

# SACTA LIBRARY FOR MATLAB

## 1. INTRODUCTION

The SACTA library allows to access a transponder antenna to get information about air traffic and representing a radar presentation in a SACTA monitor or Google Earth.

### THE TRANSPONDER ANTENNA

The system gets the information from the signals of planes equipped with mode-S transponders (figure 1).

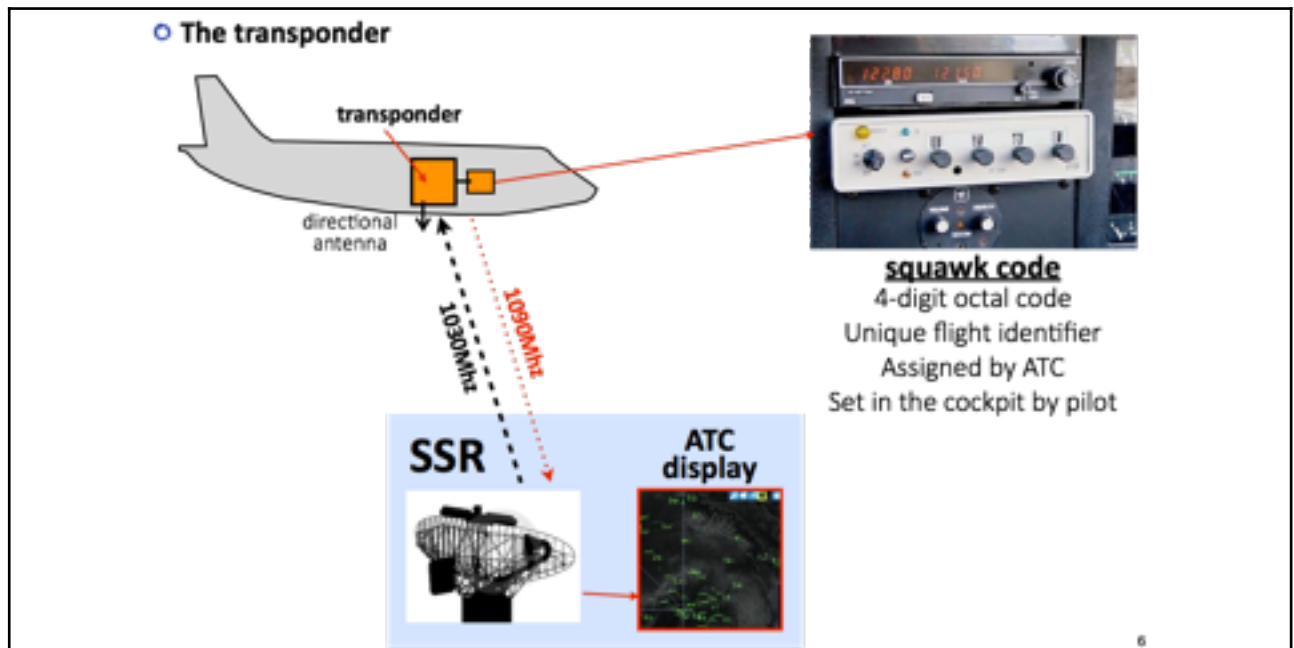


Figure 1: the transponder

Transponders transmit a response to an interrogation of a *Secondary Surveillance Radar* (SSR). But it also broadcasts an SQUITTER signal every two seconds with the information shown in figure 2. This response signal can be also freely captured with a transponder antenna.

The information provided by the transponders can be used to produce a *Radar Presentation* like the one of figure 3. The information of a mode-S transponder includes different data taken from the Flight Manager Computer (FMC) including longitude, latitude, altitude, ground speed, track angle, vertical rate, ICAO hex identifier, squawk code, and others.

