

Fall 2009

Flight Regime and Maneuver Recognition for Complex Maneuvers

Jerome H. Travert

Embry-Riddle Aeronautical University - Daytona Beach

Follow this and additional works at: <https://commons.erau.edu/db-theses>



Part of the [Aerospace Engineering Commons](#)

Scholarly Commons Citation

Travert, Jerome H., "Flight Regime and Maneuver Recognition for Complex Maneuvers" (2009). *Theses - Daytona Beach*. 200.

<https://commons.erau.edu/db-theses/200>

This thesis is brought to you for free and open access by Embry-Riddle Aeronautical University – Daytona Beach at ERAU Scholarly Commons. It has been accepted for inclusion in the Theses - Daytona Beach collection by an authorized administrator of ERAU Scholarly Commons. For more information, please contact commons@erau.edu.

**FLIGHT REGIME AND MANEUVER RECOGNITION
FOR COMPLEX MANEUVERS**

by

Jérôme H Travert

A Thesis Submitted to the
Graduate Studies Office
In Partial Fulfillment of the Requirements for the
Degree of Master of Science in Aerospace Engineering

Embry-Riddle Aeronautical University
Daytona Beach, Florida
Fall 2009

UMI Number: EP32001

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform EP32001
Copyright 2011 by ProQuest LLC
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Copyright by Jérôme Hubert Pierre Traver, 2009
All Rights Reserved

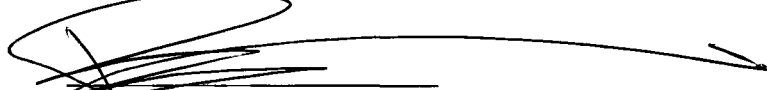
FLIGHT REGIME AND MANEUVER RECOGNITION FOR COMPLEX MANEUVERS

by

Jérôme H Traver

This thesis was prepared under the direction of the candidate's thesis committee chairman, Dr. Richard "Pat" Anderson, Department of Aerospace Engineering, and has been approved by the members of his thesis committee. It was submitted to the Department of Aerospace Engineering and was accepted in partial fulfillment of the requirements for the degree of Master of Science in Aerospace Engineering.

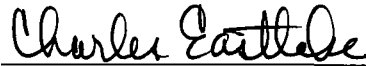
THESIS COMMITTEE:



Dr. Richard "Pat" Anderson
Chairman



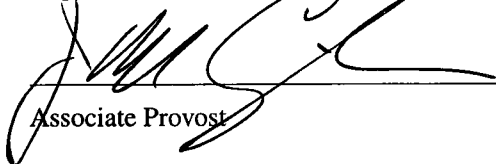
Dr. Bogdan Udrea
Member



Professor Charles Eastlake
Member


Coordinator, MSAE Program
Department Chair, Aerospace Engineering

12/9/09
Date


Associate Provost

12/15/09
Date

ACKNOWLEDGEMENTS

This thesis is a great step in the dual degree program I entered three years ago, and it wouldn't have been possible without the help of many people. It was performed under the supervision of Dr. Richard "Pat" Anderson, whose help and insights throughout this project was of great value and deserves many thanks. In addition, I would like to thank the members of the thesis committee, Dr. Udrea and Professor Eastlake for the interest they show in this research.

Thanks also to the other people who participated directly or indirectly in this thesis, for example Christopher Brown, who set up the flight simulator for me, Mikhael Ponso, who gave me insights about aerobatics and helped me generate data when flying the simulator, and all the people at the Flight Research Center, who helped me when I needed it throughout this project: Brett Mather, Monica Londono, Harshad Lalan, Kashif Ali, Lori Costello and Shirley Koelker.

A more personal thank you to my friends from the ERAU/EPF dual degree program, that brought me here, Florent Lucas, Julien Thivend and Camille Decoust, for helping me out when I first arrived in Daytona two years ago, Steven Armstrong, François Mandonnaud, Noah Becker and Mathieu Naslin, for sharing their interests and friendship all along this program, and the newly arrived students from EPF, Laurence Rebêche and Miléna Perret, for being patient and showing interest when I tell them about engineering and particularly my thesis.

And finally, many thanks to my friends and family in France and in the United States, in particular my parents for never doubting my degree choices and supporting me during my stay in Daytona Beach, my twin brother Christophe, for pushing me into giving the best of me even when an ocean splits us apart, Denise Maurice whose support and love helped me through this degree, and all the others who I cannot exhaustively list, for their friendship.

ABSTRACT

Author: Jérôme H Travert
Title: FLIGHT REGIME AND MANEUVER
RECOGNITION FOR COMPLEX MANEUVERS
Institution: Embry-Riddle Aeronautical University
Degree: Master of Science in Aerospace Engineering
Year: 2009

The purpose of this study is to demonstrate capability of flight regime recognition during complex maneuvers flown in a fixed wing airplane using measured data from an Inertial Measurement Unit (IMU). Flight Regime Recognition (FRR) is required for numerous applications in the aerospace and aviation industry, including the determination of loads for stress and strain analysis. It can also be used in recreational aviation for maneuver recognition, for example in aerobatics.

This study uses a flight simulator to generate representative flight data that is parsed by a specifically developed algorithm into appropriate flight regimes. This algorithm is a filter technique that uses states based on the aircraft's attitude, accelerations and rates and compares them to known trajectories for the identification of specific maneuvers. Particular care has been taken to ensure appropriate noise rejection and tolerance to errors in the realization of the maneuver.

Presented here will be a particularly challenging test case of identification of complex aerobatic, aresti maneuvers, from specific flight trajectory. Results are conclusive in terms of regime recognition but further testing of the maneuver identification algorithms will be necessary in order to derive a robust maneuver recognition program.

TABLE OF CONTENTS

Acknowledgements	iv
Abstract	v
Abstract	v
1 Introduction	1
1.1 Flight Regime Recognition	1
1.2 Pattern Recognition	2
1.3 Competitive Aerobatics	2
2 Scope and Applicability	3
2.1 Problem Statement	3
2.2 Literature Review	4
2.3 Objectives	5
3 Flight Regimes Recognition Algorithm	6
3.1 Regimes Used	6
3.2 Characterization	7
3.3 Flight States	8
3.4 States to Regimes Transition	10
4 Aerobatic Maneuvers Identification	12
4.1 Aresti Notation	12
4.2 Maneuver Representation	14
4.3 Error Correction	15
4.4 Distance Measurement	16
4.5 Identification	18

5	Tests and Results	19
5.1	Data Acquisition	19
5.2	Data Treatment	19
5.3	First Sequence	19
5.3.1	States Observed	21
5.3.2	Maneuvers' Description	22
5.4	Second Sequence	29
6	Conclusion	36
	References	38
	Appendix	40
A	Data Acquisition Model	40
B	Flight Regime Recognition Model	42
C	States Observed During First Sequence	44
D	Source Code	47
D.1	Main Program (Import script)	47
D.2	Catalog Parsing (loadcatalog function)	49
D.3	Flight Decomposition into Legs (getflightlegs function)	54
D.4	Legs Filtering (trim_maneuver function)	56
D.5	Maneuvers Identification (getmaneuvers function)	58
D.6	Mapping (altermaneuver function)	60
D.7	Comparing Maneuvers (comparemaneuvers function)	70
D.8	Plots scripts	71
D.9	Minor Functions	77

LIST OF FIGURES

1	State ($ N_z \approx 1$) as a function of N_z	9
2	Cost of errors in parameter change	16
3	Maneuver identification technique	17
4	Identification example: badly flown cuban	18
5	Data treatment algorithm	20
6	First test sequence	20
7	Regimes during climb	23
8	Regimes during vertical descent	24
9	Regimes during half cuban	25
10	Regimes during loop	26
11	Regimes during turn	27
12	Regimes during roll	28
13	Second test sequence	29
14	Regimes during vertical S	30
15	Regimes during square loop	31
16	Regimes during bow-tie	32
17	Regimes during reverse half cuban	33
18	Regimes during avalanche (loop with roll on top)	34
19	Regimes during Immelmann	35
20	Data acquisition model	41
21	Flight Regime Recognition model - top level	42
22	States determination block	43

23	Implementation of equation 1 in Simulink	43
24	Sequence 1 - Load factor	44
25	Sequence 1 - Pitch angle	44
26	Sequence 1 - Bank angle	45
27	Sequence 1 - Pitch rate	45
28	Sequence 1 - Roll rate	46
29	Sequence 1 - Yaw rate	46

LIST OF TABLES

1	Characterization of flight regimes	8
2	Flight states	9
3	Tolerances used for states determination	11
4	Aresti representation of simple maneuvers	13
5	Errors considered and associated distance	15
6	Sequence of legs detected for the first sequence	22
7	X Plane data export settings	40

1 INTRODUCTION

The need for flight regime recognition is very present in aircraft maintenance and continued airworthiness: knowing the events the aircraft has seen in his lifetime are of great help for maintenance actions, failure prevention and life extension[1][2][3]. It can also be used, as in this study, for a different purpose: telling how maneuvers were flown, which can help a pilot to try improving his maneuvering skills, an instructor tell what his student did wrong, or a competition judge grade an aerobatic program.

1.1 Flight Regime Recognition

Flight regimes are specific conditions under which an aircraft flies. In a standard commercial airplane flight, regimes usually seen include take off, climb, cruise, loiter, descent and landing. Real-time flight regime recognition can allow an autopilot to automatically select its functioning mode when triggered, or to correct for a mistake in mode selection, allow a pilot take measures to avoid regimes that present a danger for the safety of the aircraft, or an instructor help his student improve his flying skills with well defined metrics. Post-processing flight regime recognition can help the technicians maintaining an aircraft to know the solicitations seen by the different parts of the vehicle, and deduce necessary maintenance actions to be taken on those parts[4][5][6].

In the case of maneuver identification, the picture is slightly different since the regime recognition is used for another purpose: sequencing flight legs and aircraft attitude for reconstruction of its flight. This can be done straight from flight data, but the analysis of an aircraft's flight from raw data is a very time consuming process, and requires advanced methods such as neural network[7] in which neither the candidate nor his supervisor had prior knowledge, and was therefore not considered.

1.2 Pattern Recognition

Pattern recognition is the analysis of data, searching for known motives, in our case, for regimes and then specific sequences of regimes flown by an aircraft. Classical pattern recognition algorithms used for example in optical and speech recognition first involve isolation of possible pattern from the rest of the data, sometimes called filtering, then some kind of treatment to see the major features of the studied data, and finally tries to match the figure it has seen to one of several known reference patterns[7][8].

In this study, two pattern recognitions are performed: the first one for regime recognition at each time step, comparing flight data to reference values to find major traits of the flight and classify the aircraft's behavior into flight regimes, and a second one for identification of flight regimes sequences to one of the catalog's maneuvers.

1.3 Competitive Aerobatics

Aerobatics are the practice of flying maneuvers that are not used in normal flight, for entertainment of both the pilot and his public. It explores all dimensions of its flight domain -horizontal plane as well as altitude and aircraft rotations- and often gets close to the aircraft's limitations. Competitive aerobatics is the use of aerobatics skill for competition. Applications of similar practices also include in-flight demonstration, which shows aircraft capabilities, generally for commercial purposes, and combat, where the pilot tries to take advantage on his opponent using his piloting skills and his aircraft's maneuverability.

Aerobic maneuvers are sequences of aircraft attitudes and are regulated and rated for competition. Grades are traditionally given by judges who observe the maneuvers by eyeball, from the ground, with well-defined criteria[9]. This way of judging flight skills suffers from some drawbacks, one of which is being accused of subjectivity. A solution to that can be found by the addition of a computer that helps the judges know how the maneuver was flown, and removes the subjectivity.

2 SCOPE AND APPLICABILITY

Many applications using Flight Regime Recognition have been designed for usage monitoring of aging military aircraft[2][5][6], but FRR for commercial and general aviation aircrafts has been subject of little investigation available in the public domain, including at Embry-Riddle Aeronautical University[1][4] and in the aerospace industry[3], even though it would bring the same improvements in these fields as it would to the military aviation.

2.1 Problem Statement

General aviation and commercial FRR algorithms available in the public domain cover most common regimes for a normal cross country flight, and do not address atypical regimes seen for example in aerobatic and demonstration flights. Such regimes are however especially demanding for aircraft structures and should enter into consideration when monitoring flight loads on aircrafts that see them on a regular basis. The demonstration of capability for Flight Regime Recognition in complex maneuvers is therefore an important step in aircraft usage monitoring.

The case of interest of this study is aerobatic maneuvers, and one application of Flight Regime Recognition is flight legs reconstruction, which is also of interest in this field: being able to determine the sequence of regimes seen by an aircraft allows determining how well the intended program was performed, or identification of maneuvers that were flown. Advanced maneuver identification is of great interest for competitive aerobatics ratings as well as training and for flight instruction in general.

2.2 Literature Review

Flight Regime Recognition has been studied extensively for rotorcraft aging and fatigue analysis for all sorts of helicopters: commercial[3], military[2][5][6] and general aviation[4].

For military aircrafts, the cost of data acquisition and treatment is usually not a limiting factor, and complex data acquisition systems and detailed algorithms have been used. For example the US Army Integrated Mechanical Diagnostic System (IMDS) records 16 parameters for filtering and decomposition into 65 very detailed regimes, sometimes with very small differences between two regimes[2]. This leads to complex and time consuming treatment, that could induce a high cost for both the flight recorder and the post-treatment station. •

On the other hand, general aviation FRR algorithm are more modest and only use limited data channels for regime recognition as well as a reduced set of regimes that aims at qualifying the flight profile rather than quantifying flight loads with great details[1][4], which means large scale application expect a limited cost for the flight recorder as well as the treatment station.

Approaches used are however very similar, using the range in which each parameter is to allow classification of the current flight regime in a specified set. Only the level of detail differs: where military algorithms decompose the flight into dozens of different regimes, general aviation algorithms use less than 10 regimes.

All these studies aimed at helicopter structural monitoring, and very few applications of FRR algorithm in fixed wings aircrafts have been found. A study of general aviation fixed wing aircraft loads was conducted at Embry-Riddle Aeronautical University by David Kim[1]. It used a neural network approach to find a relationship between the flight parameters and the loads seen by different parts of the aircraft for a set of regimes seen in normal operation. Results showed that a classification of the flight parameters into regimes along with the neural network operation provided better results and a Flight Regime Recognition algorithm was developed for that purpose. It used a neural network approach to classify the flight maneuver into 5 different types to allow more accurate prediction of the flight loads.

The results of Kim's study were conclusive for regime recognition in 5 simple maneuvers that were estimated sufficient to cover the range of normal general aviation airplane usage.

2.3 Objectives

This study aims at demonstrating Flight Regime Recognition capability for complex, aerobatic maneuvers that involve unusual flight regimes and transitions, with a minimum data set and a possible real-time implementation. The set of regimes is more evolved than those used for general aviation airplanes as it is not limited to normal cross country operations, but whole of competitive aerobatics flight domain.

This study also intends to identify maneuvers flown, gathering time dependent flight regimes into maneuver legs in order to enable a qualitative evaluation of the flight that could eventually lead to a quantitative grading of the program flown. For that purpose the flight regimes are limited to a reasonably-sized set to allow identification of flight legs from the flight regimes history with just enough details to describe the flown maneuver.

3 FLIGHT REGIMES RECOGNITION ALGORITHM

Aerobic maneuvers are the combination of a flight path and rotations of an aircraft along its pitching and rolling axes. They are combinations of six basic regimes: lines, turns, loops, rolls, spins and tailslides[10]. In power competition, lines can be flown at 5 different angles from the horizontal line: 0° , 45° , 90° , -45° and -90° [10]. A maneuver is a sequence of flight regimes, and determining, at each time step, in which regime the airplane flies is the first step for maneuver identification.

3.1 Flight Regimes Used for Aerobatics

In this study, rolls and spins have been treated differently, as they are elements that superimpose on other regimes, and not regimes by themselves. Spins have been neglected as they only bring complexity to the recognition algorithm, and can be added once a good understanding of maneuver identification is achieved. This gives us a total of eight basic regimes:

- **Level flight:** 0° straight line
- **Turn:** heading change at high bank angle, aircraft stays on a horizontal plane
- **Climb:** $+45^\circ$ straight line
- **Descent:** -45° straight line
- **Vertical Climb:** $+90^\circ$ straight line
- **Vertical Descent:** -90° straight line
- **Loop:** progressive change in flight path angle, aircraft stays on a vertical plane
- **Tailslide:** airplane goes down tail first

All these regimes can be flown with addition of rolls, and can also be flown in two different ways: with positive or negative normal load factor (except for vertical lines, which are supposedly flown with a normal load factor of 0), which gives us two additional flags that superimpose on the regimes: a roll flag, as well as a “negative load factor” or inverted flag.

3.2 Flight Regimes Characterization

Flight regimes were studied to find relevant parameters for their characterization. Parameters were added until each regime would be a unique combination of flight data. Table 1 summarizes regimes characteristics.

Pitch rate (q) is the first parameters to distinguish lines from loops: a $0^\circ/s$ pitch rate means the aircraft's flight path is close to a straight line, a different pitch rate means the aircraft is either flying in a loop or a turn.

Pitch angle (θ) allows distinguishing lines from each other. This would preferably be done using flight path angle (except for Diagonal Lines, where the criterion is attitude and not flight path), but simpler measurement made pitch angle the variable of choice. The difference between flight path and pitch angle is to be accounted for by higher tolerances in detection of flight parameters. For Horizontal lines, it was considered redundant with the load factor criterion.

Yaw rate (r) is used to tell loops apart from turns: the turn is the only regime where a yaw rate should be present: a non-zero yaw rate means the airplane is in a turn, a non-zero pitch rate with no yaw rate means it is in a loop.

Airspeed is also measured for Tailslide detection: it is the only regime flown at a negative airspeed. In the algorithm presented in this study, True Airspeed (TAS) was used, but since it is only for sign discrimination, standard measurement of airspeed (Indicated Airspeed) or even axial speed of the aircraft (u , from an IMU) should be acceptable.

Roll rate (p) characterizes rolls.

Load factor (N_z), more accurately normal load factor, is required for the inverted flag, which is to be active when the load factor is negative. For aerobatics rating, vertical and horizontal lines have criterion in terms of load factors, which are also taken into account.

Bank angle (ϕ) is also taken into account as a redundancy check for ensuring a turn is performed accordingly to grading criterion which states "Turns should be flown at bank angles of at least 60° "[9].

Regime	TAS	N_z (g)	p (°/s)	q (°/s)	r (°/s)	ϕ (°)	θ (°)
Level Flight	> 0	$\approx \pm 1$	0	0	0	0	≈ 0
Turn	> 0	X	0	$\neq 0$	$\neq 0$	> 60	≈ 0
Climb	> 0	X	0	0	0	0	≈ 45
Descent	> 0	X	0	0	0	0	≈ -45
Vertical Climb	> 0	≈ 0	0	0	0	X	≈ 90
Vertical Descent	> 0	≈ 0	0	0	0	X	≈ -90
Loop	> 0	X	0	$\neq 0$	0	0	X
Tailslide	< 0	X	0	X	X	X	X
Roll	X	X	$\neq 0$	X	X	X	X
Inverted	X	< 0	X	X	X	X	X

Table 1: Characterization of flight regimes
X representing a non-specific value.

3.3 Flight States

Flight states are used for identification of a flight regime. They tell whether a flight parameter is equal (respectively different) to a reference value, with a certain tolerance to account for approximations and noise. Each states activity is a value taken between 0 and 1 depending on how close the parameter is to the reference value, 0 meaning completely off (respectively close) and 1 very close (respectively very different). States are an intermediate step between flight data and regime recognition, derived from table 1 values of flight parameters, and listed in table 2.

