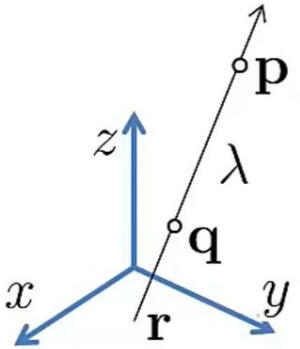
3D Geometry

Points in 3D

- 3D point $\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3$
- Augmented vector $\bar{\mathbf{x}} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \in \mathbb{R}^4$
- Homogeneous coordinates $\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{pmatrix} \in \mathbb{P}^3$

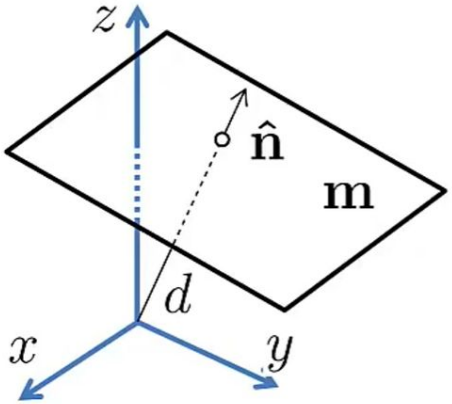
Geometric Primitives in 3D

- 3D line $\mathbf{r} = (1 - \lambda)\mathbf{p} + \lambda\mathbf{q}$ through points $\mathbf{p}, \mathbf{q}$
- Infinite line: $\lambda \in \mathbb{R}$
- Line segment joining $\mathbf{p}, \mathbf{q}$: $0 \leq \lambda \leq 1$



Geometric Primitives in 3D

- 3D plane $\tilde{\mathbf{m}} = (a, b, c, d)^\top$
- 3D plane equation $\bar{\mathbf{x}} \cdot \tilde{\mathbf{m}} = ax + by + cz + d = 0$
- Normalized plane $\mathbf{m} = (\hat{n}_x, \hat{n}_y, \hat{n}_z, d)^\top = (\hat{\mathbf{n}}, d)$
  with unit normal vector $\|\hat{\mathbf{n}}\| = 1$ and distance d



3D Transformations

Identity matrix

3D translation vector

- Translation

$$\tilde{\mathbf{x}}' = \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix}}_{4\times 4} \tilde{\mathbf{x}}$$

- Euclidean transform (translation + rotation), (also called the Special Euclidean group SE(3))

$$\tilde{\mathbf{x}}' = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \tilde{\mathbf{x}}$$

- Scaled rotation, affine transform, projective transform…

Where we have an additional scaling factor; to grow or shrink objects or trajectories in 3D.

3D Euclidean Transformations

We need it to move around the robot in 3D and to compute the motions between poses.

- Translation $\mathbf{t} \in \mathbb{R}^3$ has 3 degrees of freedom
- Rotation $\mathbf{R} \in \mathbb{R}^{3\times3}$ has 3 degrees of freedom

Because we can turn the quad rotor around all 3 principle axes.

Because we can move around the object or quadcopter along all 3 axes.

$$\mathbf{X} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3D Rotations

- A rotation matrix is a 3x3 orthogonal matrix

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

Because it preserves the scale and angles.

- Also called the special orientation group SO(3)
- Column vectors correspond to coordinate axes **after the transformation**

3D Rotations

Concatenation = Multiplication
As we did in the odometry from sensors.

- Advantage:
  Can be easily concatenated and inverted (how?)

Sometimes we need to estimate a rotation matrix because we find out how the robot moved between two timestamps. The rotation matrix (or orthogonal matrix) can be inverted by taking the transpose of the matrix.

- Disadvantage:
  Over-parameterized (9 parameters instead of 3)

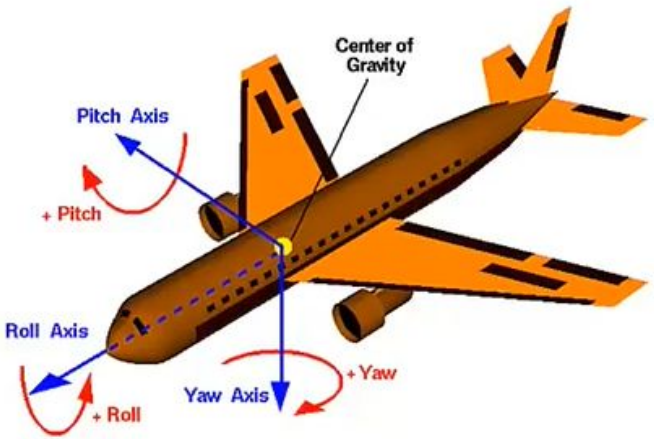
So, if we have an optimization problem, we have to make sure that all these parameters are important and needed.

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

Euler Angles

One of the best approaches for optimization

- Product of 3 consecutive rotations (e.g., around X-Y-Z axes)
- Roll-pitch-yaw convention is very common in aerial navigation (DIN 9300)



http://en.wikipedia.org/wiki/File:Rollpitchyawplain.png

This means:

1. Specify the first roll angle along the main axis of the quadcopter.
2. Then, the pitch angle, that specifies the rotation around the wings of the quadcopter.
3. Then, the yaw angle, that specifies the rotation around the heading of the quadcopter.