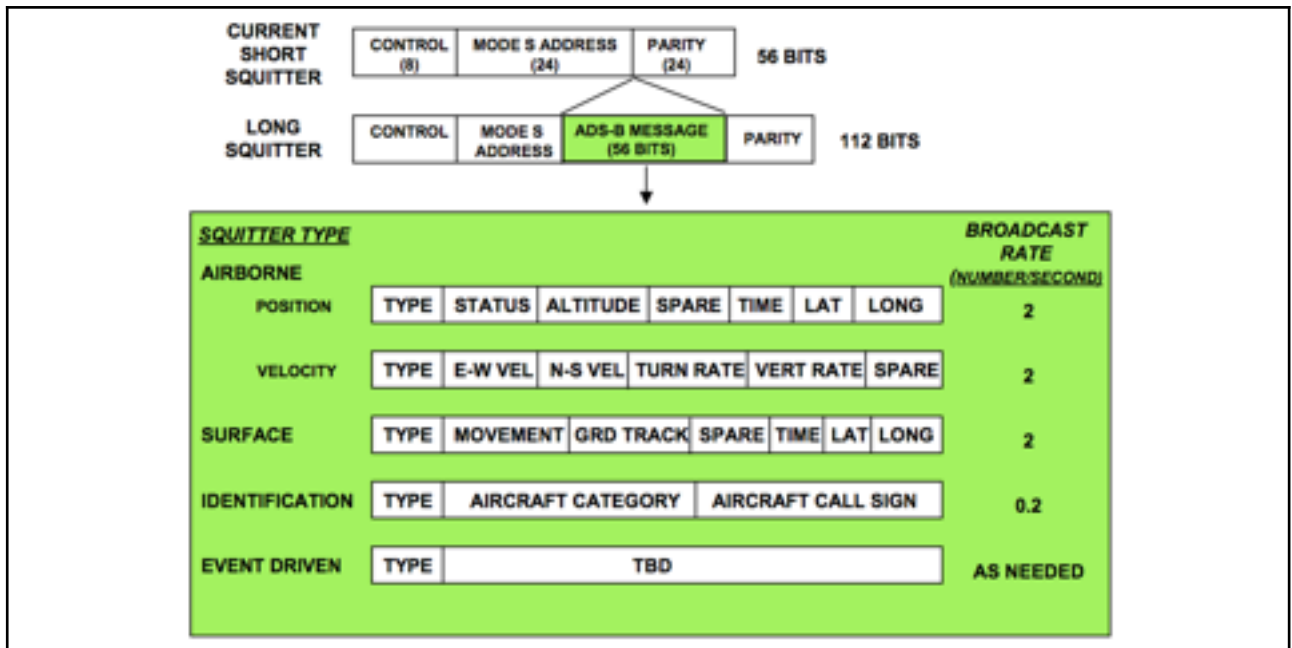


Figure 2: the transponder data frame

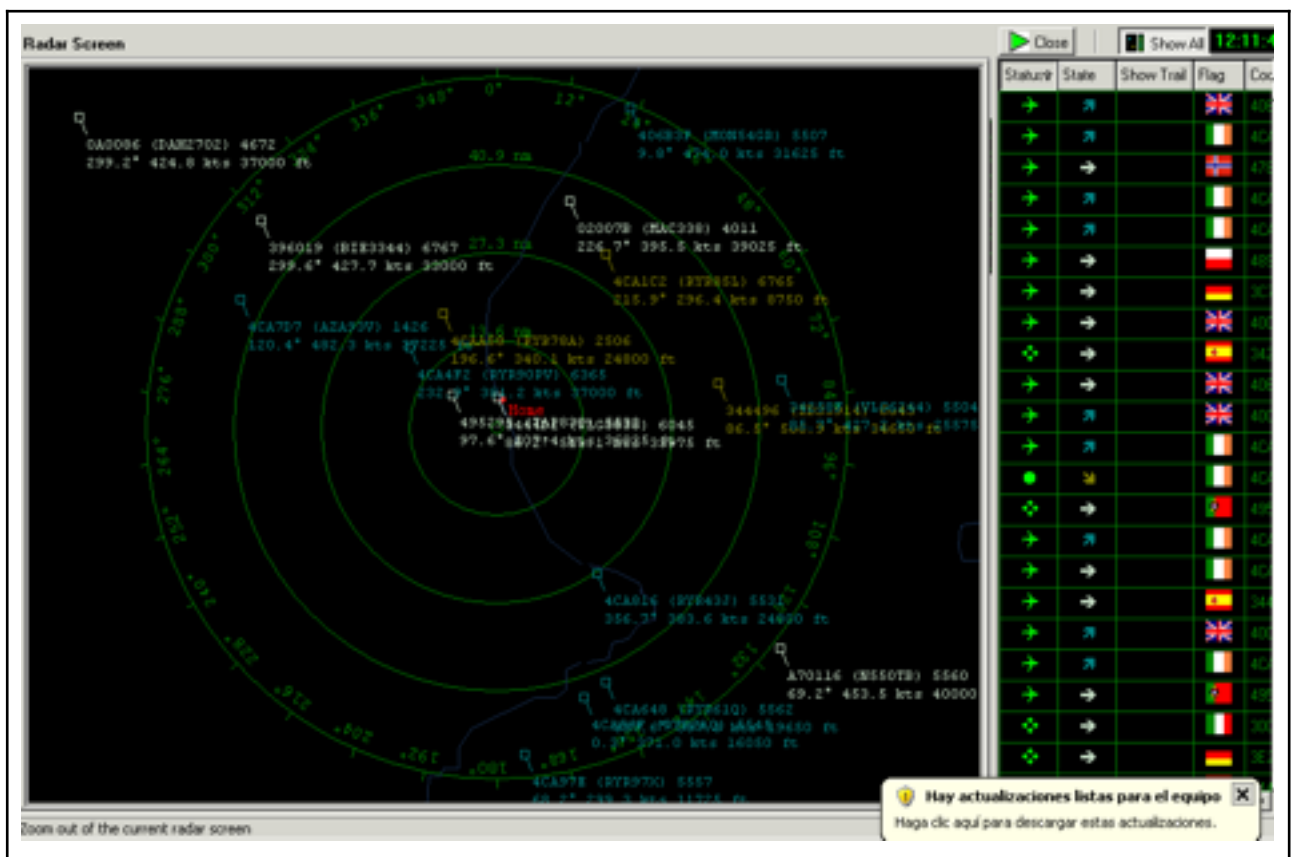


Figure 3: radar presentation

In the UPV there exists a computer equipped with an antenna that receives the transponder signal. This computer has a retransmitter program that distributes this information via TCP/IP to other computers in the UPVNET (UPV network).

This information is available inside the UPVNET network and also outside the UPV through the UPV VPN (Virtual Private Network). The host with the transponder antenna in our case uses the following IP address and port:

**host:** 158.42.40.150 , **port=**2220

### THE SACTA MONITOR

The SACTA monitor is a java executable file named **monitor.jar** contained in the SACTA directory. When it is executed it displays the two windows of figure 4.

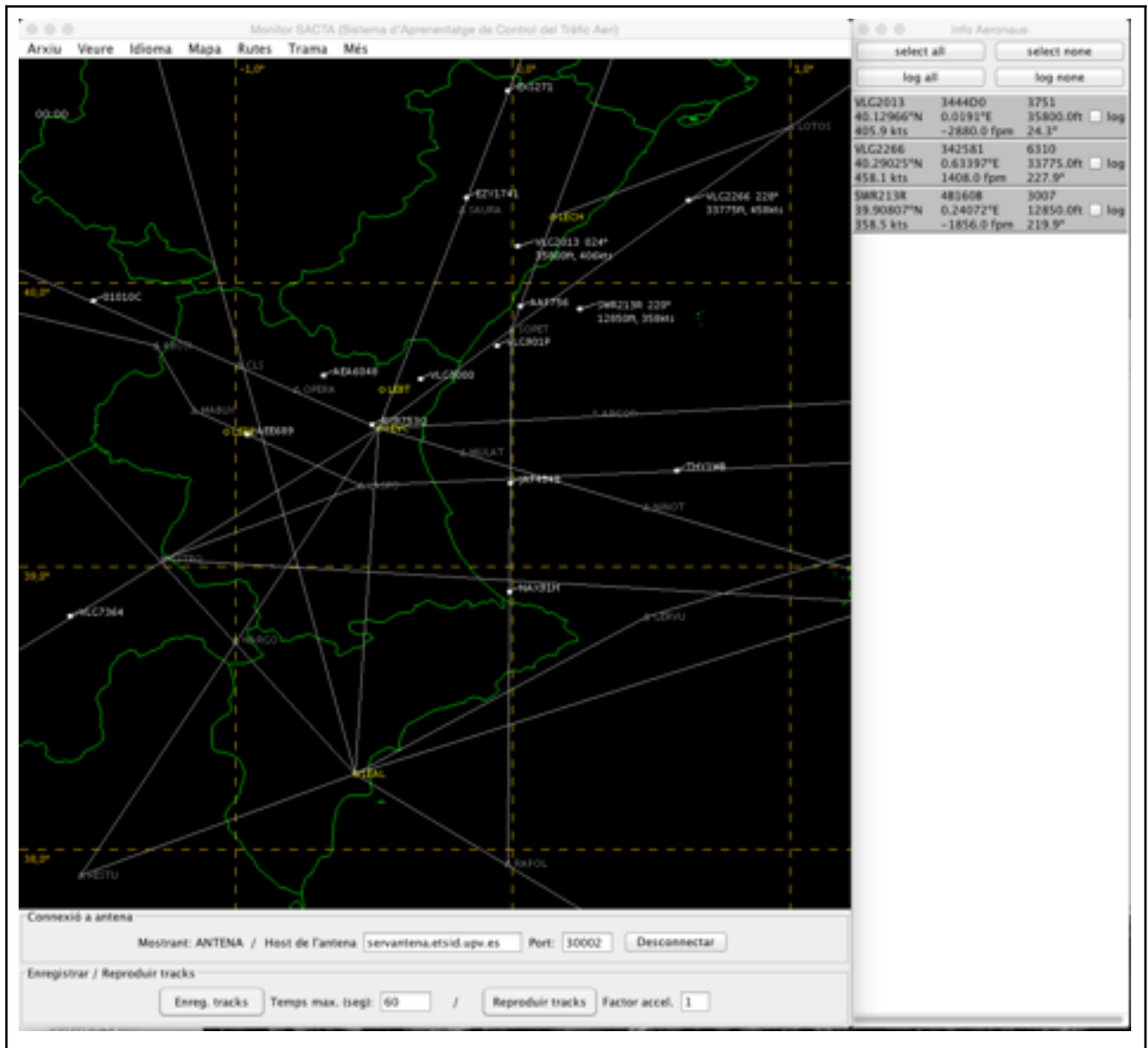


Figure 4: The SACTA monitor

One of the windows shows a list of traffics in text form. The other one presents it in graphically. Once it is started, the SACTA monitor is listening to MATLAB (or some other program) to send traffic information. However, using menu **"More"** and option **"Panel Antenna/Tracks"** you can display the bottom panels to get the traffic from some other data sources.

Panel “**Antenna connection**” allows to connect to a host with a transponder antenna and get real-time traffic.

Panel “**Record/Play tracks**” allows to record tracks in a .txt file or to play previously recorded tracks. Recorded traffic can be played using an accelerator factor.

In summary, you can get traffic information from three sources: a Matlab **program**, the transponder **antenna** or a previously recorded **track**. The source is indicated after the “Displaying” label in the “**Antenna connection**” panel. Recording traffic takes the traffic from the currently used traffic data source.

## 2. THE SACTA EDUCATIONAL SYSTEM

The SACTA educational system has the structure of figure 5. Basically, it is a system that allows MATLAB to get the traffic from a transponder antenna and represent it graphically on a map. The system consists of two main MATLAB libraries in jar format:

- I. **ANTENNA (radar.jar)**: it provides access to the transponder antenna in order to get traffic data.
- II. **MONITOR (monitor.jar)**: it provides access to a radar screen in order to represent traffics.