The states that tell whether a parameter x is close to a reference value x_{ref} or not are computed using a function f of the distance $\Delta x = x - x_{ref}$ given in equation 1, where K is computed to satisfy the third criterion: $K = \frac{\ln(2)}{\text{tolerance}^4}$, and plotted for the ($|N_z| \approx 1$) state in figure 1. This function was designed for the following characteristics:

- $f(\Delta x) \approx 1$ when $\Delta x < \frac{1}{2} \times \text{tolerance}$
- $f(\Delta x) = 0$ when $\Delta x > 2 \times \text{tolerance}$
- $f(\Delta x) = 0.5$ when $\Delta x = \text{tolerance}$

$$f(\Delta x) = e^{-K \times \Delta x^4} \quad (1)$$

For states that tell whether the parameter is far from a reference value (all $\neq 0$ states), the function defined in equation 1 is subtracted from 1 to get the states activity: state = $1 - f(\Delta x)$.

State	Application
$ N_z \approx 1$	Level Flight characterization
$N_z \neq 0$	Vertical Lines elimination
$N_z < 0$	Inverted status
$\theta \approx 45^\circ$ $\theta \approx 90^\circ$ $\theta \approx -45^\circ$ $\theta \approx -90^\circ$	Lines discrimination
TAS < 0	Tailslide characterization All other regimes elimination
$p \neq 0$	Roll detection Level and Loop elimination
$q \neq 0$	Loop characterization Lines elimination
$r \neq 0$	Turn characterization Loop elimination
$\phi \neq 0^\circ$ or 180°	Turn elimination

Table 2: Flight states used for regime identification

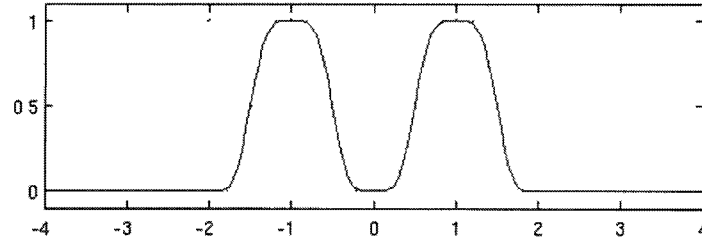


Figure 1: State $|N_z| \approx 1$ as a function of N_z (tolerance=0.5)

This is used for computation of all states except $N_z < 0$ and TAS < 0, which are treated individually, because of their particular aspect. The airspeed state, TAS < 0 is the only non continuous state, as it is very unlikely that an airspeed close to 0 is maintained in a non-transient way. It is defined as the logical result of comparing TAS to 0. The inverted state, however requires a continuous definition since a 0g load factor often happens, and close to 0 but positive values could be seen in inverted legs. The definition chosen in this case is a multi-linear function defined as: 1 when $N_z \leq 0$, 0 when $N_z \geq 0.5$ and the linear interpolation $2 \times (0.5 - N_z)$ in between.

3.4 States to Regimes Transition

The states to regime transition is performed using an adaptation of table 1 to the states, arranged in a vector for easier manipulation. The probability of being in each regime is deducted from the applicable states. It is computed for all regimes using matrix multiplication: $\{\text{regimes}\} = [H] \times \{\text{states}\}$ where the states are ordered as in table 2, the regimes as in table 1, and H is an adaptation of table 1, tuned for correct regimes detection throughout all test flights:

$$H = \begin{matrix} & \begin{matrix} N_z \approx 1 \\ N_z \neq 0 \\ N_z < 0 \\ \theta = 45^\circ \\ \theta = 90^\circ \\ \theta = -45^\circ \\ \theta = -90^\circ \\ \text{TAS} < 0 \\ p \neq 0 \\ q \neq 0 \\ r \neq 0 \\ \phi \neq 0^\circ \end{matrix} & \begin{bmatrix} 1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -0.7 & -0.7 & -0.5 & 0 \\ 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -0.7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -0.7 & 0 & 0 \\ 0 & -0.4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -0.7 & 0 & 0 \\ 0 & -0.4 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -0.7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -0.7 & 1 & -0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} & \begin{matrix} \text{Level Flight} \\ \text{Turn} \\ \text{Climb} \\ \text{Descent} \\ \text{V-Climb} \\ \text{V-Descent} \\ \text{Loop} \\ \text{Tailslide} \\ \text{Roll} \\ \text{Inverted} \end{matrix} \end{matrix}$$

This whole process is the flight regime recognition algorithm used in this study, and gives conclusive results when proper tolerances are set in the states determinations process. It is to be expected that those tolerances can vary from one aircraft to another, or from a pilot to another: even though judging criteria don't vary, it is important to make sure the flight states are in agreement with what was intended by the pilot. For instance, the initial tolerance on pitch rate ($8^\circ/s$), adapted to Dr. Anderson's flights has proven to be too high for Mikhael Ponso's flights, who is pulling his loops with a lower pitch rate (around $6^\circ/s$).

A possible way of accounting for those differences is allowing calibration of tolerances

States	Tolerance		K value	
$N_z \approx 1$ $N_z \neq 0$	0.5		11.09	
$\theta = 45^\circ$ $\theta = -45^\circ$	10°		7×10^{-5}	
$\theta = 90^\circ$ $\theta = -90^\circ$	20°		4.33×10^{-6}	
$p \neq 0$	40°/s		2.7×10^{-7}	
$\phi \neq 0$	15°/s		1.4×10^{-5}	
	Dr. Anderson	M. Ponso	Dr. Anderson	M. Ponso
$q \neq 0$	8°/s	5°/s	1.1×10^{-3}	1.7×10^{-3}
$r \neq 0$	5°/s	10°/s	1.7×10^{-4}	7×10^{-5}

Table 3: Tolerances used for states determination

by flying a sustained leg in each regime prior to starting the aerobatics sequence. The tolerances used for each pilot are shown in table 3 and were determined by tuning for proper regime recognition as well as for respecting aerobic ratings criteria. For example, the tolerance on ($\theta = 90^\circ$) is high since the judging criteria is not based on pitch angle but vertical flight path, which is seen on IMU data by a N_z value of 0. The FRR algorithm requires knowledge of θ so it does not consider vertical climb if the pitch angle is small, even if the load factor matches the requirement ($N_z \approx 0$). For this particular regime, the considerations are inverted because of initial choice, but similar results are expected using either $a(\theta \approx 90^\circ) - b(N_z \neq 0)$ or $a'(\theta \approx 90^\circ) + b'(N_z \approx 0)$, with tolerances and coefficients adapted to each case.

4 AEROBATIC MANEUVERS IDENTIFICATION

A standard pattern recognition algorithm was used for maneuver identification from flight regime evolution. It is done by 2 important steps: the organization of regimes into “words” that we call maneuvers and the identification of the maneuver seen to one of the known maneuvers. It requires knowledge of possible maneuvers to which the reconstruction from flight data will be compared for identification.

Aerobatic maneuvers are referenced for competition in a catalog named the Aresti catalog, after the Spanish aviator José Luis de Aresti Aguirre, its first designer. The Fédération Aéronautique Internationale (FAI) Aresti Aerobatic Catalog, version 2003-1 was used as list of reference maneuvers in this study.

4.1 Aresti Notation

Before exploring the details of maneuver identification, a quick description of Aresti’s notation of aerobatic maneuvers is probably necessary. This notation consists of a graphical representation of the trajectory of the center of gravity of the airplane, usually in a vertical plane that contains it (except for turns), with rolls superimposed on the trajectory, as illustrated in table 4. A few more rules are useful to understand this notation:

Maneuvers start and finish in level flight (horizontal line).

Entry in the maneuver is represented by a dot while its exit is represented by a cross-line.

Lines can only be inclined by a multiple of 45° from the horizontal line.

Legs flown with positive angle of attack are represented with a solid line while a dashed line represents portions of the flight where the angle of attack is negative.

Angles will replace circular arcs of less than 180° to make the visualization simpler.

Some maneuvers (turns and rolling turns) consist in out of the plane motion that requires switching the representation from the vertical to a horizontal plane.

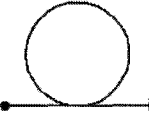

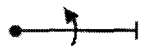
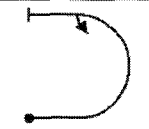

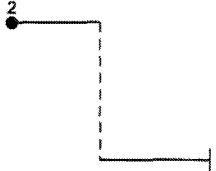
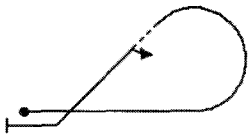
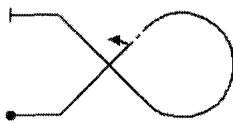
Figure	Representation	Description (leg by leg)
Loop		Pull 360°
Turn		Turn 180° at constant altitude
Roll		Roll 360° at constant altitude
Immelmann		Pull 180° Roll 180°
Climb		Pull to 45° pitch angle Maintain pitch angle Push back to level flight
Dive		Push to vertical descent Maintain vertical flight path Pull back to level flight
Half Cuban		Pull 225° Maintain pitch angle Roll 180° Maintain pitch angle Pull back to level flight
Goldfish		Pull to 45° pitch Maintain pitch angle Roll 180° Maintain pitch angle Pull 270° to 45° pitch Maintain pitch angle Push back to level flight

Table 4: Aresti representation of simple maneuvers

4.2 Maneuver Representation

Maneuvers (analogically referred to as “words”) are combinations of legs (“letters”) that are characterized by several things:

- Regime of the leg, one of the 8 basic regimes (see table 1)
- Length of the leg flown, measured in terms of a parameter relevant to the given regime:
 - horizontal length for Level Flight
 - altitude change for Lines (except horizontal) and Tailslides
 - pitch angle change for Loops
 - heading change for Turns
- Roll status: whether a roll is flown during the leg (and its length)
- Inverted status: whether the status was flown with positive or negative load factor

Changes in flight regime throughout the flight are detected by gathering consecutive data points that share a common dominant regime, and give a decomposition of the flight into a sequence of legs, forming the “sentence” that describes the flight. Legs which have a small parameter change are removed to get rid of the transient regimes. Level Flight legs with no rolls and that are at least 100ft long are used as separations that allow breaking the sequence of legs into several maneuvers, analogically, spaces that allow separating words in a sentence.

4.3 Error Correction

Each flown maneuver is then compared to each of the reference maneuvers to know which is the closest one, and identify it, to give it a name and so on. Possible errors in the realization of a maneuver or data treatment are considered to allow better recognition and matching of the maneuvers. Errors considered are described here:

Leg alteration: Measurement of legs' length is not perfect and it is possible that the distance measured does not match the reference distance. To account for that, an error of 20° in a loop's length has a small impact on recognition, and bigger errors are also allowed, but have higher cost in terms of proximity to the reference maneuver.

Leg addition: It is possible that an additional leg is seen during the maneuver, the most obvious example being a loop with little hesitation that lead to the addition of a line in the middle of two parts of the same loop.

Leg suppression: A leg in a maneuver could not be seen when trying to match a maneuver to its reference version, for example a line between two sections of a loop could be too short to be seen by the algorithm, resulting in a continuous loop leg instead of 2 loop legs separated by a line leg.

Leg replacement: In case a regime is flown inadequately from the algorithm standpoint, it is important to consider the possibility of replacing a leg with a slightly different one, for example the angle of a line could appear to be different from the reference one, especially vertical lines could be seen diagonal because of the offset between actual criterion (based on flight path) and the one used here (based on pitch angle).

Each error has a cost in terms of proximity to the reference maneuvers, which are given by table 5.

Error	Gravity	Cost
Leg alteration	Variable	$d(\Delta p)$
Leg addition	Moderate	$20 + d(\Delta p)$
Leg suppression	Important	$30 + d(\Delta p)$
Leg replacement	Important	30

Table 5: Errors considered and associated distance

A function of the error in parameter change is considered in most of these distance definitions since suppressing a 45° loop should not have the same impact on recognition as

suppressing a 180° loop. Similarly, when adding a line in the middle of a loop, a 50ft line should not have the same impact as a 400ft one. This function is described by equation 2, plotted in figure 2, and responds to the following criteria:

- $d(\Delta p) \approx 0$ when $\Delta p < 10$
- $d(\Delta p) = 20$ when $\Delta p = 20$ (corresponds to a small error)
- $d(\Delta p) \approx 2\Delta p$ when $\Delta p > 30$

$$d(\Delta p) = 2 \left(1 - e^{-\ln(2) \times \left(\frac{\Delta p}{20}\right)^4} \right) \times \Delta p \quad (2)$$

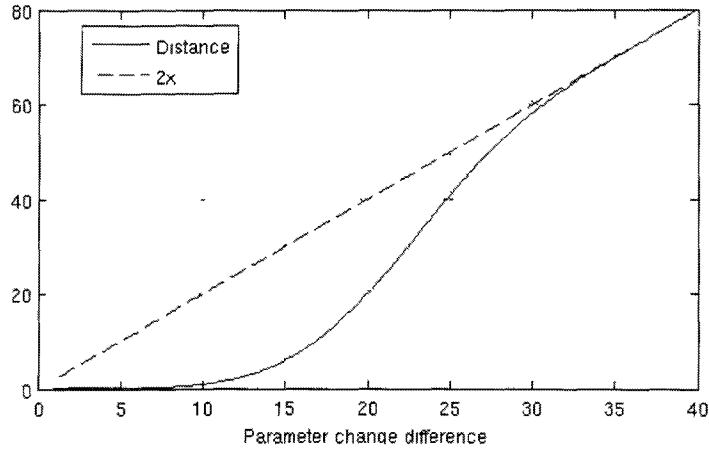


Figure 2: Cost of errors in parameter change

4.4 Distance Measurement

To determine which reference maneuver is the closest to the one that was flown, one needs the distance between the flown maneuver and each of the reference. The distance between two maneuvers is determined from errors, to go from the flown maneuver to the reference we are comparing it to. The sum of the cost of each error (defined in table 5) to go from the reference maneuver to the flown one gives a distance that is used to determine whether the maneuver can be matched to the reference.

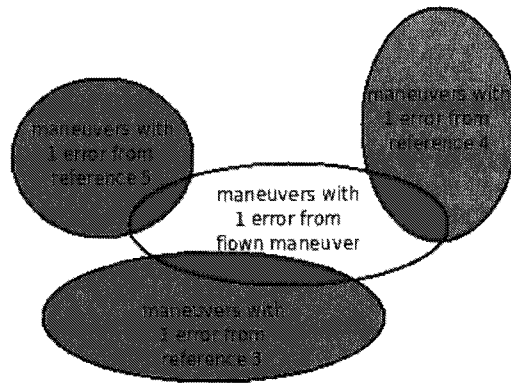


Figure 3: Accounting for errors when identifying maneuver

Determination of the errors path to go from a maneuver to another has no direct method, and only browsing all possible paths starting at the reference maneuver to check when the flown maneuver is attained has been found possible. This gives particularly good results if the paths starting at each reference maneuver are kept in memory between two runs of the recognition algorithm, in some sort of a map, since the mapping is a very time-consuming process, especially when mapping around hundreds of reference maneuvers. However for memory reasons the length of explored paths has to be limited to a small value, as the size of the map increases exponentially with the length of explored paths.

This yields a problem in terms of number of errors that can be considered. A good solution to allow slightly longer paths, which has been adopted in this study, is to generate small maps around each reference maneuver once and for all, which can take some time but reasonable memory, then, when processing a maneuver, map the region around the maneuver to identify, which takes little time (there is only one region to map), and check for common points between this region and each initial map. This method is schematically represented in figure 3, and can be summarized as follows:

Memorize maps around each reference maneuvers.

Generate a map around the flown maneuver.

Look for common points, and sum the distances to get flown-to-reference distance.

4.5 Identification

After comparison of the flown maneuver with each of the reference maneuvers, a decision as to which one is the best match has to be done. For this study, the choice algorithm is very simple: the reference that shows shortest distance with the flown maneuver is selected. Other criteria are of course possible, especially if there is a prior knowledge of the flight program, as it would be the case when judging aerobatics, and this would probably lead to a different distance definitions, with more detailed error scale, and a precise cost for each error that would match International Aerobatics Club's (IAC) judging criteria.

Another possible method, if there is only a partial knowledge of the flown program is to weigh the reference maneuvers with the probability of this maneuver being in the program. For instance, Half Cubans are very common in demonstration aerobatics and would be weighed more than Goldfishes, which are more rarely seen. For example a badly-flown Half Cuban which looks like the one in figure 4 could be considered a badly-flown Reverse Goldfish (the maneuver obtained by flying a goldfish's legs backwards, represented in figure 4), the Half Cuban could still be recognized over the Goldfish with this weighing process.

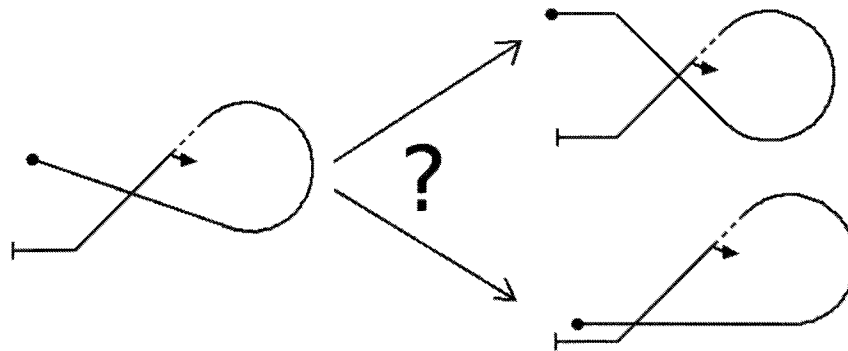


Figure 4: Identification example: badly flown cuban

5 TESTS AND RESULTS

The developed algorithm was tested as a post processing algorithm on two test sequences that were flown on X-Plane Flight Simulator for data acquisition. The regime recognition algorithm, being composed of filters, can be used for real time regime recognition.

5.1 Data Acquisition

The two flight tests were flown by Dr. Anderson and Mikhael Ponso on X-Plane Flight Simulator, and the data acquisition was performed using a Simulink model that reads data packets from X-Plane in real time. This model was developed by Embry-Riddle Aeronautical University's Flight Research Center for research on Helicopter Health and Usage Monitoring Systems, and used for similar purposes in this research program. It is described in appendix A.

5.2 Data Treatment

The Flight Regime Recognition algorithm was implemented using Simulink to enable real time identification and is described by appendix B. The Maneuver Identification algorithm was implemented in Matlab, treating data output by the Simulink model. The whole process is summarized in figure 5.

5.3 First Sequence

The first test sequence, represented in aresti notation in figure 6, was designed for testing and tuning of both algorithm, and includes a long leg of most regimes, as well as simple maneuvers that allow simple recognition. It consists in the following maneuvers: Climb, Vertical descent, Half cuban eight, Full loop (360°), 180° turn, Full roll (360°) in level flight. It was flown twice by Dr. Anderson at the beginning of the work on this study. Results for the second run of this flight are shown in great details here, they are very similar to those obtained with the first set of data.

