2], while others are

J. Hodicky (Ed.): MESAS 2016, LNCS 9991, pp. 139–157, 2016.
DOI: 10.1007/978-3-319-47605-6_11

user interfaces for auto-pilot control systems, see [3,4]. In this sense, an easy to use tool for control design is missing. Other simulators are available with a commercial licence or closed source, see e.g., [5–7]. Our aim with APRICOT (Aerospace PRototypIng COntrol Toolbox) is to provide a simulator environment with a focus on control system design, which is multi-platform, highly customizable and open source[1]. For code, videos and details on APRICOT please refer to [8]. Moreover, being based on MATLAB/Simulink, it can be easily used for education purposes or by control system designer for rapid control prototyping in industry.

The 6 degrees of freedom nonlinear aircraft dynamics can be simulated with a preferred degree of accuracy, meaning that aerodynamic coefficients, stability derivatives [9], actuators and sensor dynamics, disturbances in measurements, wind gusts, control limits and other characteristics can be enabled separately and with different models by simply changing related MATLAB functions or Simulink diagrams. Moreover, custom atmospheric models can be used; by default the International Standard Atmosphere model [10] is implemented. A manual input interface has also been implemented to perturb the aircraft on-line during the simulation through an external gamepad to validate the effectiveness of the implemented control laws in response to various disturbances.

The possibility to visualize the simulation in a detailed 3D environment is obtained thanks to an interface with the open-source flight simulator FlightGear [11–13], as shown in Fig. 1. Both the toolbox and the flight simulator support multiple platforms (MacOS, Linux, Windows). Also, different aircraft models can be easily loaded from various sources like XML files, DATCOM files and even custom formats can be easily supported.

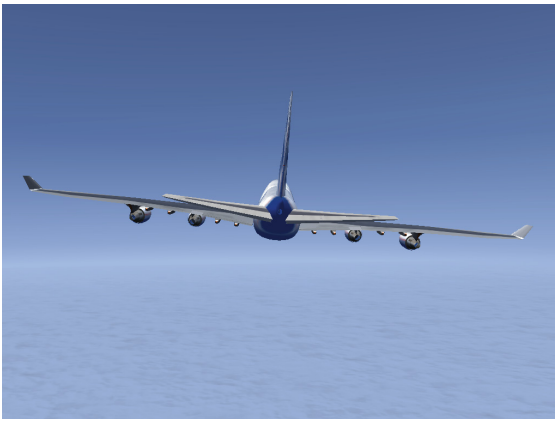


Fig. 1. APRICOT simulation animated in Flightgear.

[1] APRICOT Software available at http://aferrarelli.github.io/APRICOT/.

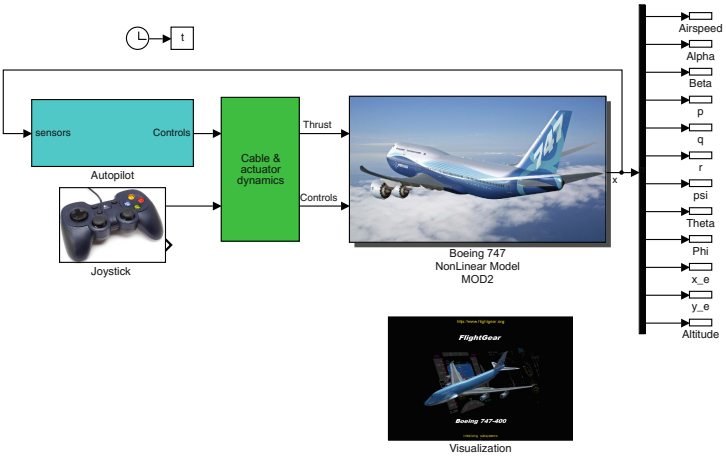


Fig. 2. APRICOT Simulink scheme

The whole system is organized so that it is highly usable and reconfigurable, in that any detail of the simulation can be easily customized from the aircraft model to the used control law. In particular, the main features of APRICOT are the possibility to easily customize the geometric and physical characteristics of an aircraft for what concerns the model and the environmental disturbances. In particular, different unmanned aerial vehicle (UAV) models can be included and simulated in APRICOT such as quadrotors and fixed wings UAVs. As for the control design, control laws can be implemented with Simulink blocks and MATLAB functions; several control laws are already provided in the toolbox. The APRICOT Simulink scheme is reported in Fig. 2.

Another major difference with other simulation environments, where a control law is typically tested on the subsystem for which it has been designed (e.g. a longitudinal controller is tested on the longitudinal subsystem), APRICOT allows multiple controllers to run simultaneously on a subsystem or on the full system dynamics. In other words, with APRICOT a more realistic behavior can be reproduced and tested with separated and/or redundant controllers, providing the control designer with an easy assessment of robustness in presence of failures.

Other than robustness with respect to failure, control laws can be tested to track a given flight trajectory. An optimization based planning is used to steer the system between desired configurations while minimizing fuel consumption and actuation effort and verifying state constraints. A feed-forward control, based on the obtained trajectory, can be used jointly with a feedback controller.

For both robustness and tracking purposes, various classical and modern control techniques, as those reported in [14–16], are available in APRICOT to control both longitudinal and lateral dynamics. Linear controllers have been implemented in the toolbox to test the system even while working far from the

operation point around which linear controllers have been designed. Moreover, within APRICOT the performance assessment can be conducted to analyze the whole system behaviour due to the full nonlinear dynamics of the model.

The APRICOT control laws are based on pole-placement, Linear Quadratic Regulator (LQR), Linear Quadratic Gaussian (LQG) and Linear Parameter-Varying (LPV) techniques and nonlinear Lyapunov based techniques (see, e.g., [17–19] for detailed discussions on these methods and similar application examples). In [20] we illustrate how these control laws have been designed and implemented in APRICOT.