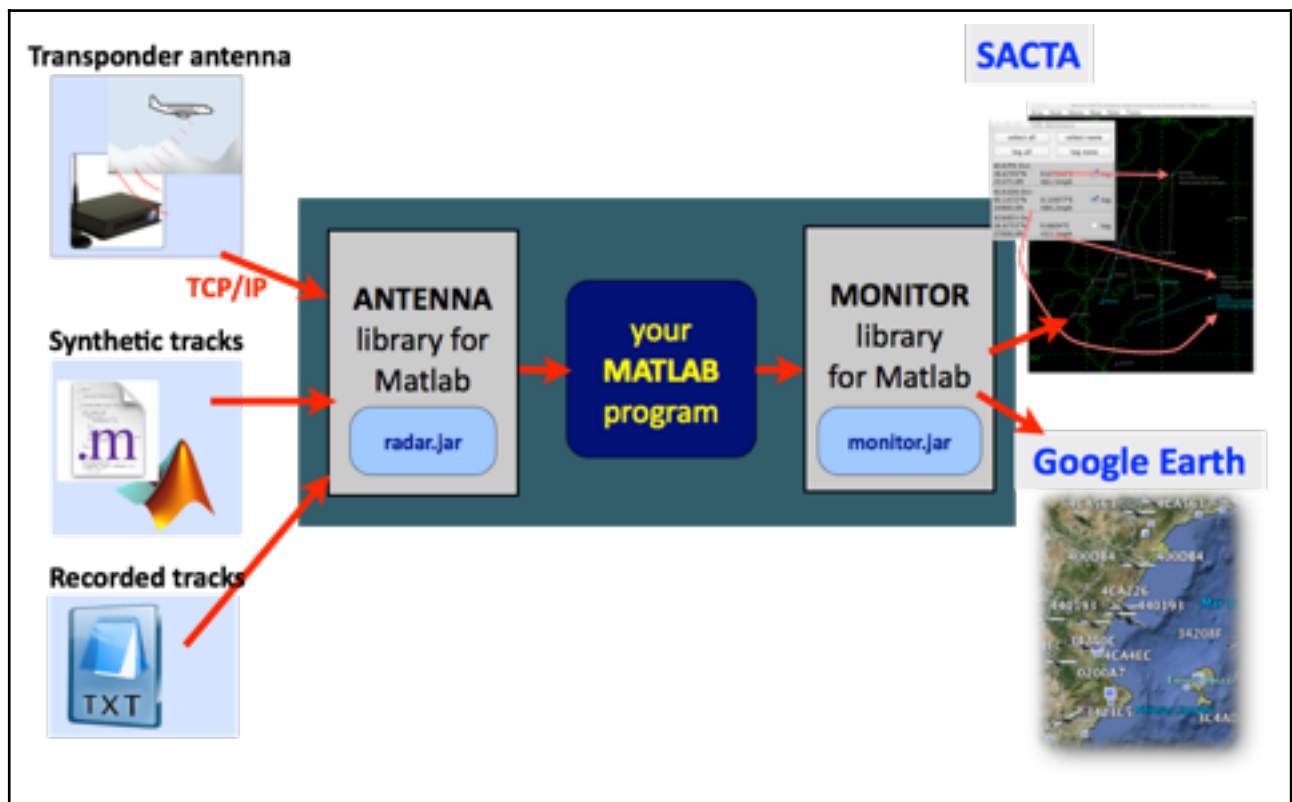


Figure 5: The SACTA system

The basic idea is to be able to write MATLAB programs (Your MATLAB program) that **read** traffic from the ANTENNA, **process** this traffic, and **display** this information in the MONITOR. For example, you can write a Matlab program to read traffic information from the ANTENNA, detecting potential collisions (this is processing) and show the conflicting traffics in the MONITOR in red color.

The SACTA system allows several possibilities for getting traffic data and for the *Radar Presentations* in the monitor:

### RADAR PRESENTATIONS

Three possible *Radar Presentations* or display options are available SACTA. Two of them are graphical and are shown in figure 6. The other one is in text form. These three options are:

- I. Graphical presentation on the **SACTA** monitor

2. Graphical presentation in **GoogleEarth**
3. Text presentation on the **MATLAB** console

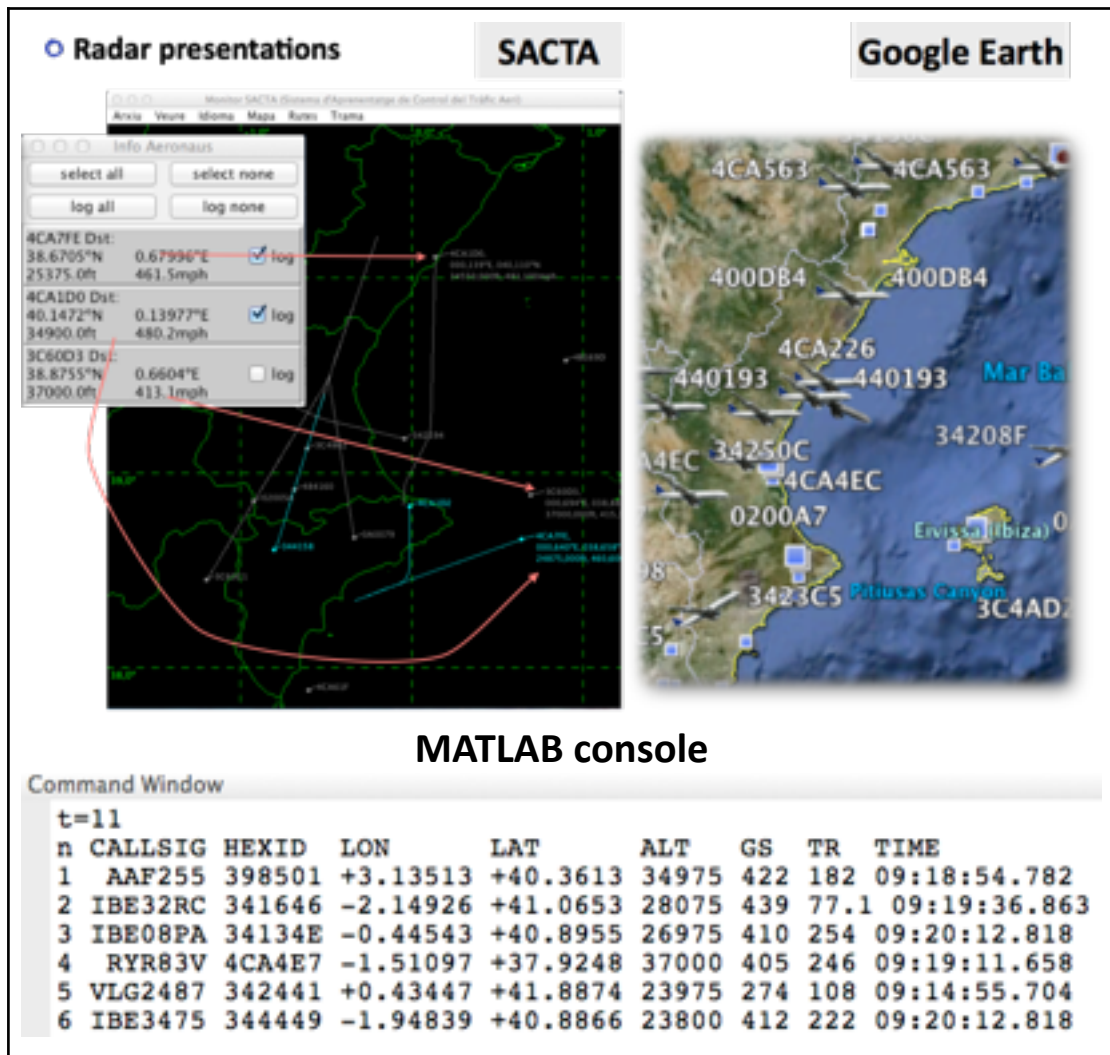


Figure 6: SACTA display options

This is easy to configure through library function **MONITOR\_Configure**. The visualization is performed in all cases using library function **MONITOR\_DisplayTraffic**.