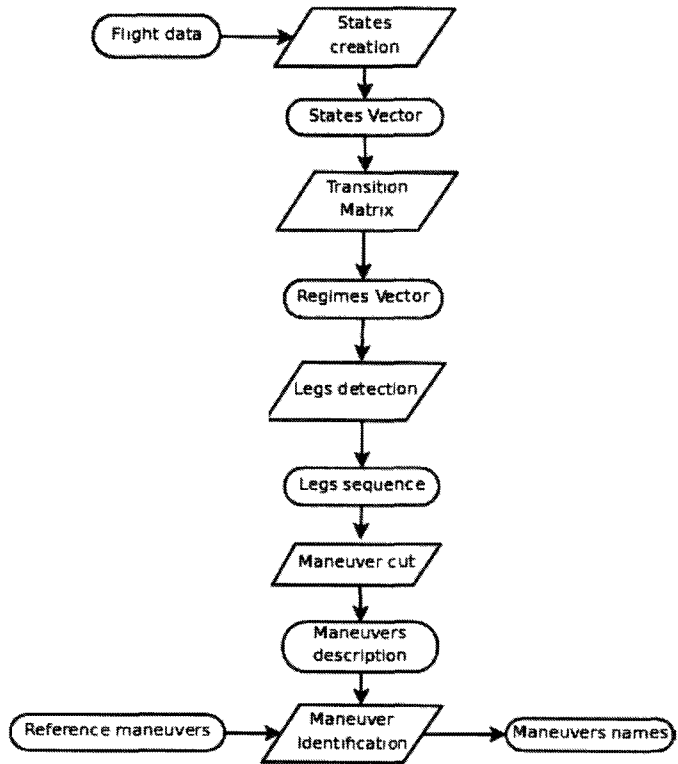


Figure 5: Data treatment algorithm

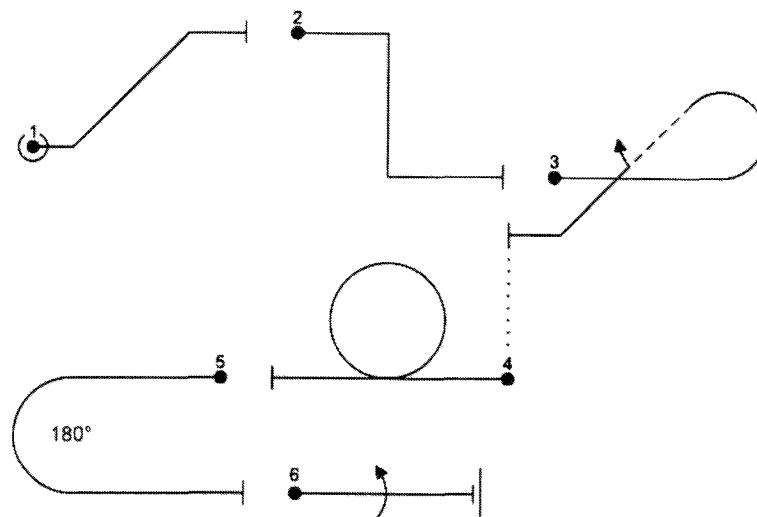


Figure 6: First test sequence in Aresti notation

Tuning of the tolerances as well as the recognition algorithm was mostly done on this sequence, for both runs, even though , and the results in terms of states and regimes are conclusive for both regime recognition and maneuver identification.

5.3.1 States Observed

First step of the process is the determination of states throughout the flight, and results are given in figures 24 through 29 in appendix C. Under this form, they do not give much information, but they allow detection of problems on states determination as well as problems that could occur later, in regimes recognition and legs detection.

The states are the result of tuning the model's tolerances on each parameter and determination of these tolerances is a process that requires prior knowledge of the flight: by knowing what leg was flown at a given time, we know which states should be present, and therefore we can tune the tolerance so the proper states are active. Of course the tolerances are constant throughout the flight so this tuning process is done only once, and then checked for the whole flight. Automatic tuning could be possible provided a given sequence of legs that the pilot would fly prior to the actual program to adapt the tolerances to his aircraft and his way of flying. The Flight Regimes were computed throughout the flight using the transition matrix and are shown in figures 7 to 12. They seem to match the intended regimes for most of the flight, and transitions between regimes appear to be measured properly.

5.3.2 Maneuvers' Description

The regimes evolutions are of great importance, and will determine how the maneuvers can be recognized. In figures 7 to 12, the maneuvers have already been decomposed into legs, which are used for readability of the figures, and given in table 6.


Length	Regime	Flags
62°	loop	
433 ft	climb	
44°	loop (inverted)	
211 ft	level flight	
87°	loop	inverted
292 ft	vertical descent	inverted
93°	loop	
2148 ft	level flight	
222°	loop	
462 ft	descent	half roll from inverted
45°	loop	
800 ft	level flight	
349°	loop	
425 ft	level flight	
321°	turn	
595 ft	level flight	
	level flight	half roll to inverted

Table 6: Sequence of legs detected for the first sequence

This long sequence of legs has been decomposed into maneuvers using the 100 ft long level flight criteria, parsed through the maneuver identification algorithm and the results are detailed in the next pages.

Climb:

Aresti number 1.3.1

Aresti notation 

Description from flight data
 62° in a loop
 433 ft climb
 44° in a loop (inverted)

Identified as 1.3.1, at distance 20

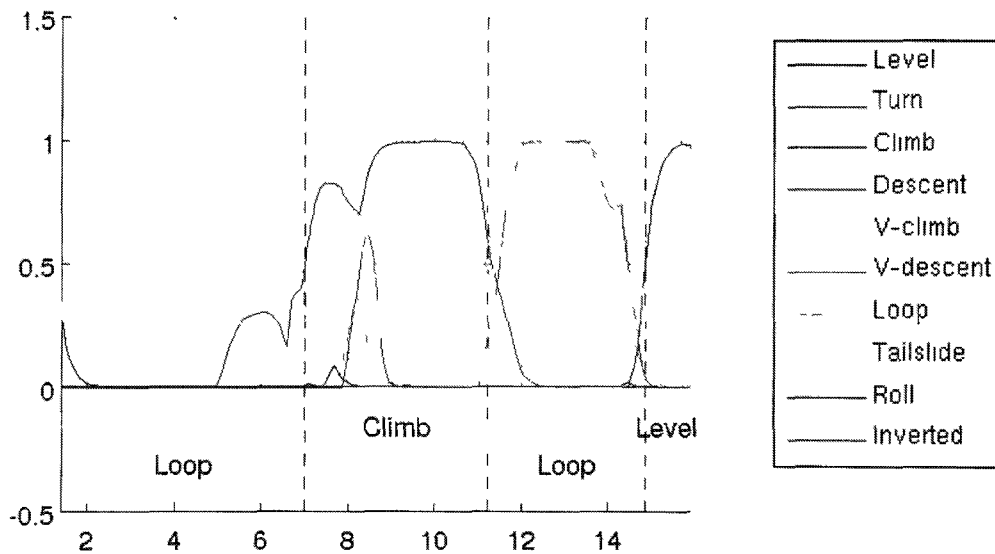
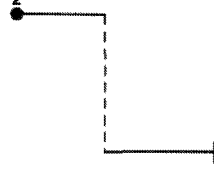


Figure 7: Regimes during climb

This maneuver demonstrates the capability of recognizing the Climb regime and short Loop legs. The maneuver identification algorithm has no problem identifying it as the sequence of legs respects the catalog's description of the maneuver.

Dive:

Aresti number	1.6.3
	2
Aresti notation	
Description from flight data	87° in a loop (inverted) 292 ft vertical descent 93° in a loop
Identified as	1.6.1, at distance 0

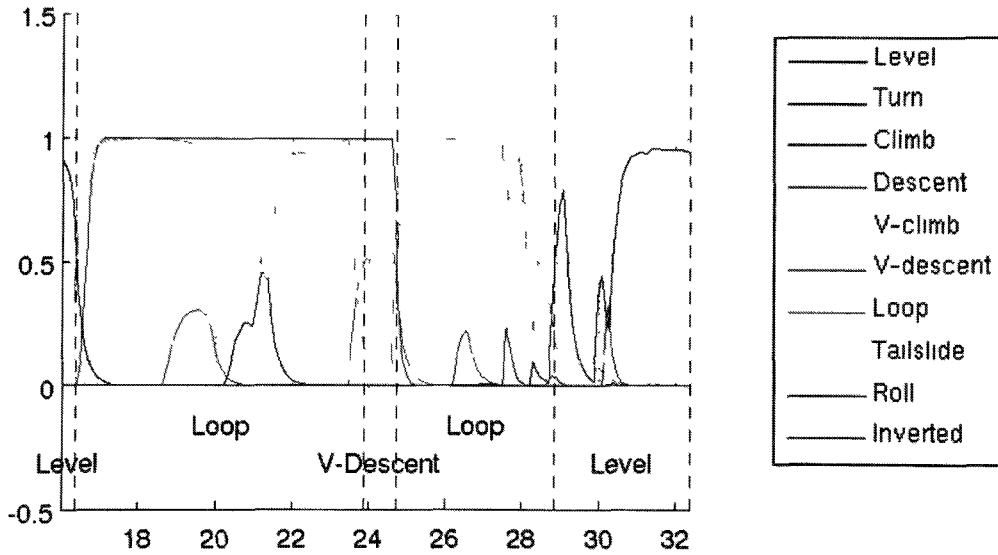
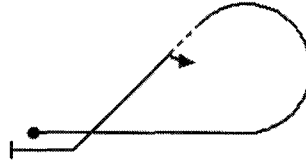


Figure 8: Regimes during vertical descent

This maneuver shows the capability of recognition of the Vertical descent regime as well as measurement of the length of Loop legs. Once again, the maneuver identification program gives good results as the legs sequence match what they are supposed to.

Half Cuban:

Aresti number 8.42.1



Aresti notation

Description from flight data 222° in a loop
462 ft descent with half roll from inverted
45° in a loop
Identified as 8.42.1, at distance 0

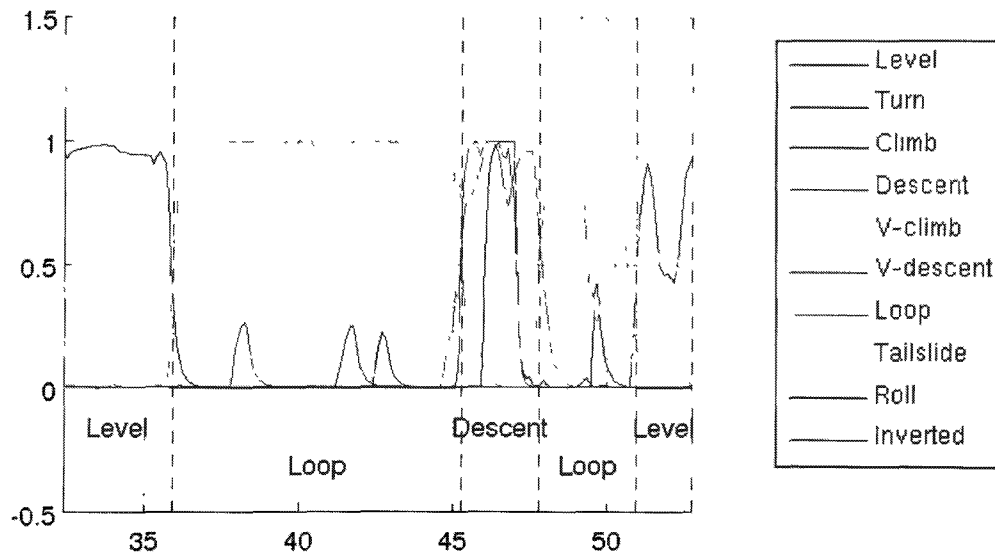
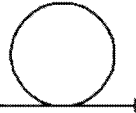


Figure 9: Regimes during half cuban

The half cuban was mostly used to check the behavior of the legs detection algorithm in presence of a roll, which is not a regime by itself, but adds on to a given regime. In this case, the performance was very good and the detected maneuver matched the description of a cuban, resulting in a good identification.

Loop:

Aresti number 7.5.1



Aresti notation

Description from flight data 349° in a loop
Identified as 7.5.1, at distance 1

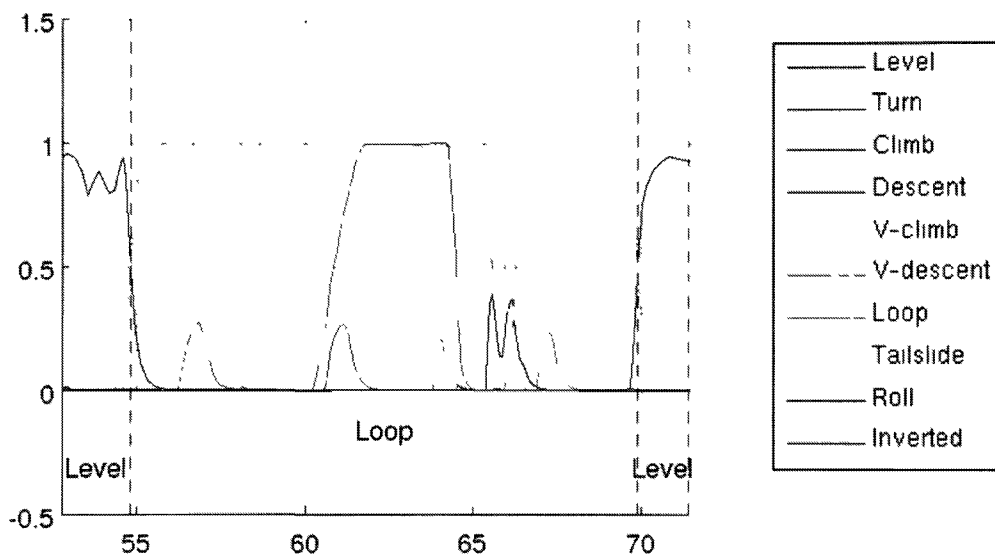



Figure 10: Regimes during loop

Another very simple maneuver, which demonstrates capability of identification of a long loop leg, and acceptance of slight errors in length for identification.

Turn:

Aresti number 2.2.1
Aresti notation 
Description from flight data 321° in a turn
Identified as 2.1.1, at distance 77 (360° turn instead of 180°)

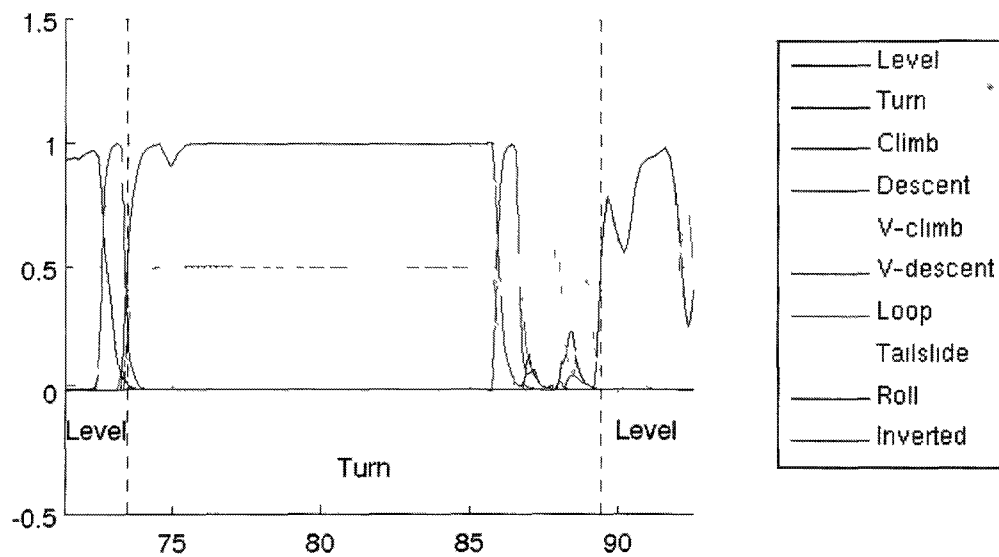



Figure 11: Regimes during turn

This maneuver suffers from a flight error, and the identification algorithm performed very well in detecting this error, showing the maneuver flown doesn't match the one that was intended, since the heading change is too important, and this shows as well in the raw flight data. The prolongation of the Turn leg when no new regime is dominant is normal and this maneuver shows in a very good way the behavior of algorithm when no regime is really dominant. This may be a problem in terms of Flight Regime Recognition, but as far as leg recognition, it is the easiest way to do it: the algorithm should be able to describe all legs.

Roll:

Aresti number 1.1.1

Aresti notation 

Description from flight data 595 ft level flight with half roll to inverted
Identified as 1.2.1, at distance 0

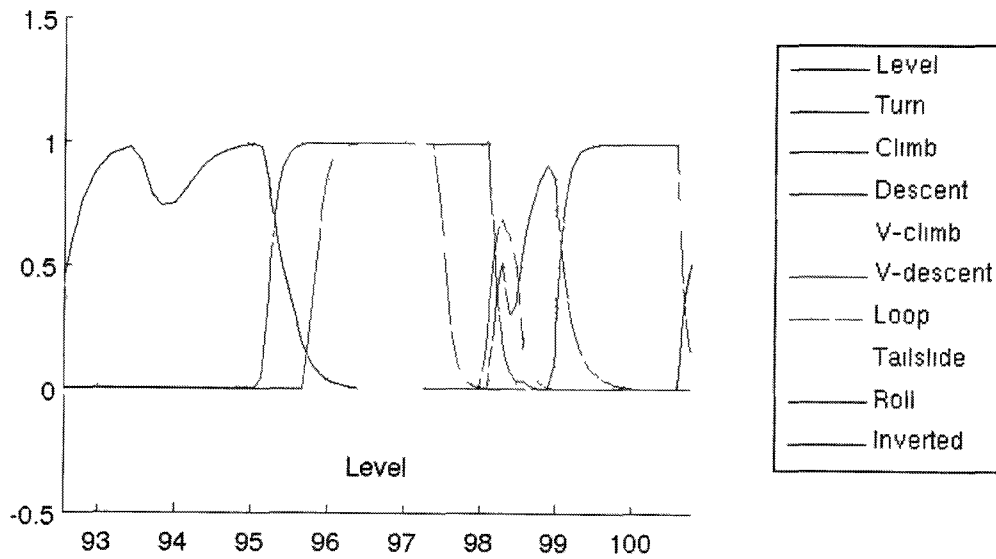


Figure 12: Regimes during roll

This maneuver was flown too close to the ground and it shows a little error in flight since the flight simulator had trouble when the wings touched the ground, and started to send erroneous data. The level flight with roll is however correctly identified except for the length of the roll, which really is 270° instead of 360°.

5.4 Second Sequence

Once the algorithms were set for the first sequence, which consisted essentially of simple maneuvers, a second sequence was necessary to test them and see how they behave on more complex maneuvers. The Flight Regime Recognition algorithm seemed to already have good results but several cases still needed testing for the maneuver identification program, including roll on enter, exit or top of a loop, inverted exit and entry in a maneuver, level flight in the middle of a maneuver, and so on.

A second flight sequence was then designed to address many of those cases and check whether the algorithm still gave good results. This sequence, depicted in figure 13, was flown by Mikhael Ponso. The FRR algorithm performed very well on this second sequence, provided that the tolerances were reviewed to accommodate M. Ponso's flying. The maneuver recognition, on the other hand, required modifications to take specific configurations into account, in particular the rolls-loops combinations. This has been performed by changing details in the way legs are detected. Another problem appears in maneuver separation: maneuvers that use level flight as one of their legs are separated into several maneuvers. The length of level flight used to separate maneuvers seems to be too short, but it was also seen that a longer distance would not enable successful separation of maneuvers for the first sequence.