To highlight the APRICOT toolbox characteristics and performance, several tests have been conducted with a Boeing 747 aircraft model. These experiments are available in the toolbox as demos and can be used as a guideline for control development and aircraft customization. The simulator has been tested mainly on MATLAB R2016a and FlightGear 3.4.0. Some parts of the simulator rely on the Simulink Aerospace, MATLAB Control System and Optimization Toolboxes.

The paper is organized as follows. In Sect. 2 we illustrate the simulation model used in APRICOT. The usage and the customization features of the environment are shown in Sect. 3. Simulation results and performance are reported in Sect. 4.

2 Modeling and Simulation Environment

This section is dedicated to the description of the aircraft dynamics and the world model necessary to understand and use APRICOT simulator. For a more formal description and equations please refer to [20].

2.1 Dynamic Equations

The aircraft model considered in this work is based on the full nonlinear 6 DoF dynamics. This is a classical model available in literature, see, e.g., [14,15], and can be used as a starting point to customize any aspect of the simulation.

The system has 6 dynamic states x_d and 6 kinematic states x_k:

$$x_d = \begin{bmatrix} V & \alpha & \beta & p & q & r \end{bmatrix}^T \quad x_k = \begin{bmatrix} x_G & y_G & z_G & \phi & \theta & \psi \end{bmatrix}^T$$

where, referring to Fig. 3, $V = V_T$ is the airspeed, α is the angle of attack, β is the sideslip angle. p, q, r are roll pitch and yaw angular rates in body coordinates, respectively, x_G, y_G, z_G are center of mass coordinates of the aircraft and ϕ, θ, ψ are the Euler angles, in fixed frame.

The whole state vector is defined as $x^T := [x_d^T, x_k^T]$. The input vector is $u = [\delta_{th}, \delta_e, \delta_a, \delta_r]^T$, namely the thrust, elevator, aileron and tail rudder commands. The dynamics of x_d depends on the forces and moments acting on the system (*i.e.,* aerodynamic forces, engine thrust and gravitational forces) and it is expressed in body coordinates.

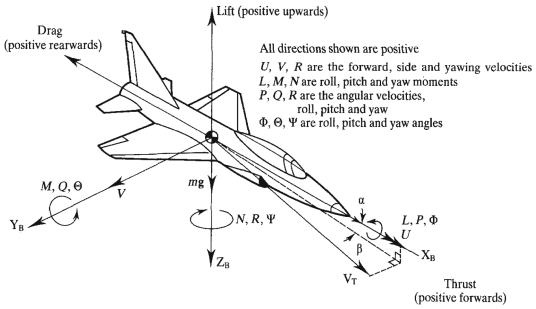


Fig. 3. Aircraft coordinated system, from [21].

2.2 Aerodynamic Forces and Moments

The aerodynamic drag D, lateral Y_A and lift L forces and moments, $\mathscr{L}$, $\mathscr{M}$ and $\mathscr{N}$, depend on the aircraft geometry, aerodynamic coefficients and dynamic pressure. They are given in wind axis coordinate frame by

$$\begin{cases} D = C_D \, q_{dyn} \, S \\ Y_A = C_Y \, q_{dyn} \, S \\ L = C_L \, q_{dyn} \, S \end{cases}$$

$$\begin{cases} \mathscr{L} = C_l \, q_{dyn} \, S \, b \\ \mathscr{M} = C_m \, q_{dyn} \, S \, \bar{c} \\ \mathscr{N} = C_n \, q_{dyn} \, S \, b \end{cases}$$

where the parameters S, $\bar{c}$ and b are aircraft geometric characteristics, see Fig. 4, and they correspond to the wing area, the mean chord of the wings and the wing length. The variable q_{dyn} is the dynamic pressure, C_D, C_Y, C_L are drag, lateral and lift aerodynamic coefficients, C_l, C_m, C_n are aerodynamic moment coefficients.

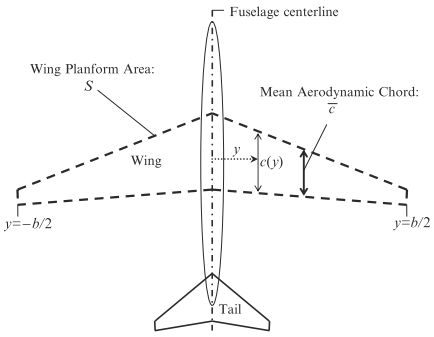


Fig. 4. Geometry of the aircraft, from [22].

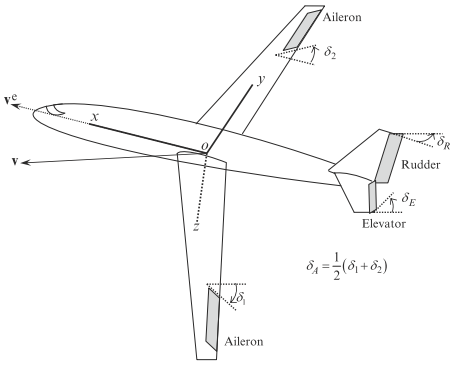


Fig. 5. Surface controls of the aircraft, from [22].

The dynamic pressure is computed as $q_{dyn} = \frac{1}{2}\rho V^2$ where the air density ρ is altitude dependent. For this reason an atmospheric model is required, such as the ISA (International Atmospheric Model) [10] used in this work.

The aerodynamic coefficients depend on the system states and on the control surfaces (elevator, aileron, rudder) and can be computed around an equilibrium configuration, see Fig. 5 as described next.