## DATA SOURCES

Three possible data sources for air traffic are available SACTA. They allow to get different traffics: real-time, synthetic, perviously recorded. The traffic source is easy to configure through library function **ANTENNA\_Configure**. Reading the traffic is performed using library function **ANTENNA\_Read** in all cases.

The different options for the traffic sources are shown in figure 7. They are also described below:

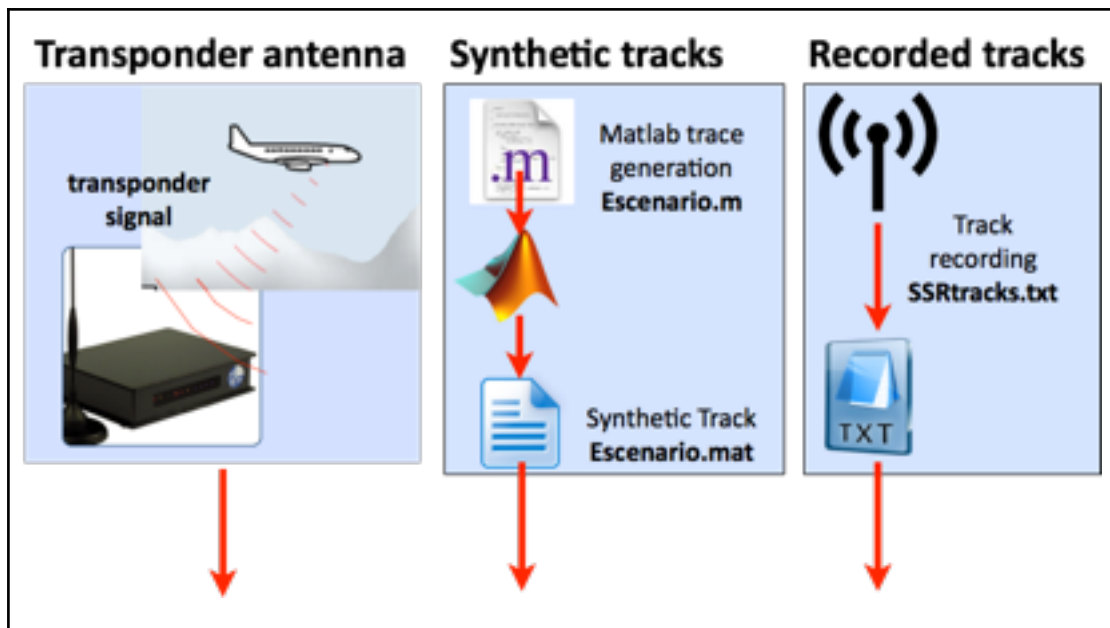


Figure 6: SACTA options for data sources

- a. The **Transponder antenna**
- b. **Synthetic tracks**
- c. **Recorded tracks**

The first option (Transponder antenna) provides real-time traffic in the Valencia area as detected by the transponder antenna.

The second option uses synthetic tracks that can be generated using a simple Matlab program. This is especially useful when the real-time traffic does not provide scenarios or situations that you want to test, as for example collision detection scenarios, which (thank God...) are highly improbable. So if you want to test this scenarios you can generate them synthetically using a Matlab program and saving it in a **.mat** file. This is an example:

```
% Generates a synthetic scenario with planes flying specified routes.

% 1 degree latitude = 110 km
% 1 degree longitude = 111 km
% 1 hour = 3600 s.
%   Some airways:
%   A33 = [ -3.17 40.22; -0.28 39.29; 2.45 39.26 ];
%   A34 = [ 0.33 42.41; -0.1 40.15; -0.19 39.53; -0.28 39.29; -0.34
38.16; 0.012 36.52];
%   B28 = [ 2.06 41.18; -0.28 39.29; -2.13 38.21];
%   G30 = [ -0.28 39.29; 1.28 38.54];
%   N609 = [ -0.012 36.52; -0.08 40.13; 0.059 42.46];

clc; clear all;
```

```

% Traffic through airway N609
plane(1).callsign = 'IB001';
plane(1).hexId    = 'A00001';
plane(1).aicraftId = 'B757';
plane(1).flightId = '9191';
plane(1).squawk   = '6001';
plane(1).color    = 1;
plane(1).trace = [-0.042, 38, 5000, 0 ;
                  -0.08, 40.13, 6000, 1200;
                  0.059, 42.46, 4000, 3000];

% Traffic through airway G30
plane(2).callsign = 'BMI702';
plane(2).hexId    = 'A00002';
plane(2).aicraftId = 'A319';
plane(2).flightId = '8151';
plane(2).squawk   = '6002';
plane(2).color    = 1;
plane(2).trace = [ -0.28 39.29, 0, 0 ;
                  -0.28 39.29, 0, 600 ;
                  -0.275 39.295 5000, 650 ;
                  1.28 38.54, 5000,1800 ];

% Traffic through airway A33   A33 = [ -3.17 40.22; -0.28 39.29; 2.45
39.26 ];
plane(3).callsign = 'RYR002';
plane(3).hexId    = 'A00003';
plane(3).aicraftId = 'B737';
plane(3).flightId = '6196';
plane(3).squawk   = '6003';
plane(3).color    = 1;
plane(3).trace = [ -3.17, 40.22, 10000, 0 ;
                  -0.28, 39.29, 10000, 1000;
                  2.45, 39.26, 10000, 2000 ];

% OVNI
plane(4).callsign = 'OVNI';
plane(4).hexId    = '????';
plane(4).aicraftId = 'OVNI';
plane(4).flightId = '0000';
plane(4).squawk   = '0000';
plane(4).color    = 3;
plane(4).trace = [ 1.5, 41, 1000, 0 ;
                  1, 38, 10000, 200 ;
                  0.5, 41, 1000, 300 ;
                  0 , 38, 10000, 400 ;
                  -0.5, 41, 1000, 500 ;
                  -1 , 38, 1000, 600 ;
                  -1.5, 41, 1000, 700 ;
                  2, 38, 1000, 800 ];

save 'scenario1' plane

```



The rule for generating a plane trace is creating an structure with the following mandatory fields: **callsign**, **hexld**, **aircraftld**, **flightld**, **squawk**, color and **trace**. the field trace defines the trajectory in 4D, i.e., 3D position and time. It consists of a matrix, with each row of the matrix defining a 4D point:

```
[longitude1, latitude1, altitude1, time1;  
  longitude2, latitude2, altitude2, time2;  
  etc.  
];
```

The system will generate a trace that goes through the 4D points at the specified times. The system will also linearly interpolate during the simulation intermediate points between two specified points of the trace.

### 3. THE ANTENNA LIBRARY FUNCTIONS

The ANTENNA library contains the following functions:

```
ANTENNA_Configure(source, simspeed, ip_file, protocol, port)
```

```
[traffics, tsim, real_time] = ANTENNA_Read()
```

These functions are described next.

```
function ANTENNA_Configure(source, simspeed, ip_file, protocol, port)
```

It configures the radar to get data from the following sources:

- The transponder antenna which is connected via TCP/IP
- .mat files: synthetic traces generated using Matlab
- .txt files: previously recorded tracks from the transponder

#### INPUT PARAMETERS

**source:** string that specifies the kind of source for traffic data. The two following strings are possible

- 'file' for simulating a track recorded in a .txt or .mat file
- 'ip' for a transponder antenna via TCP/IP

**simspeed:** Factor used to accelerate simulation traces. It does not affect real-time transponder antenna

**ip\_file:** string that specifies the specific file or ip address for the traffic source. Examples:

```
if source=='ip' -> ip_file= '158.42.40.150'
```

```
ip_file= '226.1.1.1' for multicast
```

```
if source=='file' -> ip_file= 'e1.mat' or
```

```
ip_file= 'SSRtracks_2015_enero_29-05-44.txt'
```

**protocol:** TCP/UDP protocol used to connect to the transponder antenna. This parameter is not necessary for simulating file recorded tracks. Usual value is 3, which corresponds to TCP raw. All possible values are:

0 - TCP (Java Objects) (incompatible Freemat)

1 - UDP (Java Objects)(incompatible Freemat)

2 - Multicast

3 - TCP (char) (server compatible Freemat)

4 - UDP (char) (server compatible Freemat)

**port** : integer to specify the antenna port. This parameter is not necessary for simulating file recorded tracks. The usual port is 2220 for TCP and 2221 for multicast.

