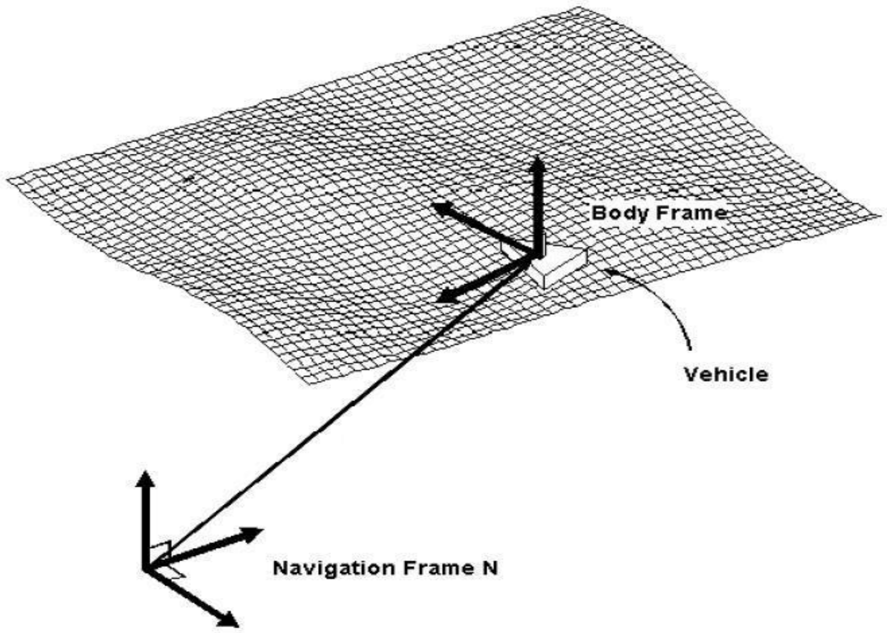
# AAS - Advanced Autonomous Systems

## INTEGRATING GYROSCOPES' MEASUREMENTS

An IMU (Inertial Measurement Unit) is composed by several sensors. 3D IMUs offer ***accelerometers*** in 3D and ***Gyroscopes*** (aka: *gyros*) in 3D; and, in some cases, other capabilities (e.g. 3D magnetometers). The accelerometers measure the acceleration of the unit. These individual sensors are internally deployed, in order to capture accelerations in 3D (it decomposes the acceleration vector in 3 orthogonal directions). The gyros measure angular velocities. They are also installed following a proper geometry, which allows the estimation of the 3D angular velocities of the unit.
These measurements (from accelerometers and gyros) are relative to the body of the IMU unit. The frame of the unit is considered to be a RIGID body; consequently, a coordinate frame is defined respect to this rigid body, and, consequently, this coordinate frame is not static, as it changes its pose with the IMU itself. This means that if we need to estimate the position and attitude of the unit, in a global (e.g. fixed to the Earth's surface) coordinate frame, we need to apply some mathematical operations (i.e. a transformation).
Even a fixed coordinate frame on the Earth's surface is not "so fixed" (it is not what we call an "inertial coordinate frame"). Remember, our planet is rotating (~360deg/day!). Any fixed point on the planet's surface is accelerated (i.e. it follows the rotating earth's surface). We will discuss the effects of this issue, later in this lecture.
For the moment just forget that our floor is being accelerated and rotated. We assume it is fully fixed, i.e. assume it is an "inertial coordinate frame" (i.e. it is neither accelerating nor rotating).

