MTRN4010.2025 / **Project 1**

In this project we will process sensors' measurement for estimating the 3D attitude of a platform and altitude. We will do it in a deterministic fashion. We will need to apply concepts of:

- State space equations (nonlinear cases)
- Discrete time approximation of nonlinear state space equations.
- Linear regression (from Math).
- 3D coordinate transformations.
- 3D attitude.
- Programming (MATLAB plain Programming Language).
- Optimization.

The sensors we use are:

- 3D gyroscopes (from 3D IMU)
- Depth images from RGB-D camera.

The data we use will be played back, in a way that is equivalent to real-time operation.

We provide an API (Application Program Interface) for controlling the playback sessions, and for certain basic data processing and data visualization functionalities.

In addition, some API functions do implement certain matters that you are required to be implemented by the students (as parts of the project). Those API functions are intended to be used by the students to validate results, comparing theirs with those of those API functions. For that reason, the API is not a clear text M file but a P file (compiled binary version)

The project is composed by nine (9) parts.

Project 1 has a relevance of 23 marks (out of the 100 of the course)

The following is the list of components to be produced by the students.

0) Play with the example program, to get used to the playback framework, and with the data (3D IMU, RGB camera, Depth camera). The example program already contains parts to visualize data, in "real-time", e.g. via oscilloscopes which do show gyroscopes and accelerometers measurements. You are required to add an additional visualization to show the measurements of the 3D magnetometers **(0.5 marks)**.

1) Implement the "3D attitude predictor" based on gyroscopes' measurements. For that you will exploit the model by which we integrate the gyroscopes' measurements (the same equation that is introduced and discussed in Lecture2). The model is a continuous time state equation, so that you will implement its discrete time version, and use it appropriately, for generating estimates of the attitude at the time of each event, in "real-time", during playback sessions. You will assume that the initial attitude is Roll=Pitch=Yaw=0 (the platform did have initial roll and pitch well close to those values, since it was resting, static, aligned with the floor). In addition to that, you will show the estimated 3D attitude in a separate 3-channel oscilloscope, expressing those attitude components (roll, pitch and yaw) in degrees. **(3 marks)**.

2) Add the capability by which the program does estimate the gyroscopes biases, taking advantage of the fact that the platform is always static during at least the first five (5) seconds of its trips (from time=0 to time =5 seconds). During that interval of time (i.e., the "IMU calibration period") your attitude predictor must not modify the current attitude, it will only run after the calibration process does end.

(3.5 marks). Note: In the document "*RemovingGyrosBiasOFF_LINE_2025.pdf*", you can read about a feasible approach.

3) Modify your implementation of item (1), so that the estimated biases are used for improving the gyroscopes measurements. **(1 mark)**.

4) Add some fictitious biases to simulate a lower quality IMU. For example, gyroscope biases of 0.5 degrees/second. Test the performance of item-1 and item-3 in those cases. This trick will test your implementation if (3), for a difficult case. Use API function ***API.d. PretendExtraIMUBiases*** for defining the extra biases for the three gyroscopes (no marks are assigned to this item).

5) Implement the following capability, a button for triggering the following actions: The user is asked to specify a rectangular *Region if interest* (ROI) in the current RGB image (*), then the 3D points associated to those selected pixels are obtained (*), and then are used to estimate an approximating plane, in the platform's CF, and thus its normal vector (expressed in the platform's CF) will be calculated as well. It is assumed that the user does select a ROI that does fully correspond to part of the floor (which is assumed to be flat and perfectly horizonal in the global coordinate frame (GCF), so that ideally, in the GCF, its normal vector is perfectly vertical, i.e. [0;0;1]; however, in the platform's CF, that normal vector will depend on the platform's roll and pitch at that time. You will estimate the normal vector and print its value. Based on that normal vector you will evaluate the ROLL and PITCH angles, at that time. For validation of you results you will use API functions (the same ones used in example program Demo02_showData.m). Those API functions will calculate the normal vector and will estimate the roll and pitch of the platform. Both results (student's ones and API calculated ones) will need to have a discrepancy lower than 2 degrees. **(4 marks)**
   (timing: this item should be solved by the end of week 3, or during early week 4)
   (*): there is an API function for that purpose.