## RETURN VALUES

none

```
function [traffics, tsim, real_time] = ANTENNA_Read()
```

It reads the information from the traffic source established by **ANTENNA\_Configure** and returns the corresponding traffic information. This information consists of an array of planes detected by the radar. It delays one second two successive reads from the traffic source.

## INPUT PARAMETERS

none

## RETURN VALUES

**tsim**: a double specifying the simulation time elapsed from the start of the simulation. If the simulation is accelerated by a factor of A, the returned value is the previous value + A\*dt.

**real\_time**: It is current time for the transponder and matlab sythetic traces. For recorded tracks it returns the time when the traffic was recorded.

**traffics**: an array of traffics or planes detected by the transponder. Each traffic is represented a struct of the form

**traffics(i).field**

where the following fields are provided:

Field	Type	Description
=====		
<code>traffics(i).aircraftId</code>	<i>string</i>	Database Aircraft record number.
<code>traffics(i).flightId</code>	<i>string</i>	Database Flight record number.
<code>traffics(i).hexId</code>	<i>string</i>	Aircraft Mode S hexadecimal code.
<code>traffics(i).callsign</code>	<i>string</i>	An eight digit flight ID. It can be a flight number or regist. (or even nothing)
<code>traffics(i).squawk</code>	<i>string</i>	Assigned Mode A squawk code.
<code>traffics(i).lon</code>	<i>double</i>	Longitude in degrees. East positive, West negative.
<code>traffics(i).lat</code>	<i>double</i>	Latitude in degrees North positive, South negative.
<code>traffics(i).alt</code>	<i>double</i>	Mode C altitude in feet. Height relative to 1013.2mb in feet. (Flight Level). Not height AMSL..
<code>traffics(i).gspeed</code>	<i>double</i>	GS Speed over ground in NM (not indicated airspeed)
<code>traffics(i).track</code>	<i>double</i>	Track of aircraft in degrees (not heading). Derived from the velocity E/W and N/S.
<code>traffics(i).vertRate</code>	<i>double</i>	Vertical rate in fpm. 64ft resolution.
<code>traffics(i).alert</code>	<i>string</i>	Flag to indicate squawk has changed.
<code>traffics(i).emergency</code>	<i>string</i>	Flag to indicate emergency code has been set.
<code>traffics(i).SPI</code>	<i>string</i>	Flag to indicate transponder Ident has been activated.
<code>traffics(i).isOnGround</code>	<i>string</i>	Flag to indicate ground squat switch is active.
<code>traffics(i).date</code>	<i>string</i>	Date when message was generated.
<code>traffics(i).time</code>	<i>string</i>	Time when message was generated.
<code>traffics(i).color</code>	<i>double</i>	An integer in range [1..9] to set the color for radar presentation. See README.colors.
<code>traffics(i).icon</code>	<i>string</i>	'avion.png' Icon file for Google Earth representation
<code>traffics(i).comments</code>	<i>string</i>	Comments for Google Earth representation

## 4. THE MONITOR LIBRARY FUNCTIONS

The MONITOR library contains the following functions which are common to all monitors.

```
iRet = MONITOR_Configure(mode, beam)
```

```
MONITOR_DisplayTraffic(traffics, t)
```

These functions are described next.

```
function status = MONITOR_Configure(mode, beam)
```

It establishes the monitor that will be used for the radar presentation. The refreshing period of the information in the monitor is 1 second. It also tries to launch the selected monitor automatically. If due to some problem the monitor could not be automatically launched, please execute the following command to launch it:

```
SACTA/monitor.jar.
```

### INPUT PARAMETERS

**mode:** it is double specifying the monitor:

- 1: SACTA monitor
- 2: GOOGLE EARTH representation
- 3: CONSOLE table

**beam:** it simulates a rotating radar beam on SACTA if this is enabled in the corresponding menu of the monitor: **Show->Beam**. Just for fun... This is not really a primary rada so the beam makes no sense, ...

### OUTPUT PARAMETERS

**status** indicates if the corresponding monitor was successfully launched:

- **~= -1:** success
- **-1:** failed

```
function MONITOR_DisplayTraffic(traffics, t)
```

It displays a traffic array and the simulation time in the monitor specified by **MONITOR\_Configure** (SACTA/Google Earth/Matlab console).

### INPUT PARAMETERS

**t**: Time to be displayed.

**traffics**: an array of structs of the form **traffics(i).field** where a number of fields are mandatory for each monitor type. It is recommended to use an array of structs as the one returned by **ANTENNA\_Read**.

*Mandatory fields for ALL monitors: **hexId, callsign, squawk, lon, lat, alt, gspeed, track, time**.*

*Additional mandatory fields for SACTA and Google Earth: **vertRate, color***

*Additional mandatory fields for Google Earth: **icon, comments***

## 5. THE SACTA LIBRARY FUNCTIONS

The SACTA library contains the functions which are specific to the SACTA monitor:

```
status = SACTA_DisplayWaypoints(wpts)
status = SACTA_DisplayLines(lines,color)
status = SACTA_DisplayRoutes(pRoutes)
status = SACTA_DisplayTraffic(t, traf , beam, accFactor)
status = SACTA_DisplayMap(pMap)
status = SACTA_DisplayGrid(pLon, pLat)
status = SACTA_SetCoord(pLonW, pLonE, pLatN, pLatS)
status = SACTA_Ping()
```

These functions are described next.

```
function SACTA_DisplayWaypoints(wpts)
```

It displays a set of waypoints in the SACTA monitor:

### INPUT PARAMETERS

**wpts**: an array of waypoints. A waypoint is an struct with the following fields:

*wpts(i).name* String

*wpts(i).Lon* double

*wpts(i).Lat* double

*wpts(i).color* integer in range [1..10]. See README.colors

*wpts(i).icon* String

### RETURN VALUES

It returns an "0" if success and different error codes otherwise.

```
function status = SACTA_DisplayLines(lines,color)
```

It draws a set of lines in the SACTA monitor:

### INPUT PARAMETERS

**lines**: this parameter is defined by a cell array of lines:

```
lines={line1, line2, line3}
```

Each line is defined by two points:

```
line1 = [ lon1, lat1; lon2, lat2];
```

```
line2 = [ lon3, lat3; lon4, lat4];
```

```
line3 = [ lon5, lat5; lon6, lat6];
```

**color:** a numeric value in the range [1..10]. See README.colors.

## RETURN VALUES

It returns an "0" if success and different error codes otherwise.

```
function SACTA_DisplayRoutes(pRoutes)
```

It draws a set of airways in the SACTA monitor.

## INPUT PARAMETERS

**pRoutes:** is a cell array of routes. Each cell consists of an array of size Nx2 where N is the number of waypoints that define the route. Each row of the array defines a waypoint of the route and it consists of two columns: longitude and latitude.