The longitudinal aerodynamic coefficients are

$$\begin{cases} C_D(\alpha, M) = C_D + C_{D_\alpha}\alpha + C_{D_M}\Delta M \\ C_L(\alpha, q, \dot{\alpha}, M, \delta_e) = C_L + C_{L_\alpha}\alpha + C_{L_q}\frac{q\bar{c}}{2V} + C_{L_{\dot{\alpha}}}\frac{\dot{\alpha}\bar{c}}{2V} + C_{L_M}\Delta M + C_{L_{\delta_e}}\delta_e \\ C_m(\alpha, q, \dot{\alpha}, M, \delta_e) = C_{m_\alpha}\alpha + C_{m_q}\frac{q\bar{c}}{2V} + C_{m_{\dot{\alpha}}}\frac{\dot{\alpha}\bar{c}}{2V} + C_{m_M}\Delta M + C_{m_{\delta_e}}\delta_e \end{cases}$$

where $\Delta M = (M - M_0)$, M is the Mach number and M_0 is the corresponding trimmed value.

On the other hand the laterodirectional aerodynamic coefficients are

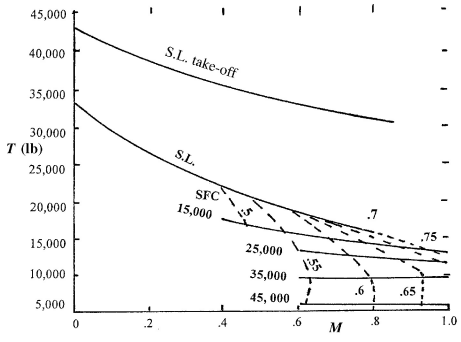
$$\begin{cases} C_Y(\beta, \delta_r) = C_{Y_\beta}\beta + C_{Y_{\delta_r}}\delta_r \\ C_l(\beta, p, r, \delta_a, \delta_r) = C_{l_\beta}\beta + C_{l_p}\frac{p\,b}{2V} + C_{l_r}\frac{r\,b}{2V} + C_{l_{\delta_a}}\delta_a + C_{l_{\delta_r}}\delta_r \\ C_n(\beta, p, r, \delta_a, \delta_r) = C_{n_\beta}\beta + C_{n_p}\frac{p\,b}{2V} + C_{n_r}\frac{r\,b}{2V} + C_{n_{\delta_a}}\delta_a + C_{n_{\delta_r}}\delta_r \end{cases}$$

The aerodynamic coefficients derivatives can be obtained from the literature for different equilibrium configurations, see [20,23] for details. They can be modified in APRICOT as shown in Sect. 3.

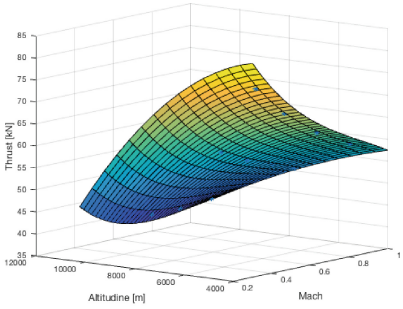
2.3 Other Forces and Actuator Dynamics

The engine response to the thrust lever is modeled for control purposes as a first order system with transfer function $G(s) = \frac{\tau}{s+\tau}$, where $\tau > 0$. More detailed models can be easily implemented based on user needs.

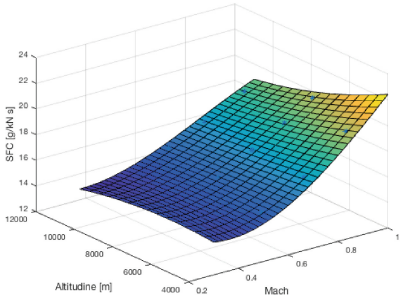
The peak thrust and the Specific Fuel Consumption (SFC) of the engine are not constant but depend on the altitude and the Mach number as shown



(a) Engine Thrust and SFC data.



(b) Thrust engine map.



(c) SFC engine map.

Fig. 6. Engine characteristics.

in Fig. 6a. Considering the engine curves and the equilibrium point, 3D engine maps are generated as in Fig. 6b and c via least squares polynomial fitting.

In the model are also included the actuator dynamics, rise limit and saturation of the control surfaces.

Finally the forces acting on the center of mass of the aircraft, expressed in body coordinates, are:

$$\begin{cases} X = -\cos\alpha\, D + \sin\alpha\, L - mg\sin\theta + X_T \\ Y = Y_A + mg\cos\theta\sin\phi \\ Z = -\sin\alpha\, D - \cos\alpha\, L + mg\cos\theta\cos\phi \end{cases}$$

where X_T is the engine thrust.

2.4 Sensor Noise and Wind Gusts

By default, all sensors are affected with a zero mean gaussian noise with a variance that depends on the output type, therefore on the sensor type. Noise models for sensors can be customized in APRICOT.

For example, in Table 1 we show the default variance values σ^2 considered for different outputs.

Table 1. Noise variance for longitudinal system.

Output	σ^2
V	10^{-2} m/s
θ	10^{-5} rad
h	0.1 m

Wind gusts are modeled via Simulink blocks. An example model of the wind speed V_{wind} is shown in Fig. 7.

$$V_{wind} = \begin{cases} 0 & x < 0 \\ \frac{V_m}{2}(1 - \cos(\frac{\pi x}{d_m})) & 0 \leq x \leq d_m \\ V_m & x > d_m \end{cases}$$

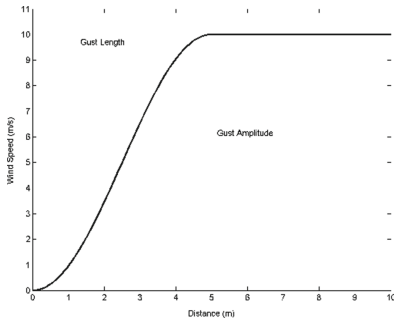


Fig. 7. Wind velocity with respect to the travelled distance.

3 APRICOT Environment

In this section we illustrate the main features available to the user to fully customize APRICOT environment. Indeed, it has been organized as a collection of MATLAB scripts, functions and Simulink blocks that can be singularly modified and tested creating several possible combinations.

First of all, APRICOT initialization process consists of running several scripts to load information on the aircraft and the world and to configure the control laws and the flight plan to be tested.

