MTRN4010

Description of some relevant API functions, that are used in the example program.

This document is consistent with the API version **API4010_v04**
We may add new functions to the API, to help students, in subsequent versions.
That is why you may see other APIs having names **API4010_vNN*, in which NN >04.***

FUNCTIONS:

ok=API.e.SelectDataset(ThisDataset);
Select dataset to be played back.
ThisDataset: path to data (folder specification.) It is a 'char' type, e.g. ***'M:\myData\G02r\'***

e.g., ***API.e.SelectDataset('M:\myData\G02r\');***

[id , r, ti]= API.GetEvnt()

Read next event (usually a measurement from a sensor)
It returns 3 variables.

id: source of the event (class of event , sensor's ***id***).
Usual cases:
id=1 : IMU measurement
id=2 : RGB image
id=3 : Depth
There are other ones. Read Demo01 and Dmeo02 example programs to see how are used.

r : data of that event (the data structure depends on the type of event (see the example programs to see the way ***r*** is used)
ti: timestamp of that event (expressed as ***uint32***. In which 1 unit represents 0.1ms. Usually, you need to express the time in seconds, using real numbers (not uint32). The is why we usually do *"t=double(ti)/10000;"*

[xx,yy,zz]=API.e.Depths2pts(DepthImage , flag)
From a depth image, we obtain the associated 3D points cloud.
DepthImage: depth image from camera (a 240 rows by 320 columns matrix).

flag:
if flag==1 : the *faulty pixels* that may be in the depth image will not be included in the resulting 3D points cloud.
if flag==0 : faulty pixels will be included in the resulting points cloud, having value (0,0,0) each of the faulty pixels. The "good pixels" are not affected by this flag.
Faulty pixels are those for which the camera was not able to estimate depth.

Returns: ***[xx,yy,zz]***, three arrays, having the x,y and z coordinates, respectively, of the set of 3D points.

[xx2,yy2]=API.e.Rotate2D(xx,yy, angle);
Applies a 2D rotation on the points **xx,yy;**
this function assumes that ***angle*** is expressed in radians.

----

r=API.b.SelectROI(figuRGB);
Allows user to interactively (using the mouse or other pointing device) select, from the figure whose figure number is specified by ***figuRGB*** , a rectangular region ("region of interest", ROI).
It returns a structure, ***r***.
r.ok: if r.ok =1, the user has selected a ROI. If not, the ROI is invalid or non-existent (the user chose a nonsensical ROI, or because the user aborted the process.
If ok=1: then the field ***pp*** will contain the ROI coordinates.
Usually, the specified figure does contains a RGB image or a Depth image.

[ok,xx,yy,zz]=API.b.Gt3DPtsFromROI(u12,v12,Depth,flag);
Get the 3D points that correspond to the pixels contained in a ROI
It is usually used with ROIs obtained using ***API.b.SelectROI();***
u12 specifies the interval of pixels, horizontally, e.g. [u1,u2] ([firstColumm, lastColumn]
v12 specifies the interval of pixels, vertically, e.g. [v1,v2] ([firstRow, lastRow]
flag: indicates if faulty pixels must be included or not, in the points cloud.

[ok, vn, xyz0]=API.p.GtNormalV(xx, yy, zz, tolerance)
Given a set of 3D points, this function tries to fit a plane.
xx,yy,zz : arrays that define the set of 3D points which compose the surface to be approximated by a plane.
tolerance: max allowed distance of points to the estimated plane.
It returns ***ok***, to indicate success. If *ok=1* then ***vn*** will be the normal vector of the approximating plane, and **xyz0** will be a 3D point that is the centre of geometry of the fed set of points.
ok=0 means those fed points do not define a flat surface.

This function is strict. The existence of at least one outlier will make the function to report ***ok=0***.
This API function can be used ONLY for validations of results.