6) Refine item (5). Now, your module must detect if the ROI does correspond to a set (of points) that do not fully define a flat patch (for those points there would be no adequate approximating plane). In such a case, your function will report that there is no solution). The method you apply must exploit all the valid points associated to the selected ROI.
   Note: For defining "adequate approximating plane" we apply this practical rule: "All the points (associated to the ROI) are near the plane, having a distance to it lower than 20mm".
   Submit a brief report explaining the approach you have implemented for the plane fitting and for obtaining roll and pitch angles in (5) and (6) **(5 marks)**

7) Implement the following capability. Offer a new button in the user menu. Each time the button is pressed, you will obtain the 3D points corresponding to the last received depth image, express them in the platform's CF, then rotate them based on the estimated platform's attitude at that time. The resulting points will be plotted in a separate figure (e.g., figure#55). If all your calculations are correct, you will appreciate that the floor appears horizontal (or almost horizontal), even when the platform's roll and pitch angles are not zero. That fact will be easily verified in the figure, by visual inspection.) **(2 marks)**
   Note: Use the best predicted attitude estimates you have (e.g., those that are based on calibrated gyros' measurements)

8) Add an extra capability to (6): to estimate "altitude", in addition to roll and pitch. We define altitude as the distance from the origin of the platform's CF to the floor's plane in GCF (which is also the distance, in the platform's CF, from [0;0;0] to the plane expressed in that CF as well). Express the measured distance, in centimetres and print its value jointly with the estimates of roll and pitch. **(2 marks)**

9) Consider a fixed predefined ROI =**[u1:u2]** x **[v1:v2]**. At each "depth event", your program will estimate the roll and pitch by applying similar approach to that used in item 6 (but in this case the ROI is the predefined one so that no user intervention is required for it, and no pause will occur). You will define and use a variable, e.g., named *LastRollPitchFromDepth*, to store the last estimated roll and pitch angles. In cases in which the calculation were not successful, *LastRollPitchFromDepth* will be set to the flag value *[NaN ; NaN]*. Since that this processing is expected to run at each depth event, it must run in real-time. The overall time spent in processing a depth event must no exceed 12ms.
   Use a 4-channel oscilloscope to also show the altitude and the processing time, in addition to roll and pitch. **(2 marks)**
   Consider **[u1,u2] = [80;180]** and **[v1,v2] = [200,240]**

Validation of results
For predictions of 3D attitude, you will compare your results with those of API functions that have similar purposes. For estimates based of 3D data (roll, pitch, altitude, normal vectors of flat surfaces), you will compare which those of equivalent API functions.

API functions: There are several API functions offered to students to simplify implementation matters and for validations. Those are usually used in the example programs. You can also have information about them in file "*API_v04__help.pdf*".
In your program you will use "*API4010_v06.p*" or any posterior version, if that were available.

You may modify and use the example program "*Demo02_ forStudents.m*", if you want. We suggest trying that way; however, you may prefer to start your program from scratch.

---------- ---------- ----------

Marking criteria:

Item 0. 100%: The Oscilloscope is correctly functioning. Implementation is correct.

Item1:
[a] 50% if discrete time model and implementation are correct.
[b] 50% if the discrepancy, between student's solution and that of the provided API, has an absolute error lower than 2 degrees, always (at any time), shown in oscilloscope.
For corroborating that performance, you must show the discrepancies of all individual attitude angles, in an oscilloscope, expressing those values in degrees.

Item 2:
[a] 50%: Implementation does follow the logic of the process described in the document ("*RemovingGyrosBiasOFF_LINE_2025.pdf*").
[b] 50%: Estimated bias, after calibration, is close to the one printed by the API. Discrepancy should be <20% (of the values reported by the API function).
You demonstrator will test it, specifying certain fictitious additional biases.

Item3:
[a] 100%: Discrepancy, between student's solution and that of the API's equivalent module must have an absolute error lower than 2 degrees, always (at any time). To corroborate it, you must plot the discrepancies of all individual attitude angles (roll, pitch, yaw), using an oscilloscope. Express the discrepancies in degrees. Note: If this item does work adequately, you will not need to show item-1, you simply show this one.

Item 4: no marks are involved in this item.

Item5:

[a] 50%: Estimates of the normal vector seem close to those of the one estimated by API function (this subitem is automatically assumed correct if subitem [b] is correct.
[b] 50%: Estimates of roll and pitch, are close to those generated by the API function, having an absolute discrepancy lower than 2 degrees (i.e., in the range of errors from -2 to +2 degrees). Both angles must satisfy this discrepancy limit, always.

Item 6
[a] 50%. Required functionality: It must detect cases in which a ROI is not fully focused on a flat patch.
[b] 50%: report is correct.

Item 7
[a] 100% : by visual inspection, the parts of the floor, scanned by the camera, seem to be horizontal.

Item 8:
[a] 100%: if Estimates of altitude are mostly consistent with those of the equivalent API function. A discrepancy of up to 5 cm will be accepted.