The main steps in the initialization process are represented in Fig. 8. Each step has associated MATLAB scripts that the user can edit to configure the simulation environment.

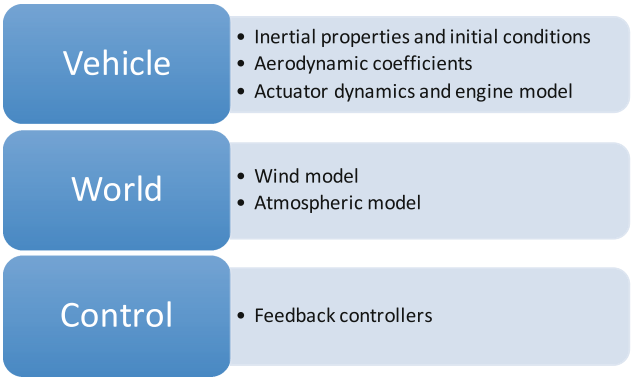


Fig. 8. Initialization of the toolbox.

Initialization of the Vehicle. The first script in the initialization process, named `init_aircraft.m` and reported below, regards the vehicle parameters and characteristics. The inertial properties and the initial conditions are loaded into the 6 DoF Simulink Block, reported in Fig. 9, that contains the dynamic equations.

Initialization of the vehicle - init_aircraft.m

```
%Mass of the vehicle
mass = 288772; % kg
%Inertia tensor for the nonlinear simulink model
Inertia = diag([24675560  44876980   67383260]);
Inertia(1, 3) = 1315126;
Inertia(3, 1) = 1315126;

%Initial position in inertial axes [Xe, Ye, Ze]
Init_pos = [0, 0, -6096];
%Initial velocity in body axes [u, v, w]
Init_vel = [252.98, 0, 0];
%Initial Euler orientation [roll, pitch, yaw]
Init_ang = [0, 0, 0];
%Initial body rotation rates [p, q, r]
Init_rot = [0, 0, 0];
```

The aerodynamic coefficients can be loaded as MAT files into the MATLAB function `init_aero_coefficients.m`. Forces and moments are computed in `Aircraft_Forces.m` by using the derivatives of the aerodynamic coefficients with respect to each state variable, as shown in Sect. 2.2. This is the output of the Forces & Moments block of Fig. 9, and the input of the vehicle model.

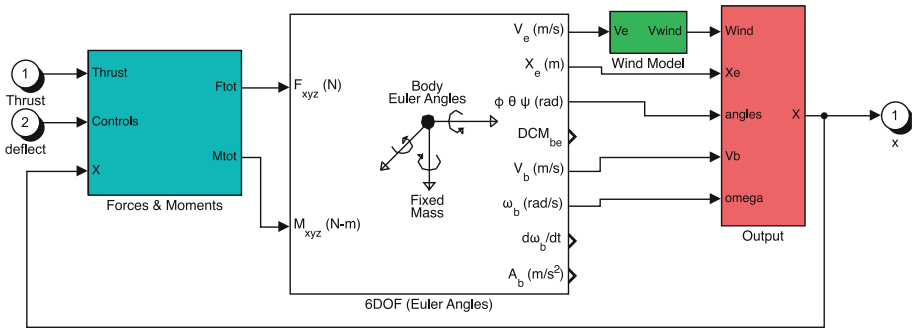


Fig. 9. Simulink model of a Boeing 747.

An example of aerodynamic coefficients matrix format

```
global CAeroMatrix
CAeroMatrix = [CD0 CDa CDap CDq CDM CDde 0 0 0 0 0;...
               0 0 0 0 0 0 Cyb Cyp Cyr Cyda Cydr;...
               CL0 CLa CLap CLq CLM CLde 0 0 0 0 0;...
               0 0 0 0 0 0 Clb Clp Clr Clda Cldr;...
               Cm0 Cma Cmap Cmq CmM Cmde 0 0 0 0 0;...
               0 0 0 0 0 0 Cnb Cnp Cnr Cnda Cndr];

save('CAeroMatrix')
```

The data shown here can be loaded from file, so that the end user does not need to manually edit this configuration scripts by itself. This can be done by selecting the path of external files in the MATLAB function `Aircraft_Forces.m`.

The dynamics of the actuators and the engine model are loaded in the Simulink Block represented in Fig. 10. Here the user can consider the nonlinearities in the engine. If a different actuator model is available, e.g. obtained

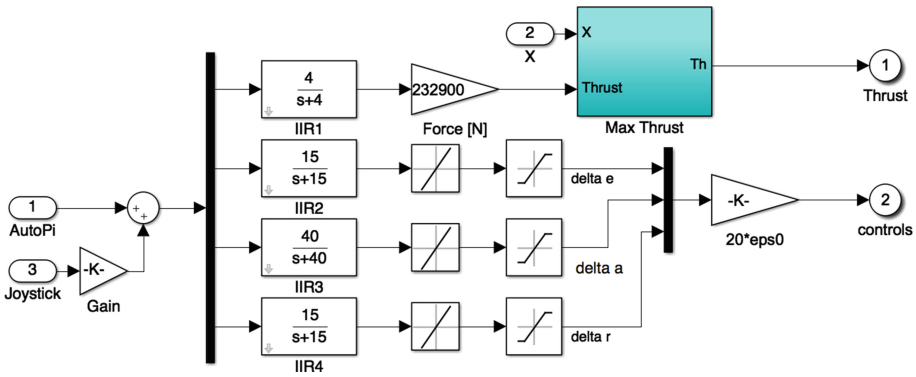


Fig. 10. Actuators dynamics and engine model.

by identification on a dataset, it can be used to replace the default dynamics, provided that the control interface is the same, *i.e.,* thrust (normalized as [0, 1] for null to full thrust), elevator, aileron and rudder (normalized in the interval [−0.5, 0.5]) as input signals and the effective thrust and angles of the control surfaces as outputs.