[ok,RollPitch] = API.e.GuessRollPitchFromVector(vn,a,b);
Given a vector, ***vn***, that is assumed to be the normal vector of a plane (that is known to be horizontal in global coordinate frame (GCF), the function returns the ROLL and PITCH of the Local CF, in which the floor plane has normal vector ***vn***.
It returns ***ok*** to indicate success. If ok=1, then the variable ***RollPitch*** will be a 2x1 vector that contains [EstimatedRoll ; EstimatedPitch] (the estimated roll and pitch).
This API function can be used ONLY for validations of results.

Context=API.i.IniAttitudePredictor(DurationOfCalibration)

Create an instance of attitude predictor
An Attitude predictor is used to integrate gyroscopes' measurement for estimating 3D attitude (it applies the model we discussed in Lecture 2, for estimating 3D attitude by transforming and integrating local angular rates usually generated by gyroscopes.)
The actual attitude prediction is performed by ***API.i.runAttitudePredictor***
You can have many instances, to integrate multiple 3D gyroscopes, or simply different versions of attitude estimations. Each instance will need to be created using ***IniAttitudePredictor***, which does return a context, which is used to call ***runAttitudePredictor*** for each of them.
The attitude predictors are used to validate your attitude estimations that are based on integrating gyroscopes measurements.
The argument ***DurationOfCalibration*** is expressed in seconds, it specifies how long is the initial calibration time (in Project1, it is usually 5 seconds)

[Attitude2, Context , ok]=
API.i.runAttitudePredictor (Attitude1,gxyz,dt, Contex);

Each time it is called, it predicts one step of attitude.
Attitude1 : current attitude. ***gxyz***: IMU angular rates (from gyroscopes), ***dt*** : time step. ***ContextofThisAttitudePredictor***: Context (i.e., to indicate which instance is to be used in this call)
It returns:
Attitude2 : new 3D attitude value
ContextofThisAttitudePredictor : same context (updated)
ok : flag, reporting success (call may fail of you feed incorrect input arguments)

The module perform bias calibration (for the gyroscopes), if it is told, when initialized (input argument ***DurationOfCalibration***, which specifies the duration of the initial calibration interval. It is expressed in seconds. We, in Project1, use 5 seconds.
During calibration, no update of the attitude is performed.

This API function can be used ONLY for validations of results.

----

h=IniScopes3Channels(figureNumber,NumberOfSamples ,min ,max, title ,labels for channels);
It creates an instance of Oscilloscope.
e.g., *hhScopesGyros=IniScopes3Channels(30,400,-50,50,'3D gyros' ,{'Wx','Wy','Wz'}); cxG=0;*
It creates an oscilloscope in figure 30, showing a horizon of 400 samples, showing vertical axis from min=-50 to max=+50. Inspect how it appears in the example programs, to appreciate the effect of those parameters
cxG is a counter that will be used each time a sample is pushed into that oscilloscope.
It returns an array of 3 graphical handles. Those are to be used when data samples are pushed to the oscilloscope.
Each Oscilloscope must use an individual counter, and individual array handles.

---

cx=PushScopes3Channels(handles , cx, sample , Horizon);
pushes a 3D sample, into an oscilloscope. The function will increment the counter cx, internally.
The input argument ***handles*** is the array of handles that was obtained in the creation of the oscilloscope. ***Horizon***: should be the same used when the oscilloscope was created. ***sample*** is that data we want to push (a 3x1 vector, for 3-channels oscilloscopes)

API.d.PretendIMUBiases(ExtraBias)
This function specifies additional fictitious biases to the gyroscopes and accelerometers measurements. It is useful for degrading measurements, to pretend that we are using a lower grade IMU. The argument is a 6x1 vector, in which element 1 to 3 are the biases for the 3D accelerometers (expressed in gravities), and the elements 4 to 6 are the extra biases for the gyroscopes' measurements (expressed in radians/second).
The IMU measurements will have the accelerometers and gyroscopes' components fictitiously affected by these defined biases.

Question: ask the lecturer, j.guivant@unsw.edu.au