*Example:*

```
route1 = [ -0.5 41.0; 0.0 39; 0.5 38 ];
```

```
route2 = [ 2 39.1; 0 39.1];
```

```
route3 = [ 0 39.1; -2 39.5];
```

```
routes = { route1 route2 route3 };
```

## RETURN VALUES

It returns an "0" if success and different error codes otherwise.

```
function SACTA_DisplayTraffic(t, traf, beam, accFactor)
```

It displays the traffic specified in the traffic array "traf" and the simulation time. It also represents the radar beam on the monitor if beam=true

## INPUT PARAMETERS

**t:** Time to be displayed.



**traf:** an array of structs of the form `traffics(i).field` where a number of fields are mandatory for each monitor type. It is recommended to use an array of structs as the one returned by **ANTENNA\_Read**.

**beam:** Boolean indicating whether the radar beam should be represented. The Show -> Beam option on SACTA menu bar must be checked.

**accFactor:** Factor to accelerate the beam accordingly to the simulation speed.

## RETURN VALUES

It returns an "0" if success and different error codes otherwise.

```
function SACTA_DisplayMap(pMap)
```

It draws a map in the SACTA monitor.

## INPUT PARAMETERS

**pMap:** a two column array (longitude, latitude) of points. An Inf value denotes that the previous coordinate is not linked to the next one. In other words, lifting the pencil from the paper.

### Example

Inf Inf

-1.000045 37.584565

-1.002392 37.584565

-1.003272 37.585445

-1.006792 37.585445

Inf Inf

-1.007672 37.586325

-1.007966 37.585738

-1.007966 37.584858

## RETURN VALUES

It returns an "0" if success and different error codes otherwise.

```
function SACTA_DisplayGrid(pLon, pLat)
```

It sets a grid of meridians and parallels on the SACTA monitor.

## INPUT PARAMETERS

**pLon:** array with the longitudes of each meridian

**pLat:** array with the longitudes of each parallel

*Example*

```
LonGrid = [-2.0 -1.0 0 1.0 2.0];
```

```
LatGrid = [37.0 38.0 39.0 40.0 41.0];
```

**RETURN VALUES**

It returns an "0" if success and different error codes otherwise.

```
function SACTA_SetCoord(pLonW, pLonE, pLatN, pLatS)
```

It sets the coordinates of the limits of the SACTA monitor:

**INPUT PARAMETERS**

**pLonW:** longitude of the WEST limit

**pLonE:** longitude of the EAST limit

**pLonN:** latitude of the NORTH limit

**pLonS:** latitude of the SOUTH limit

**RETURN VALUES**

It returns an "0" if success and different error codes otherwise.

```
function SACTA_Ping()
```

It sends a ping to the SACTA monitor to check it is alive.

**RETURN VALUES**

It returns an "0" if the monitor is alive.

## 6. THE TRACK LIBRARY FUNCTIONS

The MONITOR library contains the following functions for writing and reading tracks in .txt file.

The format of a track file is:

```
Date
Time #_of_traffics
traffic1
traffic2
traffic3
...
Time #_of_traffics
traffic1
traffic2
traffic3
...
```

The fields for each traffic line are:

<i>aircraftId</i>	<i>flightId</i>	<i>hexId</i>	<i>callsign</i>	<i>squawk</i>	<i>Lon</i>	<i>Lat</i>	<i>alt</i>
<i>gspeed</i>	<i>track</i>	<i>vertRate</i>	<i>alert</i>	<i>emergency</i>	<i>SPI</i>	<i>isOnGround</i>	<i>date</i>
<i>time</i>							

See function **ANTENNA\_Read** for a description of these fields.

### Example:

05-mar-2015

10:34:49.766 3

20844 2448976 344548 VLG8461 4774 -2.63703 40.97286 38000.0 406.2 75.9 192.0 0 0 0 0 2015/03/05 10:34:42.958

63 2448949 4B1A5E EZS98GE 5531 2.35773 39.17422 38000.0 434.2 70.8 -64.0 0 0 0 0 2015/03/05 10:34:43.973

486 2448934 0A001B 0A001B 5543 1.26299 40.48034 36000.0 338.9 353.9 64.0 0 0 0 0 2015/03/05 10:34:42.598

10:34:50.771 4

20844 2448976 344548 VLG8461 4774 -2.63703 40.97286 38000.0 406.2 75.9 192.0 0 0 0 0 2015/03/05 10:34:42.958

63 2448949 4B1A5E EZS98GE 5531 2.35991 39.17487 38000.0 434.2 70.8 -64.0 0 0 0 0 2015/03/05 10:34:43.973

486 2448934 0A001B 0A001B 5543 1.26287 40.48127 36000.0 338.9 353.9 64.0 0 0 0 0 2015/03/05 10:34:42.598

16064 2448970 344297 AEA2662 3722 -1.03343 38.68726 36000.0 368.6 28.2 0.0 0 0 0 0 2015/03/05 10:34:42.598

```
function TRACK_write(file,traffic,time)
```

It appends a line to a trace file with the data at time `t` of a traffic array that is passed as an argument.

### INPUT PARAMETERS

**file**: an output .txt file located in the SACTA/tracks directory

**time**: simulation time

**traffic**: traffic structure as defined in by **ANTENNA\_Read**.

```
function [traffics, tsim, real_time]=TRACK_read(file)
```

It reads an instant of time of recorded set of tracks from a .txt file. and returns the traffic information, i.e., a set of planes detected by the antenna. The first call opens the file, reads the information for the first instant of time and leaves the file open. Subsequent calls read the following instants of time, until the eof is reached. The file is closed and **tsim** returns **intmax**.

### INPUT PARAMETERS

**file**: Name (string) of the .txt file of the recorded track

### OUTPUT PARAMETERS

**traffics** : array of traffics for the instant of time `real_time`. See **ANTENNA\_Read** for a description of such array.

**real\_time** : time when the trace was recorded.

**tsim**: simulation time for which the traffic read has to be performed.

```
function i=traffic_find(traffics,hexcode)
```

It looks for a traffic with a specified hexcode in an array of traffic structs

### INPUT PARAMETERS

**traffics**: array of traffic structs

**hexcode**: hexcode to look for.

### OUTPUT PARAMETERS

**i**: index of the traffic we are looking for. The value `i=0` if not found.

## 7. EXAMPLES OF USE

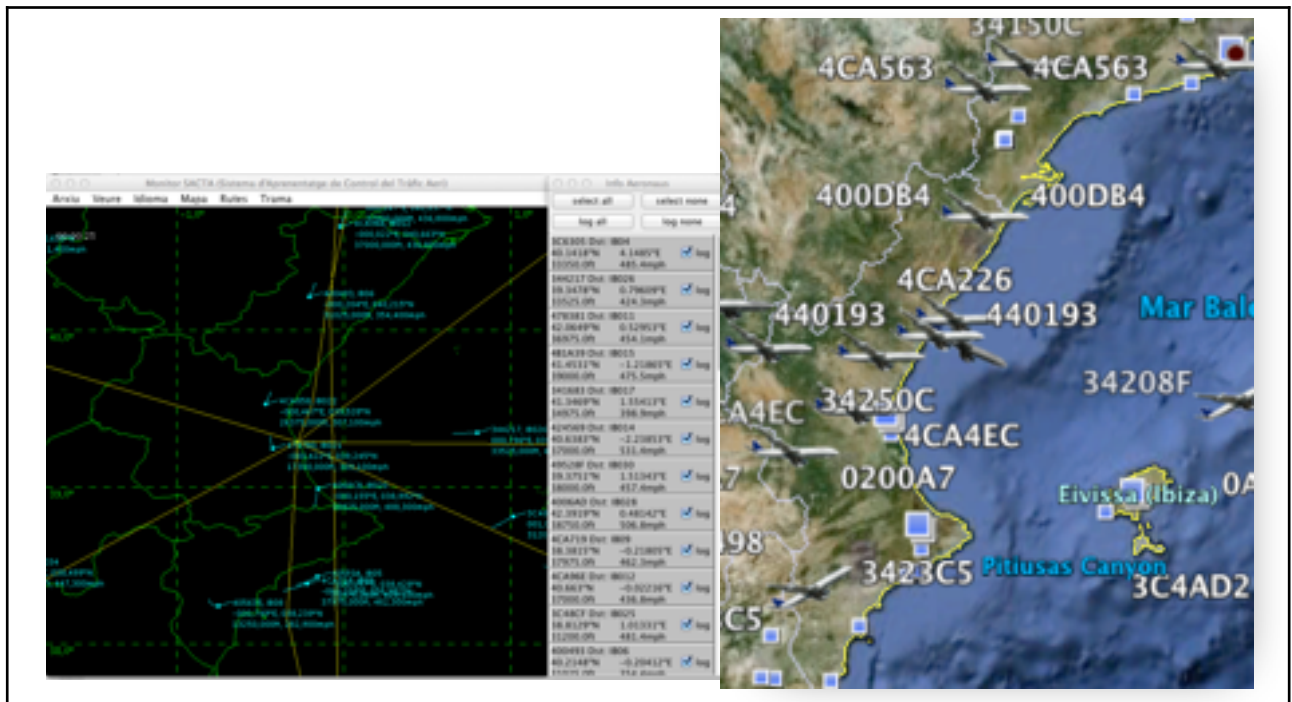
The following examples show different applications of the SACTA library. Examples are thoroughly commented so they are self-explicative. Source code for each example is provided in the SACTA distribution.