The experienced user can nonetheles choose to edit, replace or improve the provided MATLAB functions and Simulink schemes to customize the default behavior of any part of the simulator, exception made for the basic rigid-body equations.

Some key parameters, such as the control law and the disturbance characteristics, can be chosen directly from the control GUI, as shown in Sect. 3.

Initialization of the World. From `Aircraft_Forces.m`, APRICOT uses an atmosphere model to compute aerodynamic forces and moments. For example, the ISA model [10] parameters definition is shown below.

Atmosphere model

```
% Atmospheric model
alt = abs(h);
gamma = 1.4;
T0 = 288.15; p0 = 101325; M0 = 0.800488;
R = 287.053;
T = T0 - 6.5*alt/1000;                    %Temperature in K
press = p0*(1 - 0.0065*alt/T0)^5.2561;    %Pressure in Pa
rho = press/(R*T);                        %Density in kg/m^3
a = (gamma*R*T)^0.5;                      %Speed of sound in m/s
qdyn = 0.5*rho*V^2;                       %Dynamic pressure
```

The wind model is thus loaded in the Simulink Block reported in Fig. 11.

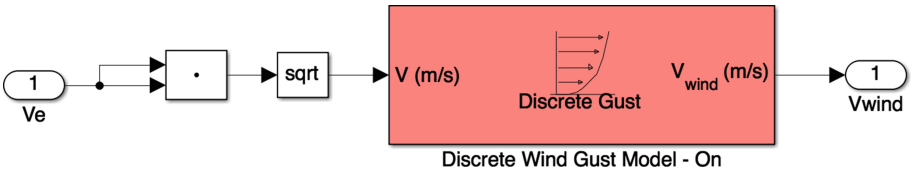


Fig. 11. Wind model.

Initialization and User Interface for the Control Design. Control laws setup in APRICOT follows the scheme in Fig. 12.

The user can choose to interact with the aircraft control system by means of a GUI (see Fig. 13) where default or user defined controllers can be loaded either on the full dynamics model or on the reduced longitudinal or lateral dynamics models. The GUI is organized in different panels to modify different aspects of the control laws acting on the aircraft.

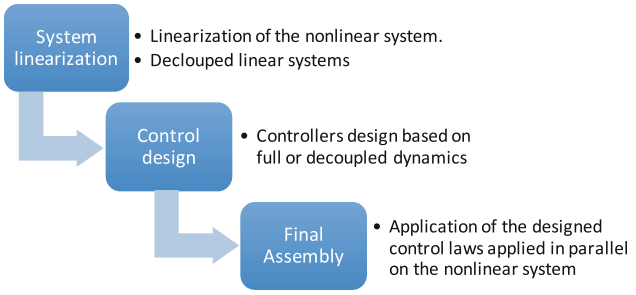


Fig. 12. Control walkthrough.

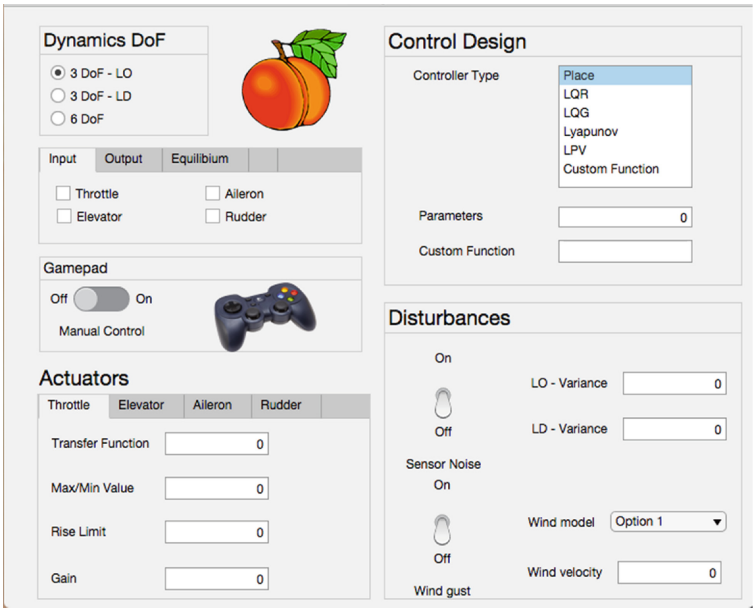


Fig. 13. The APRICOT control GUI.

User can also choose to use a subset of the available inputs and outputs for a given controller, enabling in this way fail-safe tests in different conditions of actuation or sensing. The same applies for disturbances, whose entity or models can be varied to assess the system performances in various conditions. This is particularly useful when designing a control law. By default the user can choose between the full nonlinear dynamics to be controlled, or between the longitudinal and laterodirectional subsystems. The longitudinal subsystem state vector is taken as $x_{LO} = [V, \alpha, q, \theta, h]^T$, with inputs $u_{LO} = [\delta_{th}, \delta_e]^T$. The laterodirectional subsystem state vector is taken as $x_{LD} = [r, \beta, p, \phi, \psi, y]^T$, with inputs $u_{LD} = [\delta_a, \delta_r]^T$.

In [20] the reader can find a more detailed description of the control laws implemented by default in APRICOT. These are the classical LQR, LQG, LPV and Lyapunov based nonlinear controllers and can be chosen in the Control Design panel, reported in Fig. 13.

Input control from an external pad can be enabled in the Gamepad panel, and the user can choose which commands are related to which control inputs from the Joystick block, see Fig. 2. The gamepad can be configured to provide disturbances on the auto-pilot control signals, or to control the airplane manually.

The actuator models described in Sect. 3 can be customized in the Actuators panel shown in Fig. 13. Moreover, from the Disturbances panel the user can test its controls under different conditions, as illustrated in Sect. 2.4.