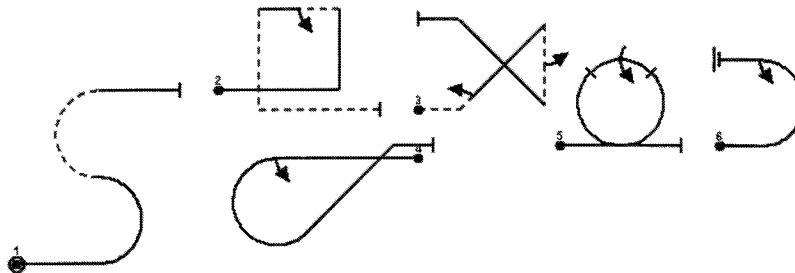
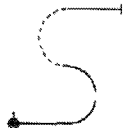



Figure 13: Second test sequence in Aresti notation

Vertical S:

Aresti number 7.11.1



Aresti notation 

Description from flight data 181° in a loop
76 ft in level flight (inverted)
76 ft in level flight (inverted)
185° in a loop (inverted)

Identified as 7.1.1 + 7.1.2 (seen as two maneuvers)

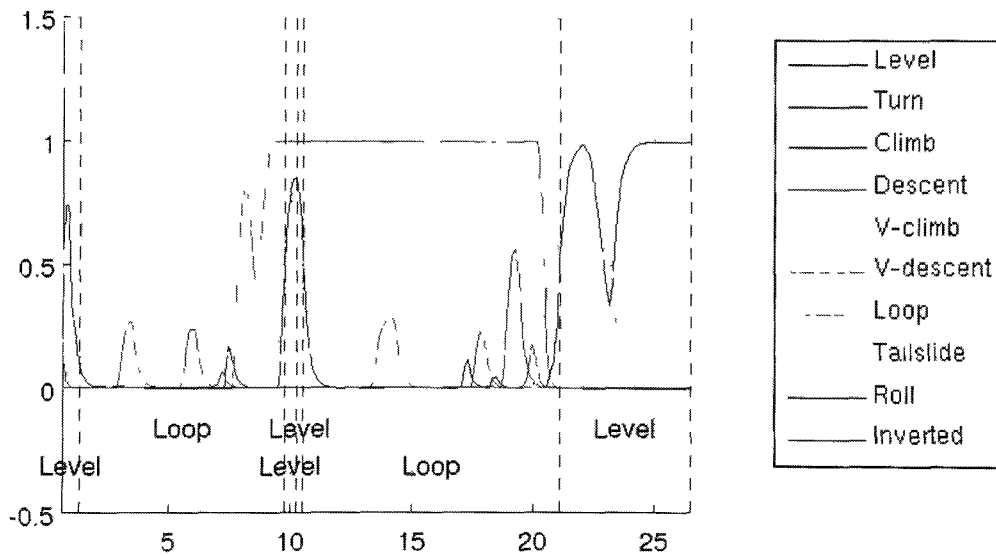
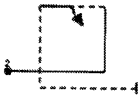


Figure 14: Regimes during vertical S

This maneuver combines errors from flight and treatment, as it should not have a long level flight leg in the middle, which can be seen in the raw flight data, thus results from the way it was flown. This level flight is detected by the legs detection process, and mistakenly considered as a maneuver separation by the maneuver identification algorithm.

Square loop:

Aresti number 7.8.1

Aresti notation 

Description from flight data

- 153° in a loop
- 81 ft in level flight (inverted)
- 333 ft in level flight with half roll from inverted
- 76° in a loop (inverted)
- 92 ft vertical descent (inverted)
- 87° in a loop (inverted)
- 342 ft in level flight (inverted)

Identified as 7.1.1 + 1.1.4 + 1.7.3 (seen as three maneuvers, one of which is badly flown)

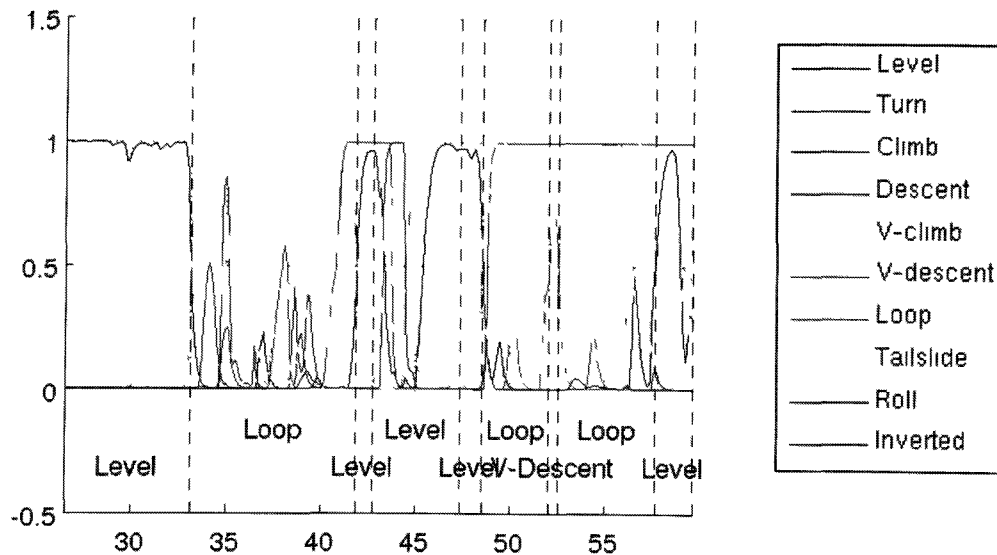



Figure 15: Regimes during square loop

This maneuver has a small error from flight, with no vertical line marked on the first leg, resulting in a 180° loop seen instead of a 90° loop followed by a vertical climb an another 90° loop. But it mostly has a problem in the separation of maneuvers, which creates three maneuvers out of a single one because of the level flight leg.

Bow-tie:

Aresti number 1.33.2

Aresti notation 

Description from flight data

- 342 ft in level flight (inverted)
- 34° in a loop (inverted)
- 705 ft climb with half roll from inverted
- 103° in a loop (inverted)
- 362 ft vertical descent (inverted) with half roll
- 122° in a loop
- 359 ft climb
- 38° in a loop (inverted)

Identified as 1.33.2, at distance 68

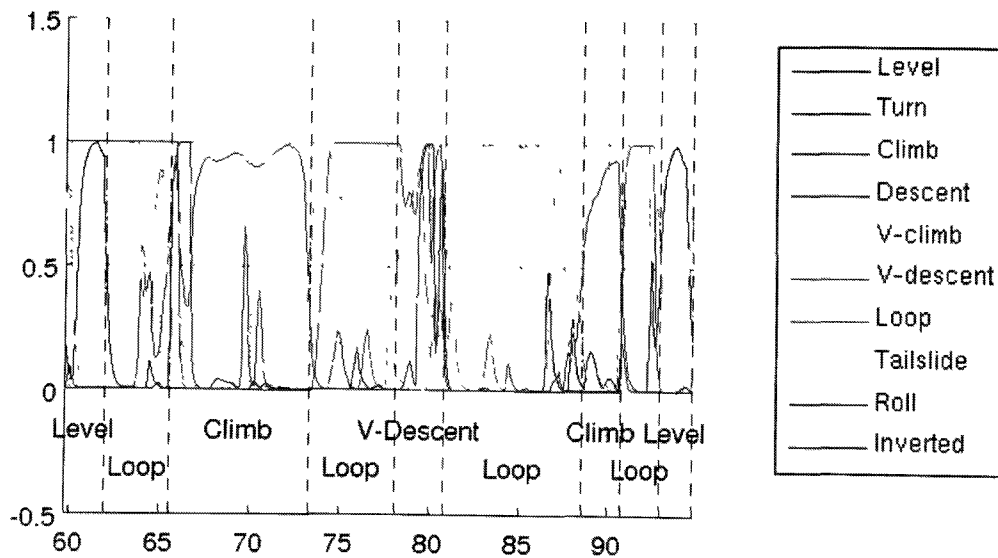
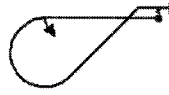


Figure 16: Regimes during bow-tie

This maneuver was mostly designed for demonstration of inverted leg between maneuvers, and this is correctly detected. The bow-tie is an example of a long maneuver, in terms of number of legs, and this allowed testing the mapping algorithm's computation time on a long maneuver, which gave good results even though the process is time consuming.

Reverse Half Cuban:

Aresti number 8.47.3



Aresti notation

Description from flight data 188° in a loop
 372 ft climb
 31° in a loop (inverted)
 Identified as 8.47.3. at distance 108

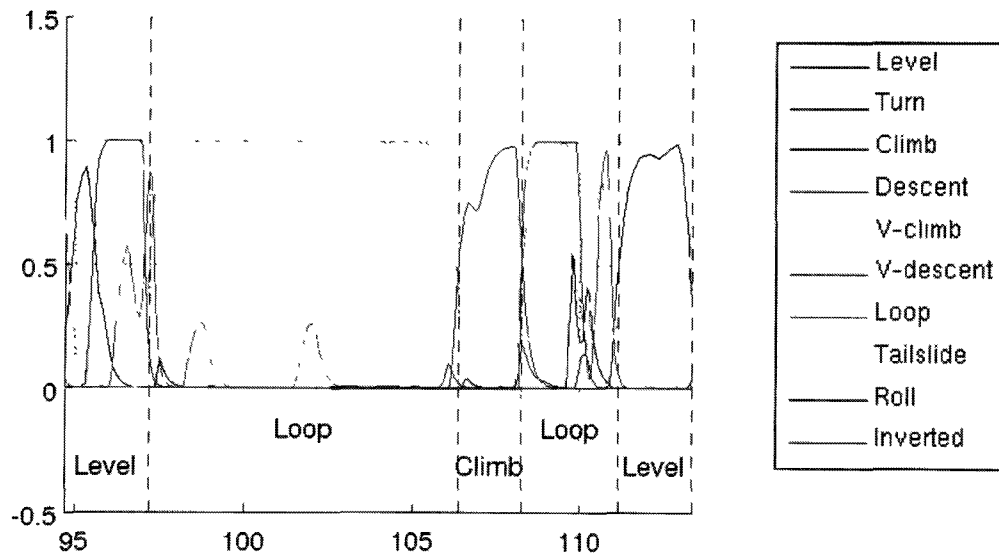
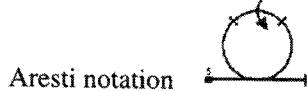


Figure 17: Regimes during reverse half cuban

This maneuver was included in the flight sequence for testing of the algorithm when confronted to a half roll at the beginning of a loop, and the results were initially not good, and showed that a change in the algorithm was necessary. After changing the way the algorithm treated roll legs between two regimes (as there is no active regime during this roll), the identification gave good results.

Avalanche (Loop with Roll on top):

Aresti number 7.5.1



Description from flight data 147° in a loop with roll
26 ft in level flight (inverted)
158° in a loop

Identified as 7.5.1, at distance 137

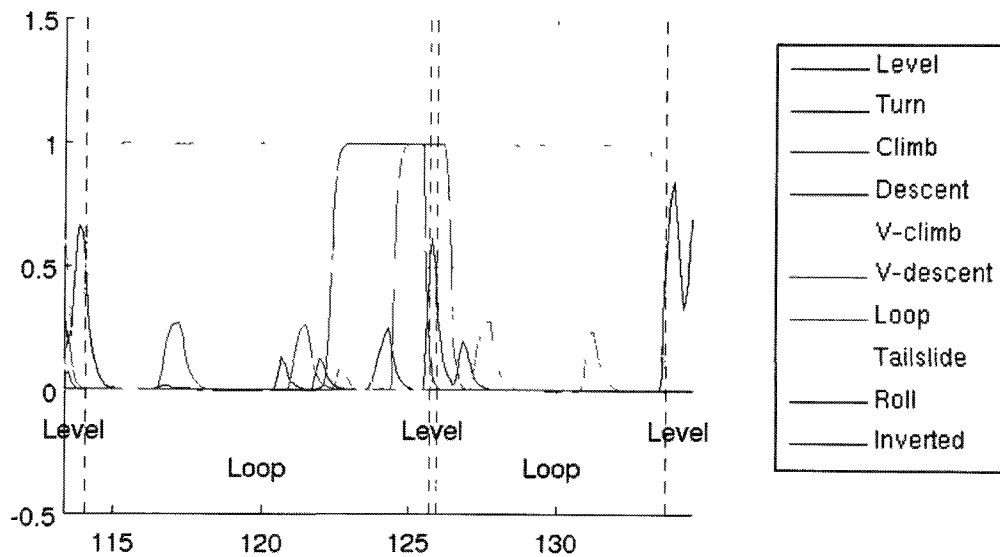



Figure 18: Regimes during avalanche (loop with roll on top)

This is another example of maneuver that combines rolls with loops, and the algorithm had trouble identifying it before the fix mentioned for the previous maneuver was performed. The level flight leg is present in the flight data and not a result of an error in the flight regimes recognition algorithm. The maneuver recognition algorithm identified this maneuver properly even though the legs detected show an error from what would be expected for this maneuver.

Immelmann:

Aresti number 7.2.1
 Aresti notation 
 Description from flight data 150° in a loop with half roll
 Identified as 7.2.1, at distance 56

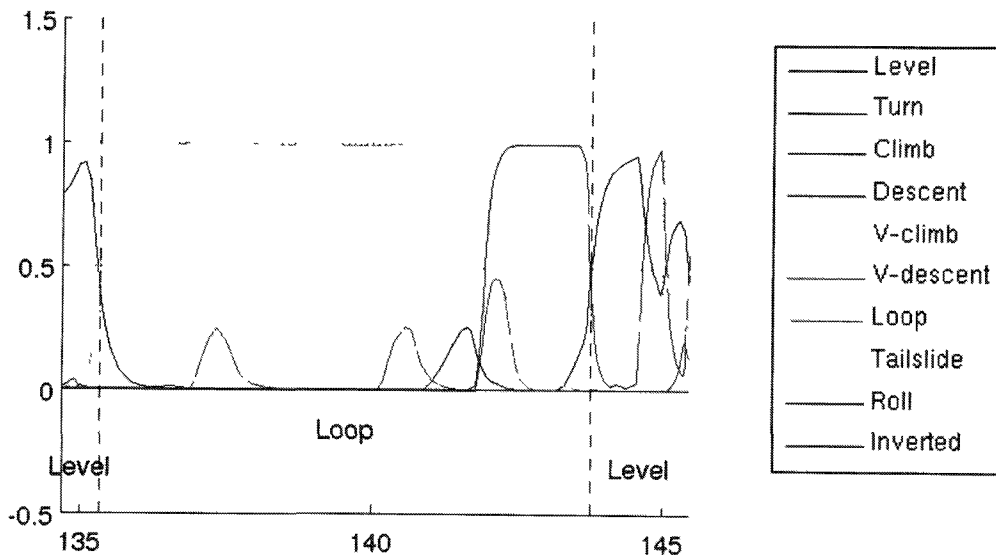


Figure 19: Regimes during Immelmann

This maneuver is the third example of loops and rolls combination, but this time the fix did not appear to be necessary as the roll happens at the end of the loop. However, before the fix, a Split S, which is the maneuver obtained by rolling to 180° of bank before entering a 180° loop would probably have been identified as an Immelmann.

The most important problem to this time, which was seen on the first two maneuvers of this sequence, is the separation of maneuvers, which is considered once and for all, and possible errors in maneuver separation is not taken into account by the maneuver identification algorithm. No solution that would not be prohibitively time consuming has been found in this study.

6 CONCLUSION

This study used a filter approach to develop a Flight Regime Recognition algorithm, which was proven to give good results and address the problem of real time FRR, since it processes the data straight from flight recorder in a continuous, time step by time step, way. In addition, the maneuvers flown implied complex regimes and transitions to which the flight regime recognition program responded with a good behavior.

Spins and snap rolls were not considered in this study in order to focus on feasibility rather than completeness. The tailslide regime was considered but not tested since all tests conducted were flown in a simulator, which brings uncertainties in terms of its behavior in a stall and a tailslide. Implementing and testing of these regimes would be necessary for future development and large scale use of this system.

No specific interface was developed for real time treatment but all it would need is the Simulink model to be implemented in the data acquisition solution. In the case of this study, it would be achieved by combining the data acquisition and the flight regime recognition models, instead of using a recording of output data from the first to feed the second one for post-processing.

This Flight Regime Recognition system shows very good results for a moderate cost, since it only requires the installation of an Inertial Measurement Unit and recording of its data. However, its approach was specifically adapted for aerobatic maneuvers and some additional data may be required for flight load analysis.

The legs identification also provided good results on both sequences and the most important source of problems is separation of the whole flight into maneuvers, which prevented correct identification in the case of level flight used as a leg of a maneuver. A priori knowledge of the sequence flown could allow better separation of the flight into maneuvers and enhance the identification of errors, which most obvious application is aerobatics rating.

The tailslides, hammerheads, rolling turns and stall turns were left out of the catalog because they were not part of the test sequences, for simplicity and because the combination

of roll and turn regimes was not very well modeled. Also, due to the way rolls are modeled superimposed on a regime for leg identification, hesitation rolls and rolling turns are not modeled properly and further development of the model should be done to allow consideration of these scenarios.

Aerobatic maneuvers are of a complex nature and capability of both regimes recognition and maneuver identification has been achieved in this study, with very good results for the first one, which would need to be completed by the addition of spins and snap rolls, and promising results in the second, which would need further development and testing to achieve completeness and robustness.

REFERENCES

- [1] David Kim and Laure Péchaud, *Maneuver Recognition and Prediction or Empennage Flight Loads of General Aviation Aircraft*. Embry-Riddle Aeronautical University, Daytona Beach, FL, AIAA 2001-5273.
- [2] Ahmet Murat Dere, *Flight Regime Recognition Analysis for the Army UH-60A IMDS Usage*, Thesis Report, Naval Postgraduate School, Monterey, CA, 2006.
- [3] S. S. Tang and L. J. O'Brien, *A Novel Method for Fatigue Life Monitoring of Non-Airframe Components*, Structural Integrity Associates, Infometrics, Inc. AIAA 1991-1088.
- [4] Rachel Rajnicek, *Application of Kalman Filtering to Real Time Flight Regime Recognition Algorithms in a Helicopter Health and Usage Monitoring System*. Thesis Report, Embry-Riddle Aeronautical University, Daytona Beach, FL, 2008.
- [5] Andrew Hess, William Hardman, Harrison Chin and John Gill, *The US Navy's Helicopter Integrated Diagnostics System (HIDS) Program: Power Drive Train Crack Detection Diagnostics and Prognostics, Life Usage Monitoring, and Damage Tolerance; Techniques, Methodologies, and Experiences*, US Naval Air Warfare Center, BFGoodrich Aerospace, 1999.
- [6] K.F. Fraser, *General Requirements and Techniques for Component Fatigue Life Substantiation in Australian Service Helicopters*, Australian Department of Defense, 1991.
- [7] John Peter Jesan, *The Neural Approach to Pattern Recognition*, Ubiquity, Volume 5, Issue 7, April 14-20, 2004.

- [8] Sergios Theodoridis, Konstantinos Koutroumbas, *Pattern Recognition*, Second Edition, 2003, ISBN 0-12-685875-6
- [9] International Aerobatics Club (IAC), *Official Contests Rules*, 2009.
- [10] Fédération Aéronautique Internationale (FAI) *Aresti Aerobatic Catalog*, Version 2003-1, 2002.

APPENDIX

A Data Acquisition Model

The data acquisition model that was used in this study uses a custom block to read data packets sent in real time by X-Plane Flight Simulator, version 8.40. X-Plane was setup to send UDP packets to the receiving computer containing the following information:

Data index	Name
01	Elapsed Time
02	Speed, Vertical Speed
03	Mach, G-load
05	Atmosphere: Ambient
07	Joystick: Ail/Elv/Rud
14	Angular Accelerations
15	Angular Velocities
16	Pitch, Roll, Heading
18	Lat, Lon, Alt
19	Loc, distance Traveled
23	Throttle Setting
33	Engine Torque
35	Prop RPM
40	MP
62	Landing Gear Vertical Force

Table 7: X Plane data export settings

The Simulink model reads data packets in real time and creates a vector of flight data, which is exported to Matlab workspace for post processing.