---

### MAIN\_1: READ ANTENNA TRAFFIC, PROCESS IT, AND DISPLAY IT

---

This example introduces how to use the basic ANTENNA and MONITOR library functions. It configures the ANTENNA and then it also configures the MONITOR to display in one of the available monitors, as shown below:



The SACTA monitor or Google Earth are automatically started. It then performs a loop where it reads the antenna traffic data and it displays it in the selected monitor.

Before displaying the traffic information in the monitor, this information can be processed using functions like:

```
[traffics]=PROCESS_UpperLower(traffics);  
[traffics]=PROCESS_ClimbDescend(traffics);  
[traffics]=PROCESS_Collision(traffics);
```

Function **PROCESS\_UpperLower** processes the traffic array in such a way that traffics in upper airspace are colored in cyan and traffics in lower airspace are colored in yellow. This implies changing the attribute `traffics(i).Color`.

Function **PROCESS\_ClimbDescend** processes the traffic array in such a way that traffics climbing are colored in red, traffics descending are colored in green, and traffics leveled are colored in cyan. This also implies changing the attribute `traffics(i).Color`. There are two methods to

find out if a traffic is climbing or descending. The simplest one consists of examining the sign of the **vertRate** attribute. The other one is based on memorizing the traffic array in a previous instant of time using a persistent variable and comparing the values of the **alt** attribute.

Function **PROCESS\_Collision** detects potential collisions and colors the traffics properly. Red color means collision alert. Based on checking the horizontal distance and vertical distance between every pair of traffics. The security cylinder has a horizontal radius of 5NM and a height of 1000 ft. Use the synthetic tracks of **scenario3.mat** to get a scenario with potential collisions.

```
function MAIN_1
% 1) READ ANTENNA TRAFFIC
% 2) PROCESS IT (optional) AND
% 3) DISPLAY IT ON:
%    1.- SACTA, 2.- GOOGLE EARTH OR 3.-CONSOLE

clc; clear all; close all;
addpath('SACTA','lib/kml','lib/geo');

%-----
% Variables and constants
%-----
%% ANTENNA PARAMETERS
ANTENNA='ip';           % SELECT ANTENNA
IP_FILE='158.42.40.150'; % IP of the host with the antenna
PORT=2220;              % Port of the antenna host
PROTOCOL= 3;           % TCP(char)=3 UDP(char)=4
%                       % Multicast=2 IP_FILE='226.1.1.1' PORT=2221

%% SIMULATION FILES
%ANTENNA='file';       % SELECT FILE
%IP_FILE = 'scenario3.mat';           %SYNTHETIC TRACE
%IP_FILE = 'SSRtracks_2015_enero_29-05-44.txt'; %RECORDED SCENARIO
%PORT = '';
%PROTOCOL=0;
%
SIM_SPEED = 1;         % Factor used to accelerate simulation traces
%
%% MONITOR PARAMETERS
MONITOR= 1;           % SACTA=1 Google_Earth = 2 Console=3;
BEAM= true;           % Paint the rotating beam on the screen. Just for fun.
% It does not affect real-time transponder antenna
t=0;                  % Time elapsed since the beginning of the program
TMAX=2000;            % Max (real or simulated) time executing the program
```

```

%-----
%% ANTENNA configuration
% Set traffic_data_source = ANTENNA_IP or TRACE_FILE .mat
%-----
ANTENNA_Configure(ANTENNA,SIM_SPEED,IP_FILE,PROTOCOL,PORT);

%-----
%% MONITOR configuration
% MONITOR = SACTA, GOOGLE EARTH, or CONSOLE
%-----
if MONITOR_Configure(MONITOR,BEAM) ~= 0
    fprintf(2,'\n Monitor not started\n');
    return;
end

%-----
%% PERIODIC LOOP
%-----
while t<TMAX
    %-----
    % READ the ANTENNA
    %-----
    [traffics, t, real_time] = ANTENNA_Read();
    fprintf('t=%s\n',real_time);

    %-----
    % PROCESS the information from the ANTENNA
    %-----
    % Process traffic array (a separated function is recommended to do
that)
    % Uncomment the desired processing action
    %[traffics]=PROCESS_UpperLower(traffics);
    %[traffics]=PROCESS_ClimbDescend2(traffics);
    %[traffics]=PROCESS_Collision(traffics);

    %-----
    % Display Traffic in MONITOR
    %-----
    MONITOR_DisplayTraffic(traffics, t);

end

end

```

---

## MAIN\_2: WRITE TRACKS IN A FILE

---

This example introduces how to save tracks in a file. It based on the use of library function **TRACK\_write**.

The example saves all traffic tracks in a file:

```
file=['SSRtracks-',date, '.txt']; %track file for all traffics
```

and also selects a single traffic and saves its track in another file:

```
file2=[hex, '-',date, '.txt']; %track file for one traffic
```

Filtering a single traffic is performed using function **traffic\_find**.

Note that a traffic with a given hexcode may appear in different positions of the array traffics in two consecutive readings of the antenna. This function can be used to look for the new position.



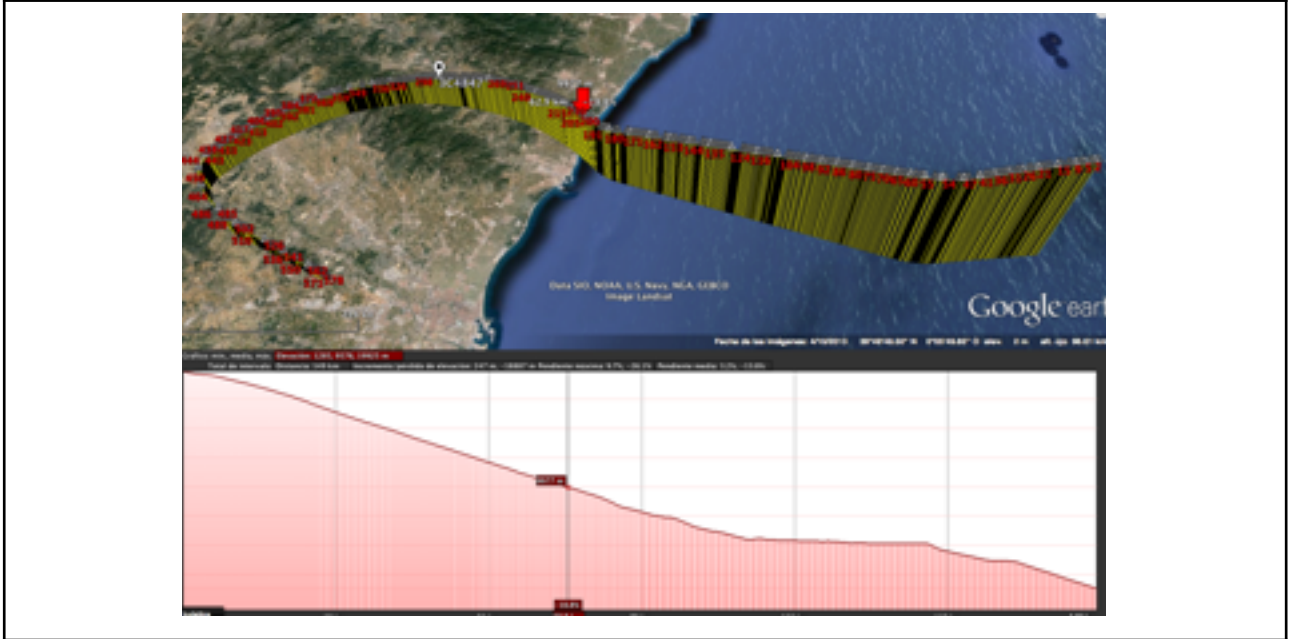
---

### MAIN\_3: READ A TRAFFIC TRACK AND REPRESENT IT IN GOOGLE EARTH

---

This example introduces the use of function **TRACK\_read**. This function is used to read a trace previously stored in a file.