Flight Planning. Flight planning is a critical task as the aircraft is subject to constraints in actuation, energy consumption and compliance with air traffic control specifications, in order to avoid midair collisions. Besides, control authority should be minimized whenever possible to extend the life of actuation surfaces and components. It is then essential to consider these aspects in order to generate reference trajectories to be used as feed-forward control inputs, and to validate the generated plans via simulation.

In APRICOT, this is achieved through optimization-based techniques. It is possible to consider constraints on the system dynamics, states and inputs, or to obtain plans for unconstrained problems. As shown in Fig. 14, the flight planning process is divided into three main steps.

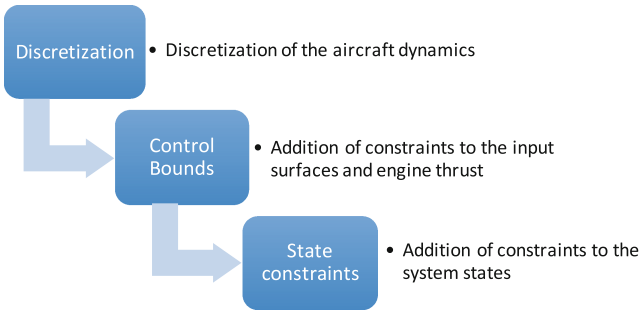


Fig. 14. Flight planning process.

As discussed in Sect. 2.3, the fuel consumption of the aircraft is described through SFC and thrust engine maps. The fuel flow is computed as the product of SFC times the thrust X_T of the engines

$$\dot{m}_{fuel} = SFC\ X_T.$$

A simplifying assumption in the planning setup can be made considering a constant SFC equal to its trimmed value, while the thrust X_T is taken as an optimization variable. It is worth noting that in APRICOT the complete computation of the fuel flow is used, see Fig. 6, and can also be customized.

Examples of flight plans are given and commented in Sect. 4. The MATLAB functions `LOoptfun.m` and `LDoptfun.m` implementing the flight planning algorithm take as input the following parameters:

- initial conditions,
- final conditions,
- initial time,
- final time,
- discretization period.

4 Simulations

In this section simulation results are reported for different simulated flights with different controllers among those illustrated in the previous sections, and given as example demos in APRICOT. The details of the implemented control laws are provided in [20].

Nonlinear and LQR Controls: First, we compare the performance of a Lyapunov based controller with an LQR controller on the nonlinear longitudinal dynamics in ideal conditions, *i.e.,* with no disturbances, but considering the actuator nonlinearities.

In Fig. 15 results are shown when initial condition is $x_0 = [V, \alpha, q, \theta]^T = [262.98, 0.2, -0.2, 0.1]^T$ around the trim condition $\bar{x} = [252.98, 0, 0, 0]^T$.

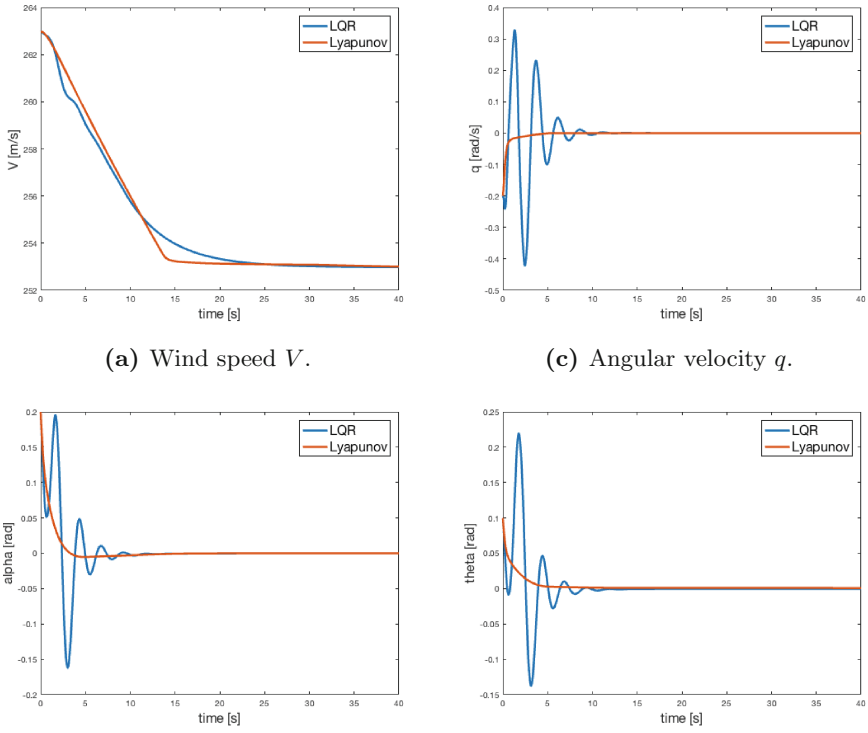
The system controlled with the linear control law has higher variations of angular velocities, angle of attack and sideslip angle with respect to the behaviour obtained with the Lyapunov based control.

Flight Planning: An example setup for longitudinal and laterodirectional planning is now illustrated.

For the longitudinal dynamics, we want to compute a plan to steer the system from the initial configurations $x_0 = [V, \alpha, \theta, q, h]^T = [252.98, 0, 0, 0, 6096]^T$ to the final configurations $x_f = [V, \alpha, \theta, q, h]^T = [252.98, 0, 0, 0, 7096]^T$, subject to saturations of the control variable as $-0.5 \leq \delta_e \leq 0.5$ and $0 \leq \delta_{th} - \delta_{th,0} \leq 1$, with a thrust control at the equilibrim $\delta_{th,0} = 0.7972$. The altitude overshoot is limited to 10 %.

In a similar way the laterodirectional planning is executed considering the following constraints on the ailerons and the tail rudder, corresponding to a maximum excursion of $\pm 20^\circ$, and given as $-0.5 \leq \delta_a \leq 0.5$, $-0.5 \geq \delta_r \leq 0.5$.