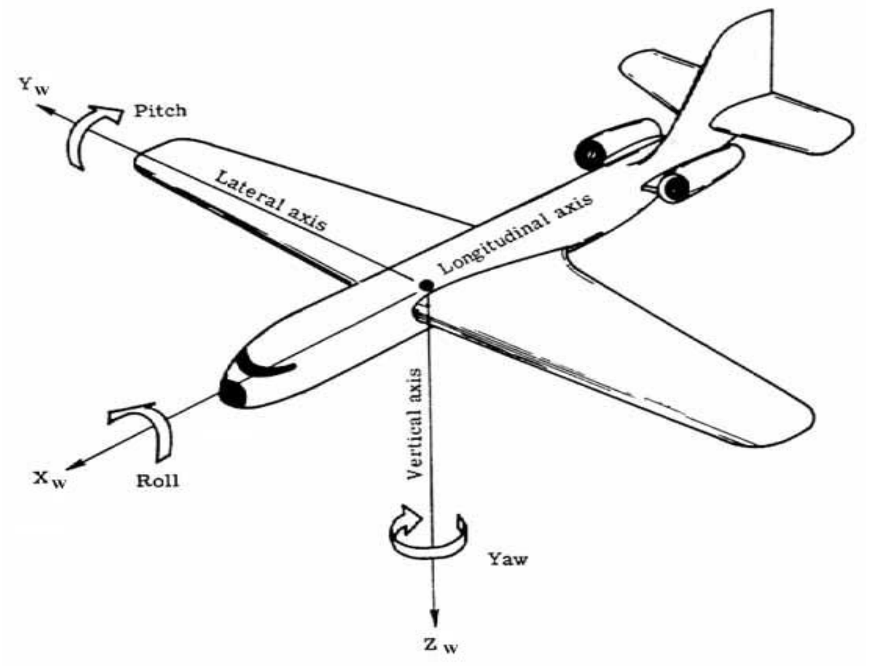


We use a navigation coordinate frame to describe the position and orientation of the robot/vehicle's body. The robot has its local coordinate frame defined on its body. Many measurements from sensors installed on the robot are expressed in this local coordinate frame. The body's frame moves (translates and rotates) with the vehicle itself. This is why we need to express the local measurement using a common coordinate frame, which we name *navigation frame*.

In some applications, the IMU's measurements can be processed (integrated) for predicting the pose (position and attitude) of the sensor (i.e. of the platform where the sensor is rigidly installed). Other applications use the gyroscopes, for estimating the attitude.
The pose's estimates are, in practical terms, usually valid for a limited period of time. The reason of it is that predicting the position implies a double integration of the accelerations. Small errors, such as a small bias, would make the resulting estimated position to diverge from the real position, in a short period of time. Integration of accelerations for position estimation is usually used in combination with other sources of information.
On the other side, the attitude can be predicted by single integration of the angular velocities (provided by the gyros). Although the angular velocities are also polluted by noise and bias, the effect is less relevant due to that the integration process involves only one time-integration.

## Attitude Representation

There are different conventions to define orientation in 3D. The following convention for *Roll, Pitch* and *Yaw* will be used in our projects.



(Image from NASA, "mtp.jpl.nasa.gov/notes/pointing/Aircraft_Attitude2.png")

The mathematical representation for this sequence is composed by three pure rotations:

a) Rotation about the *z*-axis by the yaw angle.
b) Rotation about the transformed (once rotated in step (a)) *y*-axis by the pitch angle.
c) Rotation about the transformed (twice-rotated, due to (a),(b) ) *x*-axis by the roll angle.

The associated rotation matrix (to transform points from the body coordinate frame to the navigation frame) is clearly understood as a sequence of transformations, each one being a pure rotation.

$$R\left(\varphi_x,\varphi_y,\varphi_z\right)=R\left(0,0,\varphi_z\right)\cdot R\left(0,\varphi_y,0\right)\cdot R\left(\varphi_x,0,0\right)$$

Where the pure rotations are expressed by the following rotation matrixes:

$$R(\varphi_x,0,0)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos(\varphi_x) & -\sin(\varphi_x)\\ 0 & \sin(\varphi_x) & \cos(\varphi_x)\end{bmatrix},\quad R(0,\varphi_y,0)=\begin{bmatrix}\cos(\varphi_y) & 0 & \sin(\varphi_y)\\ 0 & 1 & 0\\ -\sin(\varphi_y) & 0 & \cos(\varphi_y)\end{bmatrix}$$

$$R(0,0,\varphi_z)=\begin{bmatrix}\cos(\varphi_z) & -\sin(\varphi_z) & 0\\ \sin(\varphi_z) & \cos(\varphi_z) & 0\\ 0 & 0 & 1\end{bmatrix}$$