Item 9 :
[a] 30: Oscilloscope must be functional to visualize the estimated variables (roll, pitch, altitude, processing times).
[b] 70% Processing times of the depth event are lower than 12ms (except at some sporadic events).

Additional specification: The processing of any of the events must never require more than 20ms (in the lab computers). This processing time includes all the operations you perform for servicing an event (actual calculations, plotting, etc; except those related to interaction with the user).
We will indicate how to measure that. If that processing time limit is frequently exceeded, a penalty 5 marks will be applied.
This requirement is well easy to the satisfied; however, we define it to avoid extreme cases of poor implementation.

Calculation of the project's final mark

The addition of the values obtained in each item results in the "Submitted and Demonstrated Project Mark". In addition, there is factor ***Q***, which is used for calculating the final mark of Project 1. ***Q*** is obtained based on your performance answering questions, during the demonstration, and/or via a quiz if needed. Factor ***Q*** is represented in scale [0:1]

The influence of ***Q*** on the overall project mark can be seen in the following formula.

Overall Project Mark = [Submitted and Demonstrated Project Mark] * (0.6+0.4*Q)

For instance, if you failed in answering all the questions, your **Q** factor would be ***Q=0***, which means you would get 60% of the achieved marks of your submitted/demonstrated programs.

If you correctly answer 3 questions of 4, the factor will be ***Q=0.75***, which will result in **(0.6+0.4*Q) =(0.6+0.4*0.75)= 0.90**

Deadline for submission, of the full project, is Friday Week 5, 23:59 + 2 Days (Sunday, t=23:59).

Submission will be via Moodle. It will open on Monday week 5. Details about how your program files must be organized (filenames, author details) are specified in a brief extra document, to be released in week 4.

Penalties for late submission.

1 calendar day late:	0.3 marks
2 calendar days late:	0.6 marks
N calendar days late:	[0.6 +(N-2)***4.1**] marks.

Demonstration of the project

In week 7, students will individually demonstrate their solutions, running the same program which they have submitted. A demonstrator will be present to verify performance (accuracy, etc.). The student will be asked few questions (usually four). The demonstration is a necessary step, for the project to be accepted.
The questions are used to determine the factor **Q**.

Additional useful information:

The datasets offered for Project1 are of short duration (1 minute in average), which means that once the gyroscopes are initially calibrated, the estimated gyroscopes' biases are valid for the full trip of the platform.

We always expect the estimates of the platform initial and final poses to be very similar, because in those trips the platform's initial and final poses were very similar (but not identical, due to inaccuracies of the platform's control).

More datasets are available in Moodle: https://moodle.telt.unsw.edu.au/mod/folder/view.php?id=7510950

Sensors:

The RGB-D camera and the IMU sensor are rigidly attached to the platform. The IMU is well aligned to the platform's chassis, in a way that we consider that the IMU and the platform do have the same coordinate frame (CF) for their measurements.

The camera's Y-axis is aligned to that of the platform's CF. However, the camera is rotated in pitch, so that its X-axis and Z-axis are rotated respect to the platform Y-axis. That rotation is a pure rotation in pitch, in the platform's CF. The value of that angular shift in pitch is close to 18.5 degrees. This must be considered for expressing, in the platform's CF, those 3D points clouds generated by the RGB-D camera. Read the example code, in which that compensation is applied. You will apply the same procedure.

Convention used in our coordinate frames, in this project: If I used a CF for my body, my local CF would be:

+X : my "ahead".
+Y : my left.
+Z : my upwards.

Convention for attitude angles:
φ_x (ROLL), is positive if the rotation is equivalent to +Y rotating towards +Z.
φ_y (PITCH), is positive if the rotation is equivalent to +Z rotating towards +X.
φ_z (YAW), is positive if the rotation is equivalent to +X rotating towards +Y.

Engineering units used in the sensors' data.

Accelerometers	% gravities, "G's". (1G = 9.8 m/s².)
Gyros	% radians/second
Magnetometers=	% micro-teslas

(In this project, we only use gyroscopes' measurements.)

Depth images from RGB-D camera, are expressed in millimetres (mm).

After reading those measurements you may scale them to the units you prefer, for your calculations.

For presentation/visualization of results, when we refer to angles, we will usually require those to be in expressed in degrees. You will do the necessary scaling, if needed, in your program.

Inertial Measurement Unit sensor (IMU). 3D IMU, it provides 9 channels, sampled at 200HZ.

- 3D accelerometers.
- 3D gyroscopes
- 3D magnetometers.

IMU is aligned to the platform, as follows:
- X-axis: +X pointing ahead the platform.
- Y-axis: +Y pointing to the left of the platform.
- Z-axis: +Z pointing up.