In this way it is possible to compute trajectories to steer the system between several waypoints to obtain a complicated path, as that in Fig. 16a whose first $100s$ are the result of the aforementioned planning problem.



(a) Wind speed V.

(c) Angular velocity q.

(b) Angle of attack α.

(d) Sideslip angle β.

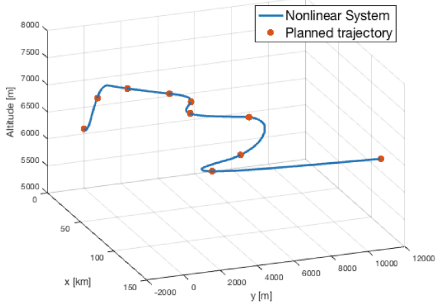
Fig. 15. Comparison of Lyapunov and LQR controllers

LQR and LQG Controllers: In the case of LQR and LQG control laws, in Fig. 16 we assess the performance of the simulator and the flight planner comparing the ideal aircraft trajectory with the simulated one considering the complete nonlinear model affected by disturbances. Note that for each waypoint reported in Fig. 16a different aircraft orientations are required and correctly tracked. The planned and executed trajectories are very close. To better appreciate the system evolution, videos of experiments are visible on APRICOT website [8].

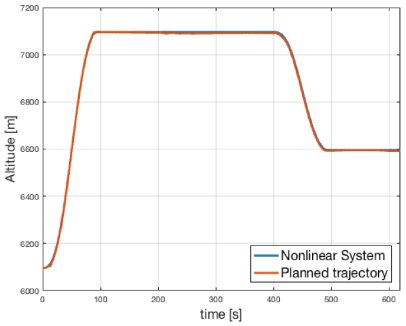
In Fig. 17 the improvement in the LQG control system performance with respect to the LQR case in presence of disturbances can be appreciated. This is due to the disturbance rejection capabilities of the LQG controller, that translates in reduced oscillations of the aircraft and a more comfortable flight for the passengers.

LPV Controller: Finally we consider the quadratic cost index

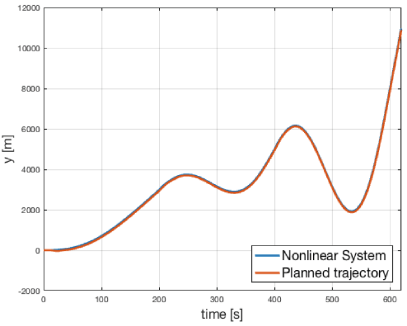
$$J(t) = \int_0^t x(\tau)^T Q x(\tau) + u(\tau)^T R u(\tau)\, d\tau$$



(a) 3D Trajectory.



(b) Altitude.



(c) y position.

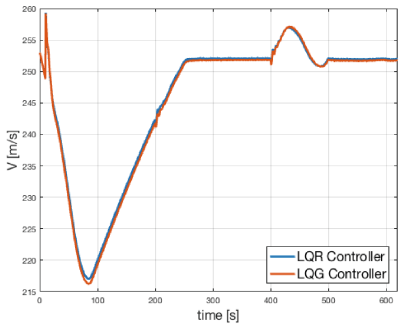
Fig. 16. Simulation results on the flight plan tracked with the LQG controller.

with diagonal matrices Q and R, obtained for the laterodirectional dynamics when the aircraft recovers from a severely perturbed initial state with $\theta = 0.2$ and $\phi = 0.2$, comparing the performance of an LQR and an LPV control law. In this example, the LPV control law has been obtained with multiple LQR controllers designed for different values of θ. The longitudinal dynamics is controlled with a single LQR controller. Simulation results are reported in Fig. 18. The LPV controller improves the trajectory tracking and energy efficiency performance of the aircraft with respect to the LQR controller, which is optimized for a single operating point and performs worse far from it.

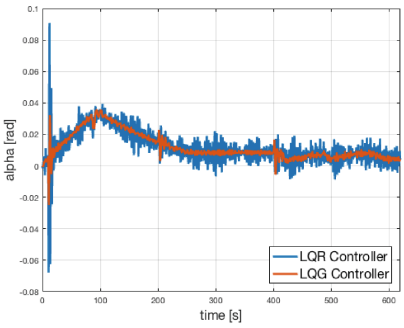
4.1 Simulator Performances

A comparison with existing toolboxes has not been possible, mainly due to the fact that, to the best of authors knowledge, there is currently no other open source toolbox that has the functionalities of APRICOT. Despite that, we show some performance indicators obtained for different use cases.

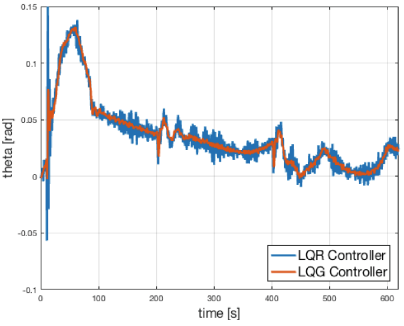
Regarding the flight planning phase, to solve the constrained optimization problem illustrated in the simulation section we registered an execution time of



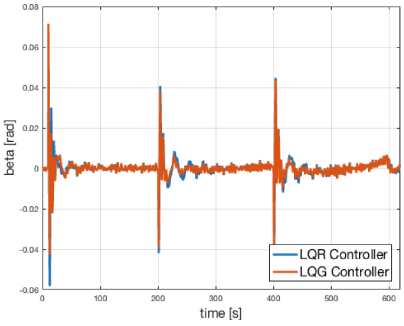
(a) Wind speed V.



(c) Angle of attack α.



(b) θ Euler angle.



(d) Sideslip angle β.

Fig. 17. Comparison of LQR and LQG controllers with noise and wind gust.