(Note: each of them is a 2D rotation matrix, operating in some of the 2D subspaces of the 3D space, i.e. XY,YZ, ZX)

## Attitude Estimation based on Gyros Integration

The angular velocities provided by the IMU's gyroscopes can be integrated to predict the attitude of the unit, in the navigation coordinate frame (btw: integrating them in the local coordinate frame does not make practical sense).
The sensor measures those physical variables in its local coordinate frame. Consequently, as the body of the sensor moves and rotates, the sensor's coordinate frame does vary as well. This fact implies that some extra processing is needed to express those angular velocities (measured in the sensor's coordinate frame) in some fixed and external coordinate frame. This external coordinate frame (navigation frame) makes sense for the users, because it is static, i.e. it is used as a common reference at any time and for multiple users that may need to interact.
For achieving that, the local angular velocities, provided by the gyros, are transformed to angular velocities in the common coordinate frame. The transformation is non-linear and its expression is the following one:

$$\frac{d\varphi_x}{dt}=\omega_x+\left(\omega_y\cdot\sin(\varphi_x)+\omega_z\cdot\cos(\varphi_x)\right)\cdot\tan(\varphi_y)$$

$$\frac{d\varphi_y}{dt}=\left(\omega_y\cdot\cos(\varphi_x)-\omega_z\cdot\sin(\varphi_x)\right)\qquad\text{(E1)}$$

$$\frac{d\varphi_z}{dt}=\left(\omega_y\cdot\sin(\varphi_x)+\omega_z\cdot\cos(\varphi_x)\right)/\cos(\varphi_y)$$

Expressions in E1 are non-linear but easy to calculate. This transformation assumes that the attitude in the navigation coordinate frame is according to the assumed attitude convention, which was discussed in the previous subsection.
Equation E1 is a differential equation for a 3D system. The expressions involve the three angles, which are needed to define the orientation of a rigid body, i.e. the variables $(\varphi_x,\varphi_y,\varphi_z)$. These variables describe the attitude of the body in the navigation frame. They are called Roll, Pitch and Yaw respectively, as it was discussed before. In **E1** the angular velocities, which are measured on the body's coordinate frame, are expressed as $(\omega_x,\omega_y,\omega_z)$ (i.e. these are the rates of the locally expressed Roll, Pitch and Yaw).

The roll and yaw derivatives, $(\dot{\varphi}_x,\dot{\varphi}_z)$, have singularities when the pitch of the platform is equal to π/2 (or -π/2). It is for this reason that these Euler equations are not generally implemented, however this is not a problem for ground vehicle applications (our case) as the real pitch (and the estimated pitch), should never reach the singularities. However, for the cases where "vertical" pitches are feasible, a different representation for the attitude must be used.

We can do a numerical integration of equation *E1* (such as ODE45 or similar numerical solvers in Matlab or Simulink) or directly approximate this equation by a discrete time version of it. Provided that the sample time of our discrete time system is small enough, the approximated discrete version of the continuous model is:

$$\varphi_x(t+\Delta t)=\varphi_x(t)+\Big(\omega_x(t)+\big(\omega_y(t)\cdot\sin(\varphi_x(t))+\omega_z(t)\cdot\cos(\varphi_x(t))\big)\cdot\tan(\varphi_y(t))\Big)\cdot\Delta t$$

$$\varphi_y(t+\Delta t)=\varphi_y(t)+\Big(\big(\omega_y(t)\cdot\cos(\varphi_x(t))-\omega_z(t)\cdot\sin(\varphi_x(t))\big)\Big)\cdot\Delta t \qquad \text{(E2)}$$

$$\varphi_z(t+\Delta t)=\varphi_z(t)+\big(\omega_y(t)\cdot\sin(\varphi_x(t))+\omega_z(t)\cdot\cos(\varphi_x(t))\big)\big/\cos(\varphi_y(t))\cdot\Delta t$$