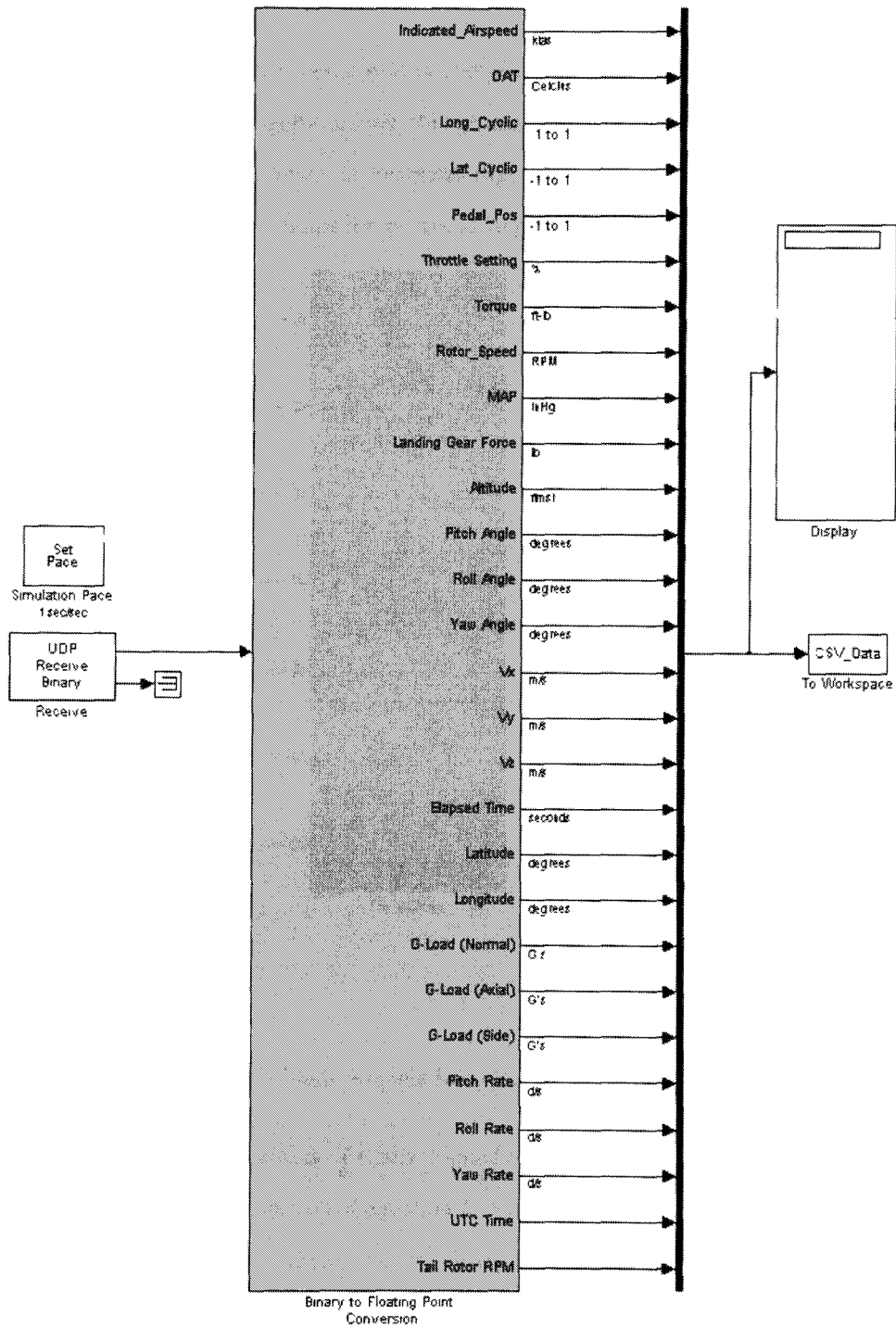


Figure 20: Data acquisition model

B Flight Regime Recognition Model

The Flight Regime Recognition being mostly composed of filters was modeled in Simulink to enable real time regime recognition, even though the current model uses recorded data. The Simulink model, depicted in figure 21, recreates flight data and uses filters to create the states vector, and multiplies it by the transition matrix H to get the vector of flight regimes.

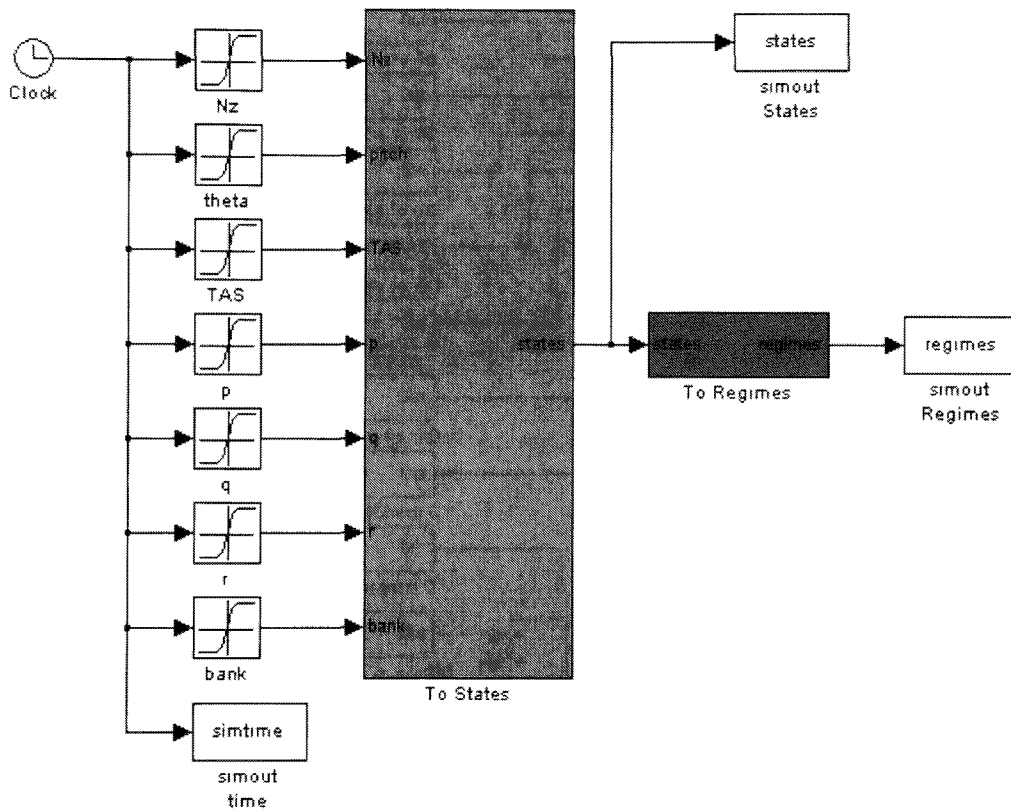


Figure 21: Flight Regime Recognition model - top level

The central block consists of filters applied to each parameter, as shown in figure 22, each filter being the implementation of equation 1, depicted in figure 23, for “close to” states, its complement to 1 for “far from” states, and specific equations for states ($TAS < 0$) and ($N_z < 0$). The “To regimes” block multiplies the states vector with the transition matrix H with upper and lower bounds set to 0 and 1 on probability of being in given regime.

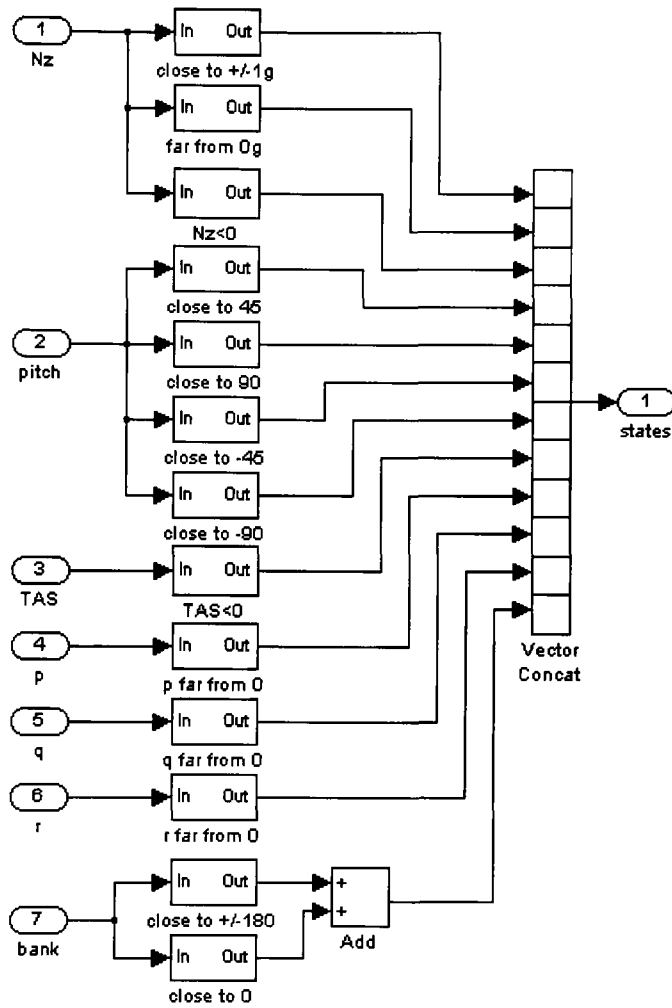


Figure 22: States determination block

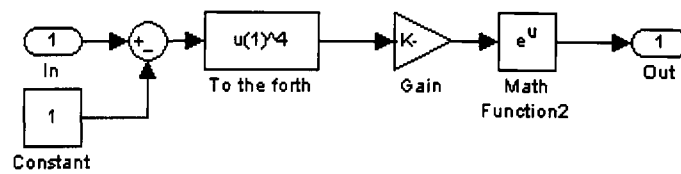


Figure 23: Implementation of equation 1 in Simulink

C States Observed During First Sequence

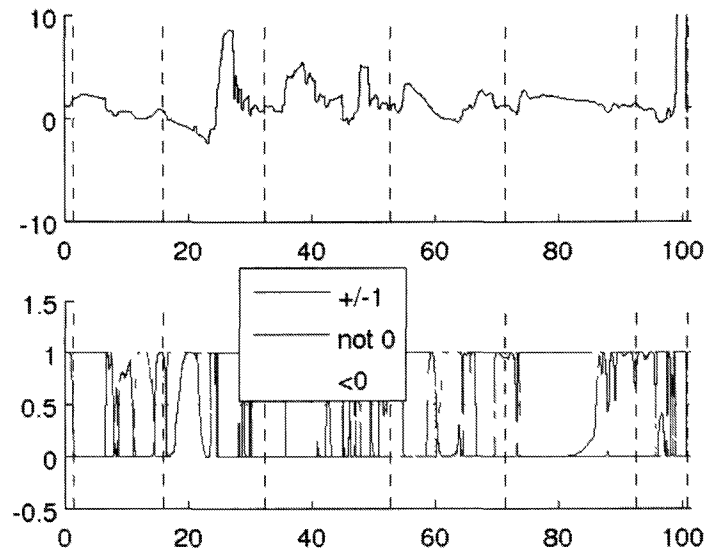


Figure 24: Load factor and associated states throughout sequence 1

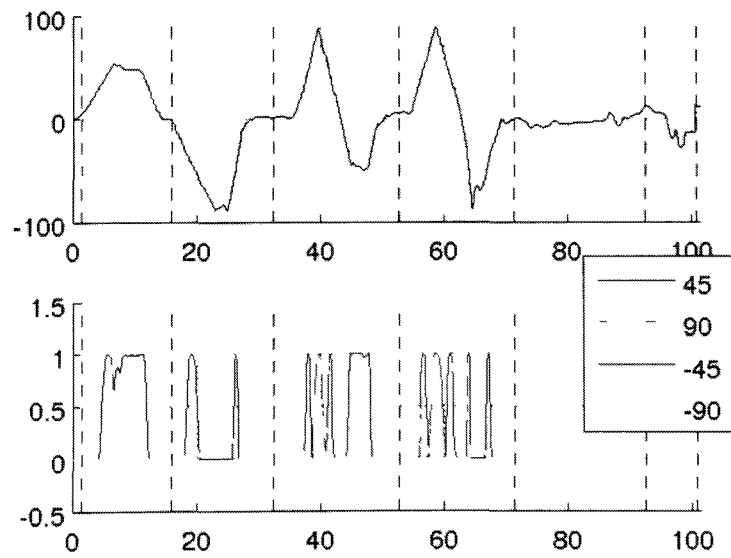


Figure 25: Pitch angle and associated states throughout sequence 1

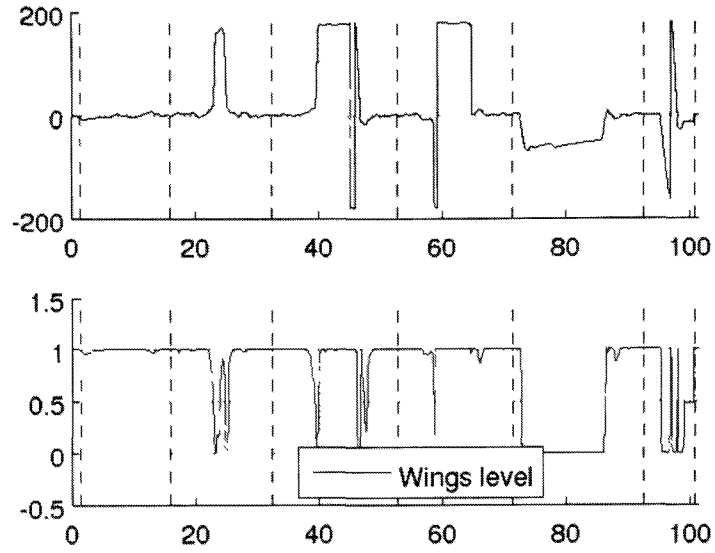


Figure 26: Bank angle and associated state throughout sequence 1

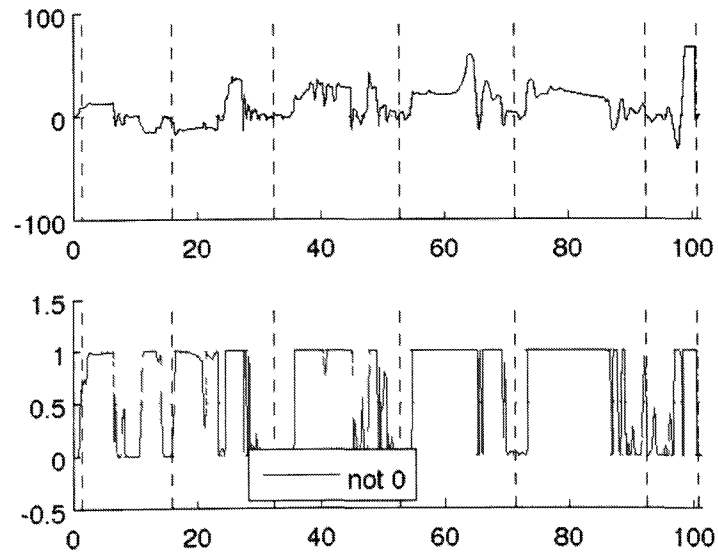


Figure 27: Pitch rate and associated state throughout sequence 1

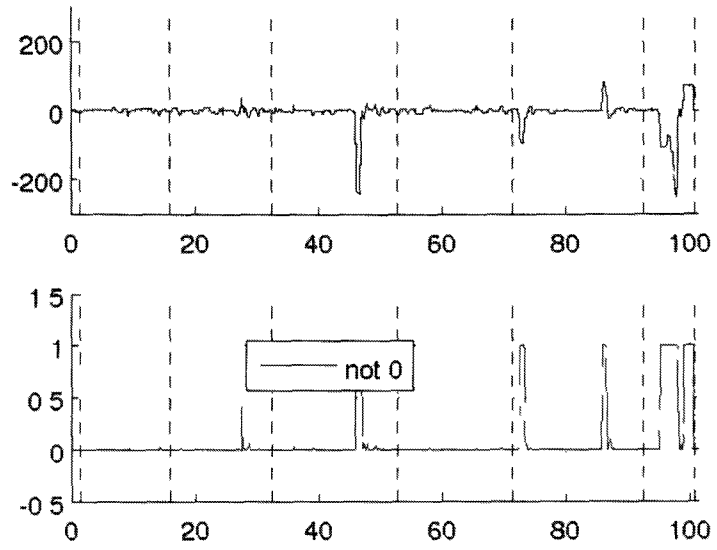


Figure 28 Roll rate and associated state throughout sequence 1

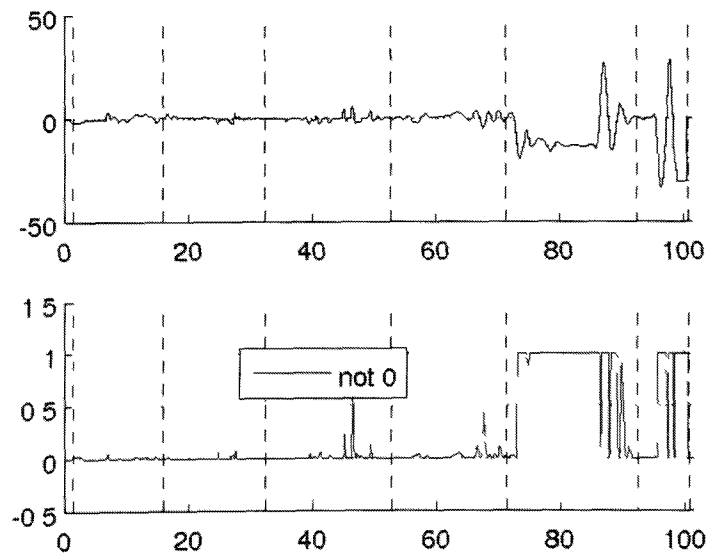


Figure 29 Yaw rate and associated state throughout sequence 1

D Source Code

The Flight Regime Recognition algorithm initialization as well as the Maneuvers Identification algorithm were implemented using Matlab, and the source code is given below.