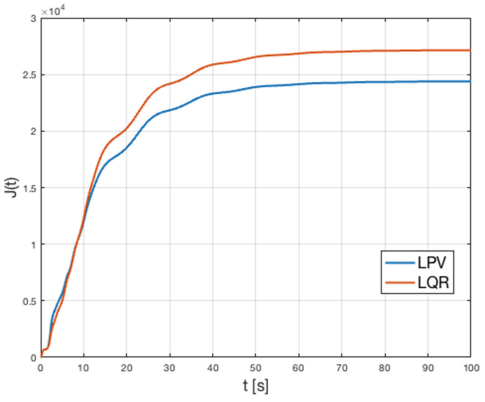


Fig. 18. Comparison between LPV and LQR performance.

1.7 s for the longitudinal dynamics and 0.4 s for the laterodirectional dynamics. Note that MATLAB Optimization toolbox solvers have been used, hence the interested user could improve these performances by using ad hoc solvers.

As for the simulation performance with controls in closed loop, we show different real-time factors (*i.e.,* the ratio between the flight duration and the simulation execution time) based on the complexity of the control laws. In Table 2 we report this data for each control law. Data have been obtained on a MacBook Pro laptop with 16 GB of RAM and 2.0 GHz Quad-core i7 CPU.

Table 2. Simulation time and real-time factor for different simulations.

Control law	Real-time factor
LQR (without disturbances)	13.7
LQR (with disturbances)	10
LQG (with disturbances)	3.3
LPV (without disturbances)	6
Lyapunov (without disturbances)	5.8

Thanks to the fact that the real-time factor is greater than 1, the animation with FlightGear runs in real time with 30 frames per second.

5 Conclusions

In this work we presented a novel framework for the design and simulation of flight planning and aircraft control systems. The toolbox, named APRICOT, is highly versatile, customizable and provided with user interfaces such as a GUI, a gamepad interface and an external open source animation environment, namely Flightgear. The software is released as an open source MATLAB/Simulink toolbox[2] and can be used for control design and system performance evaluations under different environmental conditions. The entire APRICOT framework is customizable in all its details. For example, control laws can be tested for robustness with respect to external perturbations or for flight tracking purposes. With respect to other software solutions, the main strengths of APRICOT are: the open source nature, the highly configurable control system, the ease of use, the availability of a complete aircraft dynamics and of the environmental model.

Future developments are directed towards the simulation of quadrotors and other Unmanned Aerial Vehicles. With the purpose of extending APRICOT simulation to handle multi-robot systems, the simulation of both aerial and ground vehicles is under study at the present time, in order to allow the design of coordination/mission control systems.

[2] APRICOT Software available at http://aferrarelli.github.io/APRICOT/.

References

1. JSBSim. http://jsbsim.sourceforge.net/
2. jMAVSim. https://github.com/PX4/jMAVSim
3. Ardupilot. http://ardupilot.org/
4. PX4. http://px4.io/
5. Bittar, A., Figuereido, H.V., Guimaraes, P.A., Mendes, A.C.: Guidance software-in-the-loop simulation using X-plane and simulink for UAVs. In: 2014 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 993–1002. IEEE (2014)
6. Jenie, Y.I., Indriyanto, T.: X-plane-simulink simulation of a pitch-holding automatic controlsystem for boeing 747. In: Indonesian-Taiwan Workshop, Bandung, Indonesia (2006)
7. Aircraft control toolbox. Princeton Satellite Systems. http://www.psatellite.com/act/index.php
8. APRICOT. https://github.com/aferrarelli/APRICOT/
9. Heffley, R.K., Jewell, W.F.: Aircraft Handling Qualities Data (1972)
10. Cavcar, M.: The international standard atmosphere (ISA). Anadolu Univ. Turkey **30** (2000)
11. Ondriš, D., Andoga, R.: Aircraft modeling using MATLAB/FlightGear interface. Acta Avionica **15**(27) (2013)
12. Nusyirwan, I.F.: Engineering flight simulator using MATLAB, Python and FlightGear. In: SimTecT, Melbourne, Australia (2011)
13. Moness, M., Mostafa, A.M., Abdel-Fadeel, M.A., Aly, A.I., Al-Shamandy, A.: Automatic control education using FlightGear and MATLAB based virtual lab. In: 8th International Conference on Electrical Engineering, pp. 1157–1160 (2012)
14. Stevens, B.L., Lewis, F.L., Johnson, E.N.: Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems. Wiley, Hoboken (2015)
15. Cook, M.V.: Flight Dynamics Principles: A Linear Systems Approach to Aircraft Stability and Control. Butterworth-Heinemann, Oxford (2012)
16. Tewari, A.: Advanced Control of Aircraft, Spacecraft and Rockets, vol. 37. Wiley, Hoboken (2011)
17. Chrif, L., Kadda, Z.M.: Aircraft control system using LQG and LQR controller with optimal estimation-Kalman filter design. Procedia Eng. **80**, 245–257 (2014)
18. Marcos, A., Balas, G.J.: Development of linear-parameter-varying models for aircraft. J. Guidance Control Dyn. **27**(2), 218–228 (2004)
19. Härkegård, O., Glad, T.: Flight Control Design Using Backstepping. Linköping University Electronic Press, Linköping (2000)
20. Ferrarelli, A., Caporale, D., Settimi, A., Pallottino, L.: Apricot: aerospace prototyping control toolbox. Dynamics and control details (2016). https://github.com/aferrarelli/APRICOT/blob/master/APRICOTExtended.pdf
21. Donald, M.: Automatic Flight Control System. Prentice Hall, Upper Saddle River (1990)
22. Tewari, A.: Automatic Control of Atmospheric and Space Flight Vehicles: Design and Analysis with MATLAB® and Simulink®. Springer Science & Business Media, Heidelberg (2011)
23. Caughey, D.A.: Introduction to aircraft stability and control. In: Lecture Notes. Cornell University (2011)