In the example programs, you can see that events associated to IMU measurements aways contain data that is a vector 9x1.

RGB-D camera.
- RGB images.
- Depth mages.

  We use it in low resolution, 320x240 (320 horizontal, 240 vertical lines).
  RGB/colour: RGB24 (3 channels, 8 bits each)
  Depth: 16 bits, 1mm precision (but accuracy is usually worse).

The original frame rate of the camera is 30HZ. However, it has been subsampled in time in the datasets, to 5HZ.

NEW API functions (API4010_v06) for validation of results.
New functions have been added to the API, especially for validating some results.

d=API.e.DistanceToPlane(vn, p0, points)
% This API function can be used ONLY for validations of results.
% calculates the distances to a plane of a set of 3D points.
% input arguments.
% ***vn***: plane's normal unit vector.
% ***p0***: a 3D point that belongs to the plane.
% ***points***: 3xN matrix which contains N 3D points (in its N columns)
% output : ***d*** : 1xN array, contains the distances to the plane, of each of the N points.
% it may return an empty variable, ***d=[ ]***, if input variables are not consistent with the expected characteristics.
Notes:
1) If ***points*** is a set of points that have been used to estimate the plane, all the distances will be lower than the tolerance used to find the plane.
2) If ***points*** is just the 3D point ***[0;0;0]*** then the resulting scalar is the distance between the plane and the origin of the platforms' CF (if the plane parameters were expressed in that CF).
3) Distances can be negative, depending in which side of the space, partitioned by the plane, is the point located. You simple use ***d=abs(d)*** is you have no interest on the sign.
4) The vector ***vn*** and the point ***p0*** are produced by your plane fitting, or by the equivalent API function,

[ok, vn, p0]=API.p.GtNormalV(xx, yy, zz, tolerance).

5) A plane in 3D can be fully defined by its unit normal vector and a point that belongs to the plane; consequently, those are the usual parameters obtained by plane fitting. However, there are other ways of representing planes in 3D.
6) ***p0*** and ***points*** must be expressed in the same engineering units (usually mm). The resulting distances will be expressed in the same units.

[ok, RollPitch] = API.e.GuessRollPitchFromVector(vn)
% This API function can be used ONLY for validations of results.
Given a unity vector, ***vn***, that is assumed to be the normal vector of a plane, which is known to be horizontal in global coordinate frame (GCF), the function returns the ROLL and PITCH angles 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 *[roll; pitch]* (the estimated roll and pitch). Both angles are in radians.

Drawing ROI in RGB figure
To visualize the predefined ROI, in the figure in which we show the RGB images, we can run these few lines of code, **just once**, when the figure is created (or before the events' loop)

```
function DrawROI_once()
        % proposed ROI in RGB/Depth image
         ROI01 = [ [80;180], [200;240]] ;
        FigureNumberRGB=10; %or the one in which you show the RGB images.
        h=API.c.plotROI(ROI01, FigureNumberRGB , 'g');
        % h=API.c.plotROI(ROI, FigureNumber ,ColorPattern );

        set(h,'LineWidth',3); h=[];
        title('RGB and ROI');
        % plot rectangular ROI in the RGB figure. We will see it, always in the RGB figure.
 end
% After this, the ROI will always be visible in that figure, even when the RGB image is updated.
```

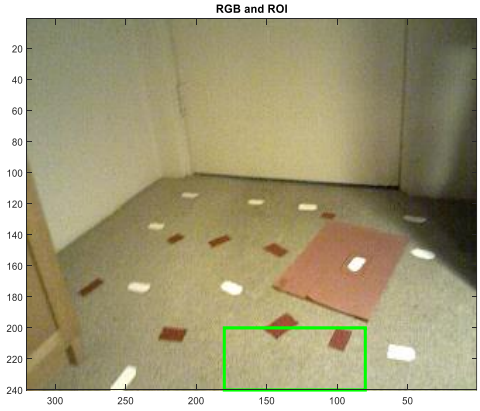

Figure 1. Predefined ROI shown in the figure for RGB camera images. By inspecting it we can see if, at that time, the ROI's associated 3D points should define, or not, a flat patch part of the floor. This may be useful to debug your "floor detector".

The rest of the API functions are consistent with those in *API4010_v04*.
API4010_v06 does include some new functions for more flexible oscilloscopes, you decide to use those or not in your program.

Typical results and performance: In addition to having API functions for validations of the students' implementations, the lecturer will show the results of his implementations, during lecture time (week 4) and via a video to be posted in Moodle and Teams.

Questions: Ask the lecturer, j.guivant@unsw.edu.au