D.1 Main Program (Import script)

The main program sets the tolerances and input data into the Simulink FRR model before running it, and then runs the maneuver identification algorithm on the model's output. It also generates graphs and outputs the maneuvers description in Matlab prompt.

```
% General flight data interpretation program preamble
clear ;
close all ;
warning('off','all') ;

%% Inputs

% File containing the flight data (Can be either a csv file or a mat file
% with
% variable CSV_Data. Make csvfile= for consideration of the mat file
csvfile='run2 csv' ;
load ponso2 ;

% Used for naming output graphs
sequence=1 ;

% Pilot tells what tolerances to use (current options = Anderson
% Ponso and anything else for default (minimal) tolerances)
pilot='Anderson' ;

catalogfile='catalog' ; % file containing catalog in plain text (input)
preloaded=1 ; % tells whether the catalog mat file contains up to date catalog
% set to 0 to force reading from the catalog file

recognition=0 % maneuver recognition from catalog set to 0 if what is
% interesting is only the maneuvers description not its catalog reference
% The recognition process being very time consuming setting this to 0 will
% save a lot of time if you don't need the maneuvers reference

%% Model parameters

% Gains for comparators ln(2)/tolerance^4 to give 0.5 when within tolerance
KNz=log(2)/0.5^4 ; % tolerance of 0.5 g on Nz-value
Kt=log(2)/10^4 ; % tolerance of 10 deg on theta+/- 45
Kt2=log(2)/20^4 ; % tolerance of 20 deg on theta+/- 90
Kp=log(2)/40^4 ; % tolerance of 40 deg/sec on p=0
if(strcmp(pilot,'Anderson'))
```

```

    Kq=log(2)/8^4 , % tolerance of 8 deg/sec on q̇=0
    Kr=log(2)/5^4 , % tolerance of 5 deg/sec on ṙ=0
elseif(strcmp(pilot,'Ponso'))
    Kq=log(2)/5^4 , % tolerance of 5 deg/sec on q̇=0
    Kr=log(2)/10^4 , % tolerance of 10 deg/sec on ṙ=0
else
    Kq=log(2)/5^4 , % tolerance of 5 deg/sec on q̇=0
    Kr=log(2)/5^4 , % tolerance of 5 deg/sec on ṙ=0
end
Kb=log(2)/15^4 , % tolerance of 15 deg on phi=±180 or 0

% Transition matrix
% Nz=1 Nż=0 Nz<0 +45 +90 -45 90 U<0 p>0 q>0 r>0 bank
H=[ 1 0 0 -1 -1 -1 -1 -1 -1 -7 -7 -5 0 % Level
    0 0 0 -1 -1 -1 -1 -1 -1 0 1 -1 % Turn
    0 0 0 1 0 0 0 0 0 -7 0 0 % Climb
    0 0 0 0 0 1 0 0 0 -7 0 0 % Descent
    0 -4 0 0 1 0 0 0 0 -7 0 0 % V-climb
    0 -4 0 0 0 0 1 0 0 -7 0 0 % V descent
    0 0 0 0 0 0 0 -1 -7 1 -5 0 % Loop
    0 0 0 0 0 0 0 1 0 0 0 0 % Tailslide
    0 0 0 0 0 0 0 0 1 0 0 0 % Rolling
    0 0 1 0 0 0 0 0 0 0 0 0 % Inverted
] ,

% minimum value to consider a state active
min_activity = 0.5

% cutoff frequency for regime change
cutoff = [5*ones(1,8) 10 10] ,

%% Catalog and Flight Data

if(recognition==1)
    % load catalog
    [catalog , extendedcatalog]=loadcatalog(catalogfile , preloaded) ,
else
    extendedcatalog=[] ,
end

% load flight data
if(~isempty(csvfile))
    data=csvread(csvfile) ,
else
    data=CSV_Data ,
end
duration=length(data(:,1)) ,

% Extract interesting data from flight record
% The indices are valid for X-plane simulator with data acquisition model
% from flight research center (HWIL simulator)
time=data(:,1) ,
Nz=data(:,22) ,
alt=data(:,12) ,
theta=data(:,13) ,

```

```

bank=data(:,14) ;
head=data(:,15) ;
TAS=data(:,2) ;
p=data(:,26) ;
q=data(:,25) ;
r=data(:,27) ;
rate_of_turn=r.*cos(bank*pi/180)+q.*sin(bank*pi/180) ;

%% Data processing
fprintf('Processing data\n') ;
tic ;
sim('processing') ;

[legs,legstime]=getflightlegs(active_regime, active_roll, active_inv, TAS, p,
    q, rate_of_turn, alt, simtime) ;
[flight,flighttime]=getmaneuvers(legs, legstime, extendedcatalog, recognition
    ) ;

toc ;

%% Output

n_fig=1 ;
% graphs ;
plotregimes ;
plotmaneuvers ;
savefigures ;
close all ;

printmaneuvers(flight) ;

```

D.2 Catalog Parsing (loadcatalog function)

This function parses a catalog file containing a list of reference maneuvers specifically formatted and generates variants of the maneuvers described. For example, following Aresti catalog's numbering, the catalog file only requires family X.X.1 maneuvers, and the function generates X.X.2, X.X.3 and X.X.4 from the first maneuver. In most cases, when the catalog has not changed since previous execution, the parsing is not necessary, and the catalog can be loaded from a mat file.

```

function [catalog, extendedcatalog]=loadcatalog(filename,preloaded)
    % loads the catalog data in file filename (or .mat file if preloaded)
    % and generates variants of given maneuvers

if(preloaded)
    load catalog ;
else

```

```

catalog_file=fopen(filename, 'rt') ;
i=0 ;
while(~ feof(catalog_file))
    i=i+1 ;
    catalogdata{i,1}=fgetl(catalog_file) ;
end
fclose(catalog_file) ;

% skip the comments and empty lines
i=1 ;
skipped=false ;
while(~ skipped)
    if(strcmp(catalogdata{i},''))
        i=i+1 ;
    elseif(catalogdata{i}(1)=='%')
        i=i+1 ;
    else
        skipped=true ;
    end
end
% Isolate each maneuver and its parameters
j=0 ;
n_maneuvers=0 ;
while(i<length(catalogdata(:,1)))
    % read regime sequence
    text=catalogdata{i} ;
    j=1 ;
    k=0 ;
    maneuver=[] ;
    while(j<=length(text))
        % skip whitespaces
        while(text(j)==' ')
            j=j+1 ;
        end
        % read regime
        regime=0 ;
        read=false ;
        while(j<=length(text) && ~ read)
            if(text(j)~=',')
                regime=regime*10+text(j) 48 ;
            else
                read=true ;
            end
            j=j+1 ;
        end
        k=k+1 ;
        maneuver(k)=regime ;
    end
    % Same maneuver down
    maneuver_down=maneuver ;
    for k=1:length(maneuver_down)
        switch(maneuver_down(k))
            case 3
                maneuver_down(k)=4 ;
        end
    end
end

```

```

        case 4
            maneuver_down(k)=3 ;
        case 5
            maneuver_down(k)=6 ;
        case 6
            maneuver_down(k)=5 ;
    end
end
% read paramchange sequence
i=i+1 ;
text=catalogdata{i} ;
j=1 ;
k=0 ;
paramchange=zeros(1, length(maneuver)) ;
while(j<=length(text))
    % skip whitespaces
    while(text(j)==' ')
        j=j+1 ;
    end
    % read change
    change=0 ;
    read=false ;
    while(j<=length(text) && ~read)
        if(text(j)~=' ','')
            change=change*10+text(j)-48 ;
        else
            read=true ;
        end
        j=j+1 ;
    end
    k=k+1 ;
    paramchange(k)=change ;
end
% read roll elements sequence
i=i+1 ;
text=catalogdata{i} ;
j=1 ;
k=0 ;
rolls=zeros(1, length(maneuver)) ;
while(j<=length(text))
    % skip whitespaces
    while(text(j)==' ')
        j=j+1 ;
    end
    % read roll
    roll=0 ;
    read=false ;
    while(j<=length(text) && ~read)
        if(text(j)~=' ','')
            roll=roll*10+text(j)-48 ;
        else
            read=true ;
        end
        j=j+1 ;
    end
end
end

```

```

        k=k+1 ;
        rolls(k)=roll ;
    end
    i=i+1 ;
    % skip the comments and empty lines
    skipped=false ;
    while (~skipped)
        if(strcmp(catalogdata{i},''))
            i=i+1 ;
        elseif(catalogdata{i}(1)=='%')
            i=i+1 ;
        else
            skipped=true ;
        end
    end
    % read inverted variants
    while (~strcmp(catalogdata{i},''))
        name=catalogdata{i} ;
        i=i+1 ;
        text=catalogdata{i} ;
        j=1 ;
        k=0 ;
        variant=zeros(1, length(manuever)) ;
        while(j<=length(text))
            % skip whitespaces
            while(text(j)==' ')
                j=j+1 ;
            end
            % read roll
            inverted=0 ;
            sign=1 ;
            read=false ;
            while(j<=length(text) && ~read)
                if(text(j)~=',')
                    if(text(j)~='-')
                        inverted=inverted*10+text(j)-48 ;
                    else
                        sign=-1 ;
                    end
                end
                else
                    read=true ;
                end
                j=j+1 ;
            end
            k=k+1 ;
            variant(k)=inverted*sign ;
        end
        n_maneuvers=n_maneuvers+1 ;
        catalog{n_maneuvers, 1}=name ;
        catalog{n_maneuvers, 2}=manuever ;
        catalog{n_maneuvers, 3}=paramchange ;
        catalog{n_maneuvers, 4}=rolls ;
        catalog{n_maneuvers, 5}=variant ;
        % Same manuever inverted
        inv_variant=variant ;
    end
end

```

```

for k=1:length(inv_variant)
    switch(inv_variant(k))
        case 0
            inv_variant(k)=1 ;
        case 1
            inv_variant(k)=0 ;
        case 2
            inv_variant(k)=3 ;
        case 3
            inv_variant(k)=2 ;
    end
end
% Change last digit of name
name(end)=name(end)+1 ;
n_maneuvers=n_maneuvers+1 ;
catalog{n_maneuvers , 1}=name ;
catalog{n_maneuvers , 2}=maneuver ;
catalog{n_maneuvers , 3}=paramchange ;
catalog{n_maneuvers , 4}=rolls ;
catalog{n_maneuvers , 5}=inv_variant ;
if(~strcmp(name(1:3),'1.1') && ~strcmp(name(1),'2'))
    % Same maneuver down
    downvariant=variant ;
    for k=1:length(maneuver_down)
        if(maneuver_down(k)==7)
            if(downvariant(k)==1)
                downvariant(k)=0 ;
            elseif(downvariant(k)==0)
                downvariant(k)=1 ;
            end
        end
    end
% Change last digit of name
name(end)=name(end)+1 ;
n_maneuvers=n_maneuvers+1 ;
catalog{n_maneuvers , 1}=name ;
catalog{n_maneuvers , 2}=maneuver_down ;
catalog{n_maneuvers , 3}=paramchange ;
catalog{n_maneuvers , 4}=rolls ;
catalog{n_maneuvers , 5}=downvariant ;
% Same maneuver inverted
inv_downvariant=downvariant ;
for k=1:length(inv_downvariant)
    switch(inv_downvariant(k))
        case 0
            inv_downvariant(k)=1 ;
        case 1
            inv_downvariant(k)=0 ;
        case 2
            inv_downvariant(k)=3 ;
        case 3
            inv_downvariant(k)=2 ;
    end
end
% Change last digit of name

```



```

        name(end)=name(end)+1 ;
        n_maneuvers=n_maneuvers+1 ;
        catalog{n_maneuvers, 1}=name ;
        catalog{n_maneuvers, 2}=maneuver_down ;
        catalog{n_maneuvers, 3}=paramchange ;
        catalog{n_maneuvers, 4}=rolls ;
        catalog{n_maneuvers, 5}=inv_downvariant ;
    end
    i=i+1 ;
end
% skip the comments and empty lines
skipped=false ;
while (~skipped && i<length(catalogdata))
    if(strcmp(catalogdata{i},' '))
        i=i+1 ;
    elseif(catalogdata{i}(1)=='%')
        i=i+1 ;
    else
        skipped=true ;
    end
end
end

% create alterations of catalog maneuvers
for j=1:length(catalog(:,1))
    extendedcatalog{j,1}=catalog{j,1} ;
    extendedcatalog{j,2}=altermaneuver(catalog(j,2:5),1,'r') ;
end

end

```

D.3 Flight Decomposition into Legs (getflightlegs function)

This is the first step of the maneuver recognition, and uses IMU data as well as regimes evolution from the Simulink FRR model.

```

function [legs, legstime]=getflightlegs(active_regime, active_roll,
    active_inv, TAS, p, q, r, alt, simtime)
% Transforms regimes history into a sequence of flight legs

% initialisation
maneuver=[] ;
paramchange=[] ;
bankchange=[] ;
inverted=[] ;
starttime=[] ;
newregime=1 ; % suppose the recording starts straight and level
j=0 ;
% enter loop
for t=1:size(simtime)-1

```

```

regime=newregime ;
knowregime=0 ;

% get dominant regime
if( active_regime(t)>0)
    newregime=active_regime(t) ;
    knowregime=1 ;
end

% case of a roll
if( active_roll(t))
    newregime=active_regime(t)+10 ;
    knowregime=1 ;
end

% get parameter change
change=0 ;
if( round(simtime(t)*30)>0 && knowregime)
    switch(newregime)
        case {1, 11}
            U_avg=mean(TAS(round(simtime(t)*30):round(simtime(t+1)
                *30))) ;
            change=U_avg*1.6878*(simtime(t+1)-simtime(t)) ;
        case {2, 12}
            r_avg=mean(r(round(simtime(t)*30):round(simtime(t+1)*30))
                ) ;
            change=r_avg*(simtime(t+1)-simtime(t)) ;
        case {3, 4, 5, 6, 13, 14, 15, 16}
            change=alt(round(simtime(t+1)*30)-alt(round(simtime(t)
                *30)) ;
        case {7, 17}
            q_avg=mean(q(round(simtime(t)*30):round(simtime(t+1)*30))
                ) ;
            change=q_avg*(simtime(t+1)-simtime(t)) ;
    end
    % change in bank angle
    p_avg=mean(p(round(simtime(t)*30):round(simtime(t+1)*30))) ;
    bchange=p_avg*(simtime(t+1)-simtime(t));
end

% if we entered a new regime
if(newregime~=regime)
    % get inverted status of previous regime
    if(j>0)
        inverted(j)=round(mean(active_inv(starttime(j):t))) ;
    end
    j=j+1 ;
    % add current regime to the maneuver history
    maneuver(j)=newregime ;
    paramchange(j)=change ;
    bankchange(j)=bchange ;
    starttime(j)=t ;
elseif(knowregime==1)
    if(j>0)
        % add parameter change

```

```

                paramchange(j)=paramchange(j)+change ;
                bankchange(j)=bankchange(j)+bchange ;
            end
        end
    end

    [legs{1,1}, legs{2,1}, legs{3,1}, legs{4,1}, legstime]=trim_maneuver(
        maneuver, paramchange, bankchange, inverted, starttime) ;
end

```

D.4 Legs Filtering (trim_maneuver function)

This function removes legs that do not respect the minimum length to be considered non-transient, and makes groups the roll legs with surrounding legs if they share the same dominant regime, except for level flight (the level flight with roll is kept as a leg by itself).

```

function [tmaneuver tparamchange rolls tinverted tstarttime]=trim_maneuver(
    maneuver, paramchange, bankchange, inverted, starttime)
% trims a maneuver to remove the transient regimes and treats the rolls

    tmaneuver=[] ;
    tparamchange=[] ;
    rolls=[] ;
    tinverted=[] ;
    tstarttime=[] ;

    if (length(maneuver)==1 && sum(bankchange)<90)
        return ;
    end
    j_trim=0 ;
    for j=1:length(maneuver)
        regime=maneuver(j) ;
        while (regime>10)
            regime=regime-10 ;
        end
        if (regime==10)
            if (j_trim~=0 && tmaneuver(j_trim)~=1)
                regime=tmaneuver(j_trim) ;
            end
        end
        minparamchange=[15, 35, 15, 15, 15, 15, 25, 10, 15, 15] ;
        if ((abs(paramchange(j))>minparamchange(regime)) || abs(bankchange(j))
            >45)
            % take this regime into account
            if (j_trim>0)
                if (tmaneuver(j_trim)==regime && (regime~=7 || inverted(j)==
                    inverted(j_trim) || paramchange(j)<minparamchange(regime)
                ))
                    % when regime does not change, add paramchange to
                    previous

```

```

    % paramchange except for loops that change directions
    tparamchange(j_trim)=tparamchange(j_trim)+paramchange(j)
    ;
    % treat changes in inverted attitude
    if (tinverted(j_trim)==0 && inverted(j)==1)
        % [0 1] = 2
        tinverted(j_trim)=2 ;
    end
    if (tinverted(j_trim)==1 && inverted(j)==0)
        % [1 0] = 3
        tinverted(j_trim)=3 ;
    end
    if (tinverted(j_trim)==3 && inverted(j)==1)
        % [1 0 1] = 1
        tinverted(j_trim)=1 ;
    end
    if (tinverted(j_trim)==2 && inverted(j)==0)
        % [0 1 0] = 0
        tinverted(j_trim)=0 ;
    end
    bank=bank+bankchange(j) ;
    rolls(j_trim)=0 ;
else
    % new regime
    j_trim=j_trim+1 ;
    rolls(j_trim)=0 ;
    tmaneuver(j_trim)=regime ;
    tstarttime(j_trim)=starttime(j) ;
    tparamchange(j_trim)=paramchange(j) ;
    tinverted(j_trim)=inverted(j) ;
    bank=bankchange(j) ;
end
else
    % first regime seen that is taken into account
    j_trim=j_trim+1 ;
    rolls(j_trim)=0 ;
    tmaneuver(j_trim)=regime ;
    tstarttime(j_trim)=starttime(j) ;
    tparamchange(j_trim)=paramchange(j) ;
    tinverted(j_trim)=inverted(j) ;
    bank=bankchange(j) ;
end
if (abs(bank)>135)
    % include rolling element
    if (abs(bank)<270)
        % half roll
        if (rolls(j_trim)==2)
            rolls(j_trim)=22 ;
        else
            rolls(j_trim)=2 ;
        end
    elseif (abs(bank)<540)
        % full roll
        rolls(j_trim)=1 ;
    end
end

```

```

        end
    end
end
tstarttime(j_trim+1)=starttime(end) ;
for j_trim=1:length(tmaneuver)
    tparamchange(j_trim)=abs(tparamchange(j_trim)) ;
end
end

```

D.5 Maneuvers Identification (getmaneuvers function)

This function performs the decomposition of the whole flight into maneuvers and, if asked to, runs the distance measurement algorithm, on the fly with maneuver cutting.

```

function [maneuvers ,flighttime]=getmaneuvers(legs , legstime , extendedcatalog
, namemaneuvers)
% Decomposes legs into maneuvers with time delimitations

% initialisation
n=1 ;
i=1 ;
for k=1:4
    maneuvers{n,k}(i)=legs{k}(1) ,
end
flighttime{n,1}(i)=legstime(1) ;
i=i+1 ;
% enter loop
for j=2:length(legs{1})
    if(legs{1}(j)==1 && legs{2}(j)>100)
        % use it for separation of maneuvers
        if(maneuvers{n,1}(i-1)~=1)
            % add the regime to current maneuver
            for k=1:4
                maneuvers{n,k}(i)=legs{k}(j) ,
            end
            flighttime{n,1}(i)=legstime(j) ;

            if(j<length(legs{1}))
                % get to the middle of the straight leg
                maneuvers{n,2}(i)=maneuvers{n,2}(i)/2 ;
                flighttime{n,1}(i+1)=floor(legstime(j)+(legstime(j+1)-
                    legstime(j))/2) ;
            else
                flighttime{n,1}(i+1)=legstime(j+1) ;
            end
        else
            % add the paramchange to previous regime
            if(j<length(legs{1}))
                % only half of it
                maneuvers{n,2}(i-1)=maneuvers{n,2}(i-1)+legs{2}(j)/2 ;
            end
        end
    end
end

```

```

        flighttime{n,1}(i)=floor((legstime(j)+(legstime(j+1)-
            legstime(j))/2) );
    else
        % total
        maneuvers{n,2}(i-1)=maneuvers{n,2}(i-1)+legs{2}(j) ;
        flighttime{n,1}(i)=legstime(j+1) ;
    end
end
% proceed to naming
if(namemaneuvers)
    maneuvers{n,5}=-1*ones(length(extendedcatalog(:,1)),1) ;

    % find distance to catalog maneuvers
    % => create list of alterations of maneuver
    alterations=altermaneuver(maneuvers(n,1:4), 1, 'f') ,
    % and compare them all to the catalog
    mindistance=200 ;
    for cat=1:length(extendedcatalog(:,1))
        for alter=1:length(alterations(:,1))
            for alt_cat=1:length(extendedcatalog{cat,2}(:,1))
                distance=comparemaneuvers(alterations(alter,1:4),
                    extendedcatalog{cat,2}(alt_cat,1:4)) ;
                if(distance~= -1)
                    distance=distance+alterations{alter,5}+
                        extendedcatalog{cat,2}{alt_cat,5} ;
                    if(maneuvers{n,5}(cat)==-1 || maneuvers{n,5}(
                        cat)>distance)
                        maneuvers{n,5}(cat)=distance ;
                    end
                end
            end
        end
    end
    if(maneuvers{n,5}(cat)~= -1 && maneuvers{n,5}(cat)<
        mindistance)
        mindistance=maneuvers{n,5}(cat) ;
        maneuvers{n,6}=extendedcatalog{cat,1} ;
    end
end
end
if(j<length(legs{1}))
    % start the next maneuver
    n=n+1 ;
    i=1 ;
    for k=1:4
        maneuvers{n,k}(i)=legs{k}(j) ;
    end
    % get to the middle of the straight leg
    maneuvers{n,2}(i)=maneuvers{n,2}(i)/2 ;
    flighttime{n,1}(i)=ceil((legstime(j)+(legstime(j+1)-legstime(j)
        ))/2) ;
    i=i+1 ;
end
else
    if(legs{1}(j)~=10)
        % simply copy the leg in maneuvers