The property of being "small enough", depends on the application case, i.e. the dynamics of the physical system (.e.g. our platform) , and the bandwidth of the sensor.
The process in E2 is a non-linear discrete system of the form *X[k+1]=F(X[k],u[k])* where the vector *X[k]* is the integrated variable at step "*k*" (which corresponds to certain time $t_k=t_0+k*T$), *X[k-1]* is the same variable but at the time step (step *k-1*), and *u(k-1)* is the vector of the inputs of the system, at that time.
In our model, *X* is a 3D vector which is composed by three components, which are the Euler angles of the attitude of the body. The inputs for this model are the angular velocities of the sensor (those expressed in its coordinate frame), which, for the IMU case, are measured by the gyros.

$$\varphi_x[k+1]=\varphi_x[k]+\Big(\omega_x[k]+\big(\omega_y[k]\cdot\sin(\varphi_x[k])+\omega_z[k]\cdot\cos(\varphi_x[k])\big)\cdot\tan(\varphi_y[k])\Big)\cdot T$$

$$\varphi_y[k+1]=\varphi_y[k]+\Big(\big(\omega_y[k]\cdot\cos(\varphi_x[k])-\omega_z[k]\cdot\sin(\varphi_x[k])\big)\Big)\cdot T \qquad \text{(E3)}$$

$$\varphi_z[k+1]=\varphi_z[k]+\big(\omega_y[k]\cdot\sin(\varphi_x[k])+\omega_z[k]\cdot\cos(\varphi_x[k])\big)\big/\cos(\varphi_y[k])\cdot T$$

This is a particular case of the Euler's approximation, expressed as follows:

$$\frac{dX}{dt}=F\big(X(t),u(t)\big)\quad\Rightarrow\quad X(t+\tau)=X(t)+\tau\cdot F(X,u)\big|_{X=X(t),u=u(t)},\quad \tau\to 0$$

Where the vector X is, in our particular case, the 3D vector $X=\begin{bmatrix}\varphi_x & \varphi_y & \varphi_z\end{bmatrix}^T$ and the inputs, *u*, are the gyros' measurements, $(\omega_x,\omega_y,\omega_z)$.
You can see that it is an "open loop" estimation of the attitude. If there is any error in the measurements, we have no way to correct it. For that reason we should not use this estimation process to obtain "long term" estimates of the attitude. We need to use some extra information to compensate those errors, when this approach is used in real applications.

## Example in Matlab.

The following piece of code implements the numerical integration (Euler Method) of the attitude equation.

```
Function NewAttitude  = IntegrateOneStepOfAttitude( gyros, dt, CurrentAttitude )

        % for a small delta time , dt
        % CurrentAttitude is the current (initial) attitude, in radians
        % gyros:vector with the gyros measurements, scaled in rad/sec

        ang = CurrentAttitude ;       % current global Roll, Pitch, Yaw  (at time t)
        wx = gyros(1);                %local roll rate
        wy = gyros(2);                %local pitch rate
        wz = gyros(3);                %local yaw rate

        % -------------------------------
        cosang1=cos(ang(1)) ;
        cosang2=cos(ang(2)) ;
        sinang1=sin(ang(1)) ;

        roll    = ang(1) + dt * (wx + (wy*sinang1 + wz*cosang1)*tan(ang(2))) ; %(*)
        pitch   = ang(2) + dt * (wy*cosang1 - wz*sinang1)                    ;
        yaw     = ang(3) + dt * ((wy*sinang1 + wz*cosang1)/cosang2)          ; %(*)
        % -------------------------------

        NewAttitude= [roll,pitch,yaw];       % new global Roll, Pitch, Yaw (at time t+dt)
return ;
```

*% (*): you could see that if ang(2) is close to pi/2 or –pi/2, the term tan( ) and 1/cos( ) would have numerical problems.*