Google Earth representation is based on the use of the **kmlwrite\_polyline** library function. See library **lib/kml**.



---

## MAIN\_4: DETECT AND LOG MANISES DEPARTURES AND ARRIVALS


---

This example is very much similar to **MAIN\_1**. The only remarkable difference is the use of function **PROCESS\_DepArr** to guess the departures and arrivals from Valencia airport. Traffics are colored in different ways according if they are departures, arrivals or they are overflying the airport.

```
- Arrivals to Manises:
- traffics(i).Color = 'cyan'
- traffics(i) is included in set arrivals

- Departures from Manises:
- traffics(i).Color = 'red'
- traffics(i) is included in set departures

- Rest of the traffic
  > The color is not modified or initialized to:
  > traffics(i).Color = 'gray'
```



Function **PROCESS\_DepArr** identifies Departures and Arrivals to an Airport and represents them in different colors: departures in red and arrivals in cyan.

It returns:

**DepTraffic:** array of departure traffics

**ArrTraffic:** array of arrival traffics

**traffics** : array of all traffics with the color attribute properly set

This function needs two phases to detect departures and arrivals.

**1st PHASE:**

*DEPARTURE candidate:* Altitude < 3000 && abs(Longitude-Airport.lon)<0.06 && abs(Latitude-Airport.lat)<0.06

*ARRIVAL candidate:* Altitude < 20000 && abs(Longitud-Airport.lon)<1.2 && abs(Latitud-Airport.lat)<1.2

## **2nd PHASE:**

DEPARTURE:

- a) Departure candidate && Altitude increases && Distance increases
- b) Departure &&  $\text{abs}(\text{Longitude-Airport.lon}) < 1.2$  &&  $\text{abs}(\text{Latitude-Airport.lat}) < 1.2$

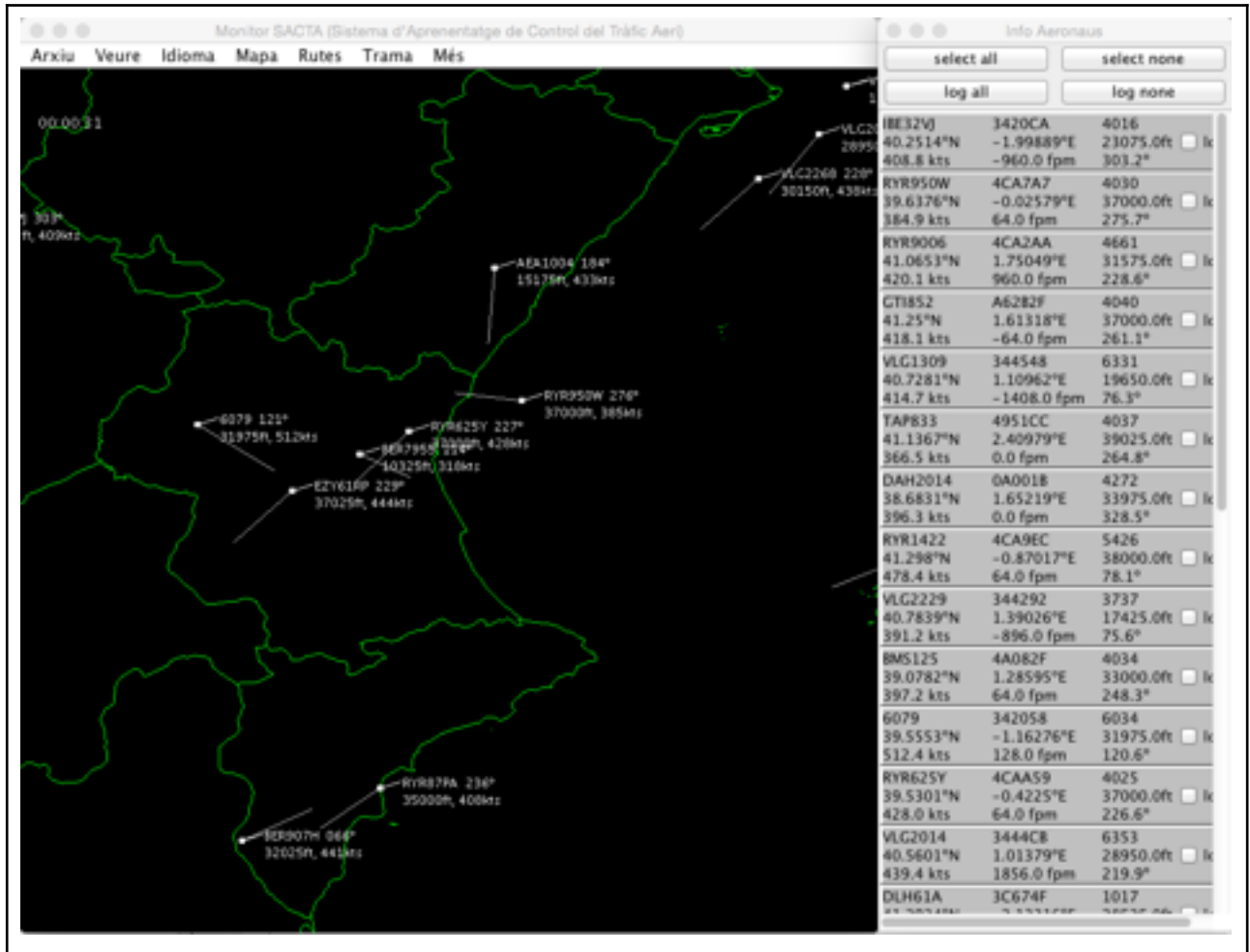
ARRIVAL

- a) Arrival candidate && Altitude decreases && Distance decreases
- b) Arrival &&  $\text{abs}(\text{Longitude-Airport.lon}) > 0.6$  &&  $\text{abs}(\text{Latitude -Airport.lat}) > 0.6$

## MAIN\_5: DRAW FLIGHT VECTORS IN SACTA MONITOR

This example draws the flight vectors of each traffic in order to easily estimate future estimated positions.

It introduces the use of function **SACTA\_DisplayLines** to draw the vectors.



Function **PROCESS\_Vectors** calculates the flight vectors (*sable*) of an array of traffics and returns it as cell array of lines. The flight vector (*sable*) is a line that goes from the nose of the plane to the calculated future position at a time  $t_{future}$  from current time  $t$ . This position is calculated by extrapolation using the ground speed, the vertical speed and the track angle.

---

## MAIN\_6: DRAW ROUTES AND FIXES IN SACTA MONITOR

---

This example draws the airways and most important waypoints in the Valencia airspace.

It introduces the use of function **SACTA\_DisplayRoutes** to draw the airways.

It introduces the use of function **SACTA\_DisplayWaypoints** to draw the waypoints.