```

```

        for k=1:4
            maneuvers{n,k}(i)=legs{k}(j) ;
        end
        flighttime{n,1}(i)=legstime(j) ;
        i=i+1 ;
    else
        % add the roll to the previous leg
        maneuvers{n,3}(i-1)=legs{3}(j) ;
        if(legs{3}(j)==2 || legs{3}(j)==24)
            if(maneuvers{n,4}(i-1)==0)
                maneuvers{n,4}(i-1)=2 ;
            else
                maneuvers{n,4}(i-1)=3 ;
            end
        end
    end
end
% check whether the end of the flight is attained
if(j==length(legs{1}))
    flighttime{n,1}(i)=legstime(j+1) ;
end
end
end
end
end

```

D.6 Mapping (altermaneuver function)

This function maps the region around a given maneuver for distance measurement. It is used for both reference maneuvers' alteration (making mistakes from the reference) and flown maneuver's alteration (canceling mistakes that could have been done).

```

function [alterations]=altermaneuver(maneuver, n_errors, maneuvertype)
    % Generates alterations of a maneuver with n_errors or less
    % Returns array of alterations and corresponding distance
    % Process differs as a function of maneuvertype.
    % - 'r' means we are altering a reference maneuver, and making errors
    % - 'f' means we are altering a flown maneuver, and canceling errors

    if (n_errors > 1)
        % generate alterations at level n-1
        alterations=altermaneuver(maneuver, n_errors-1, maneuvertype) ;
        % and alter them all by 1 error
        n_max=length(alterations(:,1)) ;
        for n=2:n_max
            % list of new alterations
            candidates=altermaneuver(alterations(n, 1:4), 1, maneuvertype) ;
            for j=2:length(candidates(:,1))
                % check whether it is already in the table of alterations
                found=false ;
                k=0 ;
                while(k<length(alterations(:,1)) && ~found)

```

```

        k=k+1 ;
        distance=comparemaneuvers( alterations(k,1:4) , candidates(
            j,1.4) ) ;
        found=(distance==0) ;
    end
    if (~found)
        % add candidate to the list of alterations
        alterations{end+1, 1}=candidates{j, 1} ;
        alterations{end, 2}=candidates{j, 2} ;
        alterations{end, 3}=candidates{j, 3} ;
        alterations{end, 4}=candidates{j, 4} ;
        alterations{end, 5}=alterations{n,5}+candidates{j,5} ;
    else
        % replace distance with shortest one
        alterations{k,5}=min( alterations{n,5}+candidates{j,5} ,
            alterations{k,5}) ;
    end
end
end
elseif (n_errors==1)
    % Generate list of alterations
    % Put the maneuver itself in the list , at distance 0
    n_alt=1 ;
    alterations{n_alt,1}=maneuver{1} ;
    alterations{n_alt,2}=maneuver{2} ;
    alterations{n_alt,3}=maneuver{3} ;
    alterations{n_alt,4}=maneuver{4} ;
    alterations{n_alt,5}=0 ;

    % Suppress a regime
    if (length(maneuver{1})>1)
        % Distance definition
        if (strcmp(maneuvertyp, 'r'))
            % Regime not seen where it should have been
            % level turn diagonal vertical loop
            distance=[ 30 40 30 30 30 30 20 ] ;
        else
            % Additional regime was seen get rid of it
            % level turn diagonal vertical loop
            distance=[ 20 40 20 20 20 20 20 ] ;
        end
        % inverted status treatment
        inverted_problem=[0 2 2 0
            3 1 1 3
            0 2 2 0
            3 1 1 3] ;

        n_alt=n_alt+1 ,
        alterations{n_alt, 1}=maneuver{1}(2:end) ;
        alterations{n_alt, 2}=maneuver{2}(2:end) ;
        alterations{n_alt, 3}=maneuver{3}(2:end) ;
        alterations{n_alt, 4}=maneuver{4}(2:end) ;

        suppressedregime=maneuver{1}(1);
        paramchange=maneuver{2}(1) ;
    end
end

```



```

alterations{n_alt, 5}=distance(suppressedregime)+2*paramchange
*(1 exp(-log(2)*(paramchange/20)^4)) ;

for j=2:length(maneuver{1})-1
    n_alt=n_alt+1 ;
    % record new regimes sequence
    regimes=[maneuver{1}(1:j-1) maneuver{1}(j+1:end)] ;
    paramchange=[maneuver{2}(1:j-1) maneuver{2}(j+1:end)] ;
    rolls=[maneuver{3}(1:j-1) maneuver{3}(j+1:end)] ;
    inverted=[maneuver{4}(1:j-1) maneuver{4}(j+1:end)] ;

    % then group repeated regimes except for loops that change
    % inverted status
    if(regimes(j-1)~=regimes(j) || (regimes(j)==7 && inverted(j)
    -1)~=inverted(j))
        alterations{n_alt, 1}=regimes ;
        alterations{n_alt, 2}=paramchange ;
        alterations{n_alt, 3}=rolls ;
        alterations{n_alt, 4}=inverted ;
    else
        % trim the repeated regime
        if(j<length(regimes))
            alterations{n_alt, 1}=[regimes(1:j-1) regimes(j+1:end)
            ] ;
            alterations{n_alt, 2}=[paramchange(1:j-1) paramchange
            (j+1:end)] ;
            alterations{n_alt, 3}=[rolls(1:j-1) rolls(j+1:end)] ;
            alterations{n_alt, 4}=[inverted(1:j-1) inverted(j+1:
            end)] ;
        else
            alterations{n_alt, 1}=regimes(1:j-1) ;
            alterations{n_alt, 2}=paramchange(1:j-1) ;
            alterations{n_alt, 3}=rolls(1:j-1) ;
            alterations{n_alt, 4}=inverted(1:j-1) ;
        end
        % and adapt paramchange, rolls and inverted
        alterations{n_alt, 2}(j-1)=paramchange(j-1)+paramchange(j)
        ) ;
        if(strcmp(maneuvertyp, `r`))
            alterations{n_alt, 3}(j-1)=max(rolls(j-1:j)) ;
        else
            if(rolls(j)~=0)
                if(rolls(j-1)~=0)
                    if(rolls(j-1)==1 && rolls(j)==1)
                        alterations{n_alt, 3}(j)=21 ;
                    end
                    if((rolls(j-1)==2 && rolls(j)==1) || (rolls(j)
                    -1)==1 && rolls(j)==2))
                        alterations{n_alt, 3}(j)=32 ;
                    end
                    if(rolls(j-1)==2 && rolls(j)==2)
                        alterations{n_alt, 3}(j)=22 ;
                    end
                    if(rolls(j-1)==22 && rolls(j)==22)
                        alterations{n_alt, 3}(j)=42 ;
                    end
                end
            end
        end
    end
end

```

```

                end
            else
                alterations{n_alt, 3}(j)=rolls(j) ;
            end
        end
    end
    if(inverted(j)~= -1 && inverted(j-1)~= -1)
        alterations{n_alt, 4}(j-1)=inverted_problem(inverted(j)
            -1)+1, inverted(j)+1) ;
    else
        alterations{n_alt, 4}(j-1)=-1 ;
    end
end

suppressedregime=maneuver{1}(j);
paramchange=maneuver{2}(j) ;
alterations{n_alt, 5}=distance(suppressedregime)+2*
    paramchange*(1-exp(-log(2)*(paramchange/20)^4)) ;
end
n_alt=n_alt+1 ;
alterations{n_alt, 1}=maneuver{1}(1:end-1) ;
alterations{n_alt, 2}=maneuver{2}(1:end-1) ;
alterations{n_alt, 3}=maneuver{3}(1:end-1) ;
alterations{n_alt, 4}=maneuver{4}(1:end-1) ;

suppressedregime=maneuver{1}(end);
paramchange=maneuver{2}(end) ;
alterations{n_alt, 5}=distance(suppressedregime)+2*paramchange
    *(1-exp(-log(2)*(paramchange/20)^4)) ;
end
% Add a regime
% that cuts another one
distance=-1*ones(8,8) ;
if(strcmp(maneuvertyp, 'r'))
    % Regime was not maintained constant while it should have been
    % line cutting a loop
    distance(1,7)=20 ;
    distance(3:6,7)=20*ones(4,1) ;
    % turn cutting a line
    distance(2,1)=40 ;
    distance(2,3:6)=40*ones(1,4) ;
    % loop cutting a line
    distance(7,1)=20 ;
    distance(7,3:6)=20*ones(1,4) ;
    % loop cutting a turn
    distance(7,2)=40 ;
else
    % A regime was not seen in between two similar regimes - add it
    % line not seen/flown between two parts of a loop
    distance(1,7)=30 ;
    distance(3:6,7)=30*ones(4,1) ;
end

% reconstruct angle between two lines (for cut loops)

```

```

%      lvl      cl des  v-c  v-d  inv      i-c  i-d
angles=[ 0.    0.  45, 315,  90, 270, 180,    0, 135, 225 % level
         0.    0.   0,   0,   0,   0,   0,    0,   0,   0 %
         315.  0.   0, 270,  45, 225, 135,    0,  90, 180 % climb
         45.   0.  90,   0, 135, 315, 225,    0, 180, 270 % desc
         270.  0. 315, 225,   0, 180,  90,    0,  45, 135 % v climb
         90.   0. 135,  45, 180,   0, 270,   0, 225, 315 % v-desc
        180.  0. 225, 135, 270,  90,   0,   0, 315,  45 % inv level
         0.    0.   0,   0,   0,   0,   0,    0,   0,   0 %
        225.  0. 270, 180, 315, 135,  45,    0,   0,  90 % inv climb
        135.  0. 180,  90, 225,  45, 315,    0, 270,   0 % inv desc
] ;

```

```

for j=1:length(maneuver{1})
    for addedregime=1:7
        if (distance(addedregime, maneuver{1}(j))>0)
            regimes=[maneuver{1}(1:j) addedregime maneuver{1}(j:end)]
                ;
            paramchange=[maneuver{2}(1:j) 5 maneuver{2}(j:end)] ;
            rolls=[maneuver{3}(1:j) 0 maneuver{3}(j:end)] ;
            inverted=[maneuver{4}(1:j) -1 maneuver{4}(j:end)] ;
            if (regimes(j)~=7)
                % no need for paramchange adaptation
                paramchange(j)=paramchange(j)/2 ;
                paramchange(j+2)=paramchange(j+2)/2 ;
                % insert alteration
                n_alt=n_alt+1 ;
                alterations{n_alt, 1}=regimes ;
                alterations{n_alt, 2}=paramchange ;
                alterations{n_alt, 3}=rolls ;
                alterations{n_alt, 4}=inverted ;
                alterations{n_alt, 5}=distance(addedregime, regimes(j)) ;
            else
                if (paramchange(j)>68) ;
                    % reconstruct loop angles
                    [fromline, frominv]=findprevline(maneuver, j) ;
                    % get previous and next regimes
                    if (j>1)
                        previousregime=maneuver{1}(j-1) ;
                        previousinverted=maneuver{4}(j-1) ;
                    else
                        previousregime=fromline ;
                        previousinverted=frominv ;
                    end
                    if (j<length(maneuver{1}))
                        nextregime=maneuver{1}(j+1) ;
                        nextinv=maneuver{4}(j+1) ;
                    else
                        nextregime=1 ;
                        nextinv=0 ;
                    end
                    % reconstruct angle
                    for inv=0:1
                        if (addedregime<5)

```

```

        inverted(j+1)=inv ;
    else
        inverted(j+1)=-1 ;
    end
    if((addedregime~=previousregime || inverted(j
+1)~=previousinverted) && (inverted(j+1)
~-1 || inv==0) && (addedregime~=
nextregime || inverted(j+1)~=nextinv))
        % not cutting a loop by the line it
        started
        % from (unless loop changed direction in
        % the middle) or exits to
        % and ignoring inverted variant of
        % vertical lines
        if (fromline+6*frominv >10)
            frominv=0 ;
        end
        deltaparamchange=angles (fromline+6*
            frominv , addedregime+6*inv) ;
        if (inverted(j))
            % pushing instead of pulling
            deltaparamchange=360-deltaparamchange
            ;
        end

        if (maneuver{2}(j)>deltaparamchange)
            paramchange(j)=deltaparamchange ;
            paramchange(j+2)=maneuver{2}(j)-
                deltaparamchange ;
            % insert alteration
            n_alt=n_alt+1 ;
            alterations{n_alt, 1}=regimes ;
            alterations{n_alt, 2}=paramchange ;
            alterations{n_alt, 3}=rolls ;
            alterations{n_alt, 4}=inverted ;
            alterations{n_alt, 5}=distance (
                addedregime , regimes(j)) ;
        end
    end
end
end
end
end
end
end
end
end

% Add a regime between two regimes
if (strcmp (maneuvertype , 'r'))
    for j=2:length (maneuver{1}-1)
        % line flown between two loops (a push and a pull)
        if (maneuver{1}(j)==7 && maneuver{1}(j+1)==7)
            % get line at which second loop should start
            [addedregime , addedinv]=findprevline (maneuver , j+1) ;

            n_alt=n_alt+1 ;
        end
    end
end

```

```

alterations{n_alt,1}=[maneuver{1}(1:j) addedregime
maneuver{1}(j+1:end)] ;
alterations{n_alt,2}=[maneuver{2}(1:j) 5 maneuver{2}(j+1:
end)] ;
alterations{n_alt,3}=[maneuver{3}(1:j) 0 maneuver{3}(j+1:
end)] ;
alterations{n_alt,4}=[maneuver{4}(1:j) addedinv maneuver
{4}(j+1:end)] ;
alterations{n_alt,5}=20 ;
end
% loop seen after or before a turn
if (maneuver{1}(j)==2)
for i=0:1
% adding before and after
for addedinv=0:1
% both possible inverted status
n_alt=n_alt+1 ;

alterations{n_alt,1}=[maneuver{1}(1:j+i-1) 7
maneuver{1}(j+i:end)] ;
alterations{n_alt,2}=[maneuver{2}(1:j+i-1) 0
maneuver{2}(j+i:end)] ;
alterations{n_alt,3}=[maneuver{3}(1:j+i-1) 0
maneuver{3}(j+i:end)] ;
alterations{n_alt,4}=[maneuver{4}(1:j+i-1)
addedinv maneuver{4}(j+i:end)] ;
alterations{n_alt,5}=20 ;
end
end
end
else
% undetected loop between two lines
% reconstruct angle between two lines
%


|          | lv   | cl | des  | v-c  | v-d  | inv  | i-c  | i-d |      |     |          |
|----------|------|----|------|------|------|------|------|-----|------|-----|----------|
| angles=[ | 0,   | 0, | 45,  | 315, | 90,  | 270, | 180, | 0,  | 135, | 225 | % level  |
|          | 0,   | 0, | 0,   | 0,   | 0,   | 0,   | 0,   | 0,  | 0,   | 0   | %        |
|          | 315, | 0, | 0,   | 270, | 45,  | 225, | 135, | 0,  | 90,  | 180 | % climb  |
|          | 45,  | 0, | 90,  | 0,   | 135, | 315, | 225, | 0,  | 180, | 270 | % desc   |
|          | 270, | 0, | 315, | 225, | 0,   | 180, | 90,  | 0,  | 45,  | 135 | % v-     |
|          |      |    |      |      |      |      |      |     |      |     | climb    |
|          | 90,  | 0, | 135, | 45,  | 180, | 0,   | 270, | 0,  | 225, | 315 | % v-desc |
|          | 180, | 0, | 225, | 135, | 270, | 90,  | 0,   | 0,  | 315, | 45  | % inv    |
|          |      |    |      |      |      |      |      |     |      |     | level    |
|          | 0,   | 0, | 0,   | 0,   | 0,   | 0,   | 0,   | 0,  | 0,   | 0   | %        |
|          | 225, | 0, | 270, | 180, | 315, | 135, | 45,  | 0,  | 0,   | 90  | % inv    |
|          |      |    |      |      |      |      |      |     |      |     | climb    |
|          | 135, | 0, | 180, | 90,  | 225, | 45,  | 315, | 0,  | 270, | 0   | % inv    |
|          |      |    |      |      |      |      |      |     |      |     | desc     |


] ;
lines=[1 3 4 5 6] ;
for j=1:length(maneuver{1})-1
found=false ;
i=0 ;
while(i<length(lines) && ~found)

```

```

        i=i+1 ;
        if (maneuver{1}(j)==lines(i))
            found=true ;
        end
    end
end
if(found)
    % regime j is a line
    i=0 ;
    found=false ;
    while(i<length(lines) && ~found)
        i=i+1 ;
        if (maneuver{1}(j+1)==lines(i))
            found=true ;
        end
    end
end
if(found)
    % regime j+1 is also a line -> add both loops (inv
    % or not)
    switch(maneuver{4}(j))
        case {-1, 0,3}
            startinv=0 ;
        case {1,2}
            startinv=1 ;
    end
    switch(maneuver{4}(j+1))
        case {-1,0,2}
            finishinv=0 ;
        case {1,3}
            finishinv=1 ;
    end
    for addedinv=0:1
        if (maneuver{1}(j)+6*startinv >10)
            startinv=0 ;
        end
        if (addedinv==0)
            addedparamchange=angles(maneuver{1}(j)+6*
                startinv , maneuver{1}(j+1)+6*finishinv) ;
        else
            addedparamchange=360-angles(maneuver{1}(j)+6*
                startinv , maneuver{1}(j+1)+6*finishinv) ;
        end
        distance=20*addedparamchange/45 ;

        n_alt=n_alt+1 ;

        alterations{n_alt,1}=[maneuver{1}(1:j) 7 maneuver
            {1}(j+1:end)] ;
        alterations{n_alt,2}=[maneuver{2}(1:j)
            addedparamchange maneuver{2}(j+1:end)] ;
        alterations{n_alt,3}=[maneuver{3}(1:j) 0 maneuver
            {3}(j+1:end)] ;
        alterations{n_alt,4}=[maneuver{4}(1:j) addedinv
            maneuver{4}(j+1:end)] ;
        alterations{n_alt,5}=distance ;
    end
end

```

```

    end
elseif (maneuver{1}(j)==7 && maneuver{1}(j+1)==7)
    % line not seen between two loop (a pull and a push)
    % get line at which second loop should start
    [addedregime , addedinv]=findprevline(maneuver , j+1) .

    n_alt=n_alt+1 ;
    alterations{n_alt,1}=[maneuver{1}(1:j) addedregime
        maneuver{1}(j+1:end)] ,
    alterations{n_alt,2}=[maneuver{2}(1:j) 0 maneuver{2}(j+1:
        end)] ,
    alterations{n_alt,3}=[maneuver{3}(1:j) 0 maneuver{3}(j+1:
        end)] ;
    alterations{n_alt,4}=[maneuver{4}(1:j) addedinv maneuver
        {4}(j+1:end)] ;
    alterations{n_alt,5}=30 ;
end
end

% initial level flight not seen
if (maneuver{1}(1)~=1)
    % find start inv status
    addedinv=[0, 1] ;
    % and add the line
    for i=1:length(addedinv)
        n_alt=n_alt+1 ;
        alterations{n_alt,1}=[1 maneuver{1}] ,
        alterations{n_alt,2}=[0 maneuver{2}] ;
        alterations{n_alt,3}=[0 maneuver{3}] ;
        alterations{n_alt,4}=[addedinv(i) maneuver{4}] ,
        alterations{n_alt,5}=10 ;
    end
end

% final level flight not seen
if (maneuver{1}(end)~=1)
    % find start inv status
    addedinv=[0, 1] ;
    % and add the line
    for i=1:length(addedinv)
        n_alt=n_alt+1 ;
        alterations{n_alt,1}=[maneuver{1} 1] ;
        alterations{n_alt,2}=[maneuver{2} 0] ;
        alterations{n_alt,3}=[maneuver{3} 0] ;
        alterations{n_alt,4}=[maneuver{4} addedinv(i)] ;
        alterations{n_alt,5}=10 ;
    end
end

end

% Alter a regime
% change a line `s angle
lines=[1 3 4 5 6] ;
%      +45 -45 from inv

```

```

newlines=[3, 4, 4, 3 % level
          0, 0, 0, 0 %
          5, 1, 1, 5 % climb
          1, 6, 6, 1 % desc
          3, 3, 0, 0 % v-d
          4, 4, 0, 0 % v c
          ] ;

for j=2:length(manuever{1})-1
    % check wether it is a line
    found=false ;
    i=0 ;
    while(i<length(lines) && ~found)
        i=i+1 ;
        if(manuever{1}(j)==lines(i))
            found=true ;
        end
    end
    % get its inverted status
    switch(manuever{4}(j))
        case {-1, 0, 2}
            inv=0 ;
        case {1, 3}
            inv=1 ;
    end
    % and alter it (+45 and -45 )
    for i=1:2
        regimes=manuever{1} ;
        inverted=manuever{4} ;
        if(regimes(j)==5 || regimes(j)==6)
            inverted(j)=i-1 ;
        end

        regimes(j)=newlines(manuever{1}(j), i+2*inv) ;

        if(regimes(j)==5 || regimes(j)==6)
            inverted(j)=-1 ;
        end

        n_alt=n_alt+1 ;
        alterations{n_alt,1}=regimes ;
        alterations{n_alt,2}=manuever{2} ;
        alterations{n_alt,3}=manuever{3} ;
        alterations{n_alt,4}=inverted ;
        alterations{n_alt,5}=30 ;
    end
end
end
else
    % Put the manuever itself in the list, at distance 0
    n_alt=1 ;
    alterations{n_alt,1}=manuever{1} ;
    alterations{n_alt,2}=manuever{2} ;
    alterations{n_alt,3}=manuever{3} ;

```



```

        alterations{n_alt,4}=maneuver{4} ;
        alterations{n_alt,5}=0 ;
    end
end

```

end

D.7 Comparing Maneuvers (comparemaneuvers function)

This function measures the distance between two maneuvers that have the same regimes sequence. This is the function that considers errors in pitch angle change during loops, and errors in rolls.

```

function distance=comparemaneuvers(maneuver, reference)
    % compares a maneuver to a reference maneuver
    % returns the distance between the two if the regimes sequence match or
    % -1 if they don't

    if (length(maneuver{1})==length(reference{1}))
        match=true ;
        j=1 ;
        while (j<=length(maneuver{1}) && match)
            if (maneuver{1}(j)~=reference{1}(j) || (maneuver{4}(j)~=reference
                {4}(j) && reference{4}(j)~= -1))
                % no match if a difference is noted on regimes sequence or
                % inverted sequence (except if inverted= 1 on reference ,
                % which means it does not matter)
                match=false ;
            end
            j=j+1 ;
        end
        if (match)
            distance=0 ;
            for j=1:length(maneuver{2})
                if (reference{2}(j)~=0)
                    % distance as a function of parameter change and
                    % reference change in pitch angle
                    delta=abs(maneuver{2}(j)-reference{2}(j)) ;
                    distance=distance+2*delta*(1-exp(-log(2)*(delta/20)^4)) ;
                end
                % Check for rolls
                switch (reference{3}(j))
                    case 0
                        % No roll authorized
                        if (maneuver{3}(j)>0)
                            distance=distance+30 ;
                        end
                    case 1
                        % Only full roll authorized
                        if (maneuver{3}(j)==2)
                            distance=distance+30 ;
                        end
                    end
            end
        end
    end

```

```

                end
            case 2
                % Mandatory half roll
                if (maneuver{3}(j)~=2)
                    distance=distance+30 ;
                end
            end
        end
    else
        distance=-1 ;
    end
else
    distance=-1 ;
end
end
end

```

D.8 Plots scripts

graphs

This script generates the flight data graphs used in this report.

```

% plots graphs of flight data and associated regimes

fig(n_fig)=figure(n_fig) ;
figname{n_fig,1}='01-Nz' ;
n_fig=n_fig+1 ;
subplot(2,1,1)
hold on
plot(time,Nz, 'k') ;
for n=1:length(flighttime)
    plot([simtime(flighttime{n}(1)) simtime(flighttime{n}(1))], [-10, 10],
        '-k')
    plot([simtime(flighttime{n}(end)) simtime(flighttime{n}(end))], [-10,
        10], '-.k')
end
axis([0 duration -10 10])
% legend('Nz', 'Location', 'Best')
grid on
hold off

subplot(2,1,2)
hold on
plot(simtime, states(:,1), 'r') ;
plot(simtime, states(:,2), 'b') ;
plot(simtime, states(:,3), 'g') ;
for n=1:length(flighttime)
    plot([simtime(flighttime{n}(1)) simtime(flighttime{n}(1))], [-0.5, 1.5],
        '-.k')
    plot([simtime(flighttime{n}(end)) simtime(flighttime{n}(end))], [-0.5,
        1.5], '-.k')
end

```

```

end
axis([0 duration -0.5 1.5])
legend('+/-1', 'not_0', '<0', 'Location', 'Best')
grid on
hold off

fig(n_fig)=figure(n_fig) ;
filename{n_fig,1}='02-theta' ;
n_fig=n_fig+1 ;
subplot(2,1,1)
hold on
plot(time,theta, 'k') ;
for n=1:length(flighttime)
    plot([simtime(flighttime{n}(1)) simtime(flighttime{n}(1))], [-100, 100],
        '-.k')
    plot([simtime(flighttime{n}(end)) simtime(flighttime{n}(end))], [-100,
        100], '-.k')
end
axis([0 duration -100 100])
% legend('theta', 'Location', 'Best')
grid on
hold off

subplot(2,1,2)
hold on
plot(simtime, states(:,4), 'r') ;
plot(simtime, states(:,5), 'g') ;
plot(simtime, states(:,6), 'b') ;
plot(simtime, states(:,7), 'y') ;
for n=1:length(flighttime)
    plot([simtime(flighttime{n}(1)) simtime(flighttime{n}(1))], [-0.5, 1.5],
        '-.k')
    plot([simtime(flighttime{n}(end)) simtime(flighttime{n}(end))], [-0.5,
        1.5], '-.k')
end
axis([0 duration -0.5 1.5])
legend('45', '90', '-45', '-90', 'Location', 'Best')
grid on
hold off

fig(n_fig)=figure(n_fig) ;
filename{n_fig,1}='03-phi' ;
n_fig=n_fig+1 ;
subplot(2,1,1)
hold on
plot(time,bank, 'k') ;
for n=1:length(flighttime)
    plot([simtime(flighttime{n}(1)) simtime(flighttime{n}(1))], [-200, 200],
        '-.k')
    plot([simtime(flighttime{n}(end)) simtime(flighttime{n}(end))], [-200,
        200], '-.k')
end
axis([0 duration -200 200])

```

```

% legend('bank angle', 'Location', 'Best')
grid on
hold off

subplot(2,1,2)
hold on
plot(simtime, states(:,12), 'r') ;

for n=1:length(flighttime)
    plot([simtime(flighttime{n}(1)) simtime(flighttime{n}(1))], [-0.5, 1.5],
        '-k')
    plot([simtime(flighttime{n}(end)) simtime(flighttime{n}(end))], [-0.5,
        1.5], '-k')
end

axis([0 duration -0.5 1.5])
legend('Wings_level', 'Location', 'Best')
grid on
hold off

fig(n_fig)=figure(n_fig) ;
filename{n_fig,1}='04-q' ;
n_fig=n_fig+1 ;
subplot(2,1,1)
hold on
plot(time,q, 'k') ;
for n=1:length(flighttime)
    plot([simtime(flighttime{n}(1)) simtime(flighttime{n}(1))], [-100, 100],
        '-k')
    plot([simtime(flighttime{n}(end)) simtime(flighttime{n}(end))], [-100,
        100], '-k')
end
axis([0 duration -100 100])
% legend('pitch rate', 'Location', 'Best')
grid on
hold off

subplot(2,1,2)
hold on
plot(simtime, states(:,10), 'r') ;
for n=1:length(flighttime)
    plot([simtime(flighttime{n}(1)) simtime(flighttime{n}(1))], [-0.5, 1.5],
        '-k')
    plot([simtime(flighttime{n}(end)) simtime(flighttime{n}(end))], [-0.5,
        1.5], '-k')
end
axis([0 duration -0.5 1.5])
legend('not_0', 'Location', 'Best')
grid on
hold off

fig(n_fig)=figure(n_fig) ;
filename{n_fig,1}='05-p' ;

```

```

n_fig=n_fig+1 ;
subplot(2,1,1)
hold on
plot(time ,p, 'k') ;
for n=1:length(flighttime)
    plot([ simtime(flighttime{n}(1)) simtime(flighttime{n}(1))], [-300, 300],
        '-.k')
    plot([ simtime(flighttime{n}(end)) simtime(flighttime{n}(end))], [-300,
        300], '-.k')
end
axis([0 duration -300 300])
% legend('roll rate', 'Location', 'Best')
grid on
hold off

subplot(2,1,2)
hold on
plot(simtime , states(:,9), 'r') ;
for n=1:length(flighttime)
    plot([ simtime(flighttime{n}(1)) simtime(flighttime{n}(1))], [-0.5, 1.5],
        '-.k')
    plot([ simtime(flighttime{n}(end)) simtime(flighttime{n}(end))], [-0.5,
        1.5], '-.k')
end
axis([0 duration -0.5 1.5])
legend('not_0', 'Location', 'Best')
grid on
hold off

fig(n_fig)=figure(n_fig) ;
figname{n_fig,1}='06-r' ;
n_fig=n_fig+1 ;
subplot(2,1,1)
hold on
plot(time ,r, 'k') ;
for n=1:length(flighttime)
    plot([ simtime(flighttime{n}(1)) simtime(flighttime{n}(1))], [-50, 50],
        '-.k')
    plot([ simtime(flighttime{n}(end)) simtime(flighttime{n}(end))], [-50,
        50], '-.k')
end
axis([0 duration -50 50])
% legend('yaw rate', 'Location', 'Best')
grid on
hold off

subplot(2,1,2)
hold on
plot(simtime , states(:,11), 'r') ;
for n=1:length(flighttime)
    plot([ simtime(flighttime{n}(1)) simtime(flighttime{n}(1))], [-0.5, 1.5],
        '-.k')
    plot([ simtime(flighttime{n}(end)) simtime(flighttime{n}(end))], [-0.5,
        1.5], '-.k')
end

```

```

end
axis([0 duration -0.5 1.5])
legend('not_0', 'Location', 'Best')
grid on
hold off

```

plotmaneuvers

This script generate the graph that is used to generate the regimes during maneuvers figures. It consists of a plot of the regimes vector evolution in time, along with captioning the legs from the flight description.

```

% Creates a plot of regimes after lowpass filter with marks at regimes
% transitions and invisible labels on regimes

fig(n_fig)=figure(n_fig) ;
n_fig=n_fig+1 ;
hold on
grid on

plot(simtime, regimes_lowpass(:,1), 'Color', [0 0 0]) ; % straight & level
plot(simtime, regimes_lowpass(:,2), 'Color', [0.749 0 0 749]) ; % turn
plot(simtime, regimes_lowpass(:,3), 'Color', [0 0 1]) ; % climb
plot(simtime, regimes_lowpass(:,4), 'Color', [0 1 0]) ; % descent
plot(simtime, regimes_lowpass(:,5), 'Color', [0.8 0.8 0.8]) ; % V-climb
plot(simtime, regimes_lowpass(:,6), 'Color', [0.5774 0 5774 0.5774]) ; % V-
desc
plot(simtime, regimes_lowpass(:,7), 'Color', [0.694 0.5744 0.392]) ; % loop
plot(simtime, regimes_lowpass(:,8), 'Color', [0.8 0.8 0.8]) ; % tailslide
plot(simtime, regimes_lowpass(:,9), 'Color', [1 0 0]) ; % roll
plot(simtime, regimes_lowpass(:,10), 'Color', [0 0.5744 0.5744]) ; % inverted

axis([0 duration -0.5 1.5]) ;

% legend('Level', 'Turn', 'Climb', 'Descent', 'V-climb', 'V-descend', . . .
%       'Loop', 'Tailslide', 'Roll', 'Inverted', 'Location', 'EastOutside') ;

y_text=-0.15 ;
for n=1:length(flighttime)
    for j=1:(length(flighttime{n})-1)
        plot([simtime(flighttime{n}(j)) simtime(flighttime{n}(j+1))], [-0.5,
            1.5], '-.k')
        x_text=(simtime(flighttime{n}(j))+simtime(flighttime{n}(j+1)))/2 ;
        if (y_text==--0.15)
            y_text=-0.3 ;
        else
            y_text=-0.15 ;
        end
        texthandle{n,1}(j)=text(x_text, y_text, regimeskey(flight{n,1}(j)), '
            HorizontalAlignment', 'center', 'Visible', 'Off') ;
    end
end

```

```

end
clear x_text y_text ;
hold off

```

savefigures

This script was developed to make saving all figures generated by the program easier. It formats the figures to two different formats and saves a version of each one for integration in this report and the presentation.

```

% Exports the figures in current directory

for i_fig=1:length(fig)-1 .
    figure(i_fig) ;
    set(gcf, 'PaperPositionMode', 'auto') ;
    set(gcf, 'Position', [0 50 800 400]) ;
    print('-dpng', strcat(figname{i_fig}, '_big.png')) ;
    set(gcf, 'Position', [0 50 400 300]) ;
    print('-dpng', strcat(figname{i_fig}, '.png')) ;
    print('-deps', strcat(figname{i_fig}, '.eps')) ;
end

figure(fig(end))

if(sequence==1)
    filename={'11-Climb'; '12-Dive'; '13-Cuban'; '14-Loop'; '15-Turn'; '16-Roll'}
;
elseif(sequence==2)
    filename={'21-S_1'; '22-S_2'; '23-Square_1'; '24-Square_2'; '25-Square_3'; '
26-X'; '27-Rev_Cuban'; '28-Loop_roll'; '29-Immelmann'} ;
end
for n=1:length(flighttime)
    axis([simtime(flighttime{n}(1)) simtime(flighttime{n}(end)) -0.5 1.5]) ;
    for j=1:length(texthandle{n})
        set(texthandle{n}(j), 'Visible', 'on') ;
    end

    set(gcf, 'PaperPositionMode', 'auto', 'Position', [0 50 600 400]) ;
    print('-dpng', strcat(filename{n}, '_big.png')) ;
    set(gcf, 'PaperPositionMode', 'auto', 'Position', [0 50 400 300]) ;
    print('-dpng', strcat(filename{n}, '.png')) ;
    print('-deps', strcat(filename{n}, '.eps')) ;
    for j=1:length(texthandle{n})
        set(texthandle{n}(j), 'Visible', 'off') ;
    end
end

if(sequence==2)
    % group maneuvers 1 and 2 (two legs of S) on the same graph
    axis([simtime(flighttime{1}(1)) simtime(flighttime{2}(end)) -0.5 1.5]) ;
    for n=1:2

```

```

        for j=1:length(texthandle{n})
            set(texthandle{n}(j), 'Visible', 'on') ;
        end
    end
    print('-dpng', '21-S.png') ;
    print('-deps', '21-S.eps') ;
    for n=1:2
        for j=1:length(texthandle{n})
            set(texthandle{n}(j), 'Visible', 'off') ;
        end
    end
    % group maneuvers 3, 4 and 5 on the same graph
    axis([simtime(flighttime{3}(1)) simtime(flighttime{5}(end)) -0.5 1.5]) ;
    for n=3:5
        for j=1:length(texthandle{n})
            set(texthandle{n}(j), 'Visible', 'on') ;
        end
    end
    print('-dpng', '23-Square.png') ;
    print('-deps', '23-Square.eps') ;
    for n=3:5
        for j=1:length(texthandle{n})
            set(texthandle{n}(j), 'Visible', 'off') ;
        end
    end
end
end

```

D.9 Minor Functions

A few more functions were defined and used in the process.

Maneuvers Description Output

This function outputs the description of the maneuvers in Matlab prompt. It basically calls `printlegs` for each maneuver.

```

function printmaneuvers(flight)
% displays maneuvers description in the command window
    for n=1:length(flight(:,1))
        fprintf('Maneuver_%i', n) ;
        printlegs(flight(n,1:4))
        if(length(flight(n,:))>=6)
            if(~isempty(length(flight{n,6})))
                fprintf('\nIdentified as_%s', flight{n,6}) ;
            end
        end
        fprintf('\n\n') ;
    end
end
end

```


Legs Description Output

The printlegs function outputs the description of the legs in sequence in Matlab prompt

```
function printlegs(legs)
    for i=1:length(legs{1})
        regime= ' ',
        switch(legs{1}(i))
            case 1
                if((i>1 && i<length(legs{1})) || legs{3}(i)~=0 || legs{4}(i)
                    ==1)
                    regime='ft_in_level_flight' ,
                end
            case 2
                regime='deg_in_a_turn' ,
            case 3
                regime='ft_climb' ,
            case 4
                regime='ft_descent' ,
            case 5
                regime='ft_vertical_climb' ,
            case 6
                regime='ft_vertical_descent' ,
            case 7
                regime='deg_in_a_loop' ,
            case 8
                regime='tailslide' ,
        end
        if(legs{4}(i)==1)
            regime=strcat(regime, '_(inverted)') ,
        end
        switch(legs{3}(i))
            case 1
                regime=strcat(regime, '_with_roll') ,
            case 2
                regime=strcat(regime, '_with_half_roll') ,
                if(legs{4}(i)==2)
                    regime=strcat(regime, '_to_inverted') ,
                elseif(legs{4}(i)==3)
                    regime=strcat(regime, '_from_inverted') ,
                end
        end
        if(~isempty(regime))
            fprintf('\n%i %s', round(legs{2}(i)), regime) ,
        end
    end
end
```

Regimes Key

The regimeskey function makes the correspondance between regimes indices and names.

```

function textregime=regimeskey(regime)
% regime number to name
switch(regime)
  case 1
    textregime='Level' ;
  case 2
    textregime='Turn' ;
  case 3
    textregime='Climb' ;
  case 4
    textregime='Descent' ;
  case 5
    textregime='V-Climb' ;
  case 6
    textregime='V-Descent' ;
  case 7
    textregime='Loop' ;
  case 8
    textregime='Tailslide' ;
  case {9, 10}
    textregime='Roll' ;
  default
    textregime='Unknown' ;
end
end

```

Previous line

Used to determine which lines can cut a loop leg, by determining on which line the loop started. Also used when adding a loop in between two lines.

```

function [previousline , previousinverted]=findprevline(maneuver ,j)
% gets the line from which current loop (j) started
found=false ;
i=j-1 ;
angle=0 ;
while(~found && i>0)
  switch(maneuver{1}(i))
    case {1, 3, 4, 5, 6}
      previousline=maneuver{1}(i) ;
      previousinverted=maneuver{4}(i) ;
      found=true ;
    case 7
      % record loops angle
      if(maneuver{4}(i)==0)
        angle=angle+maneuver{2}(i) ;
      else
        angle=angle -maneuver{2}(i) ;
      end
  end
  i=i-1 ;

```

```

end
if (~found)
    previousline=1 ;
    previousinverted=0 ;
end
if (angle~=0)
    % lines reached from level with
    %      45 90 135 180 225 270 315 360
    getline=[3, 5, 3, 1, 4, 6, 4, 1 % pull
            4, 6, 4, 1, 3, 5, 3, 1 % push
            ] ;
    getinverted=[0, 0, 1, 1, 1, 0, 0, 0] ;
    index=round(abs(angle)/45) ;
    if (angle>0)
        dir=1 ;
    else
        dir=2 ;
    end
    offset=[0, 0, 1, 7, 2, 6] ;
    offset=offset(previousline)+4*previousinverted ;
    index=index+offset ;
    while(index>8)
        index=index-8 ;
    end
    previousline=getline(dir, index) ;
    previousinverted=getinverted(index) ;
end
switch(previousinverted)
case {-1, 3}
    previousinverted=0 ;
case 2
    previousinverted=1 ;
end
end
end

```