Roll-Pitch-Yaw Convention

To convert from euler angles to the rotation matrices and vice versa.

- Roll ϕ, Pitch θ, Yaw ψ
- Conversion to 3x3 rotation matrix:

$$\mathbf{R} = \mathbf{R}_Z(\psi)\mathbf{R}_Y(\theta)\mathbf{R}_X(\phi)$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{pmatrix}$$

Roll-Pitch-Yaw Convention

- Roll ϕ, Pitch θ, Yaw ψ
- Conversion from 3x3 rotation matrix:

$$\phi = \text{Atan2}\left(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}\right)$$

$$\psi = -\text{Atan2}\left(\frac{r_{21}}{\cos(\phi)}, \frac{r_{11}}{\cos(\phi)}\right)$$

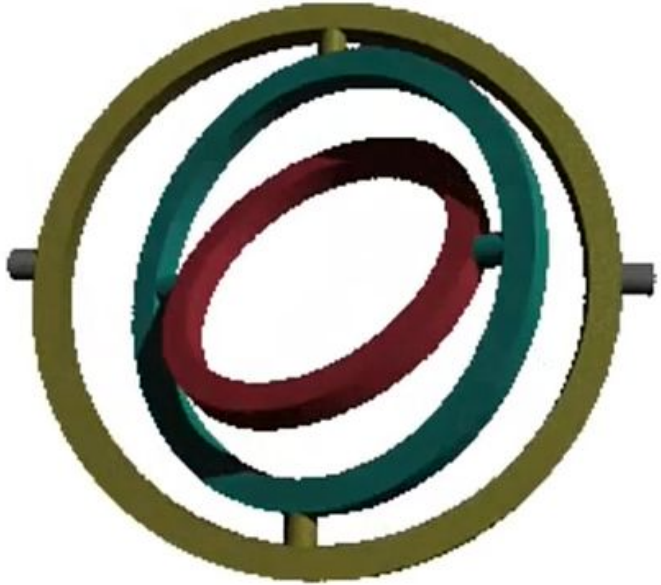
$$\theta = \text{Atan2}\left(\frac{r_{32}}{\cos(\phi)}, \frac{r_{33}}{\cos(\phi)}\right)$$

Euler Angles

- Advantage:
  - Minimal representation (3 parameters)
  - Easy interpretation → Especially if the angles are close to zero
- Disadvantages:
  - Many "alternative" Euler representations exist (XYZ, ZXZ, ZYX, …) It causes a confusion if somebody sends the euler angles, because there are many conversion ways.
  - Difficult to concatenate → Because you always have to convert them to a rotation matrix and then do the concatenation, then extract again the resulting euler angles.
  - Singularities (gimbal lock) → It occurs when the axes of the quadcopter aligns on each other, there we can't move the 3 axes at the sametime at this instance. This, we lose one degree of freedom. This is a problem with no solution in euler angles

Gimbal Lock

- When the axes align, one degree-of-freedom (DOF) is lost:

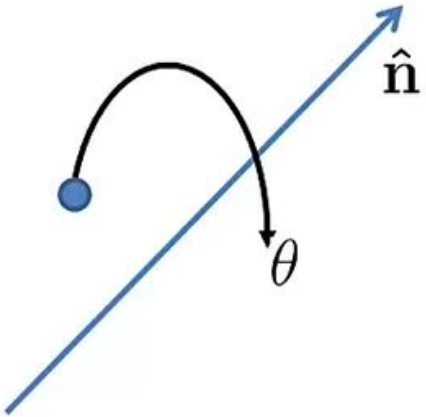


http://commons.wikimedia.org/wiki/File:Rotating_gimbal-xyz.gif

Axis/Angle

A normal direction vector in 3D and a rotation angle theta which says how far you rotate around the axis.

- Represent rotation by
  - rotation axis $\hat{\mathbf{n}}$ and
  - rotation angle θ
- 4 parameters
- 3 parameters
  - length is rotation angle
  - also called the angular velocity
  - minimal but not unique (why?)



Every 2PI, we repeat the same rotation

Conversion

- Rodriguez' formula → **To convert from axis/angle to rotation matrices.**

$$\mathbf{R}(\hat{\mathbf{n}}, \theta) = \mathbf{I} + \sin\theta[\hat{\mathbf{n}}]_\times + (1 - \cos\theta)[\hat{\mathbf{n}}]_\times^2$$

- Inverse

$$\theta = \cos^{-1}\left(\frac{\text{trace}(\mathbf{R}) - 1}{2}\right), \hat{\mathbf{n}} = \frac{1}{2\sin\theta}\begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix}$$

see: An Invitation to 3D Vision (Ma, Soatto, Kosecka, Sastry), Chapter 2

Axis/Angle

- Also called twist coordinates
- Advantages:
  - Minimal representation
  - Simple derivations
- Disadvantage:
  - Difficult to concatenate
  - Slow conversion

The same problem as euler angles as to perform the concatenation, we should transform the axis/angle coordinates into rotation matrix and then re-extract the axis/angle coordinates again

Quaternions

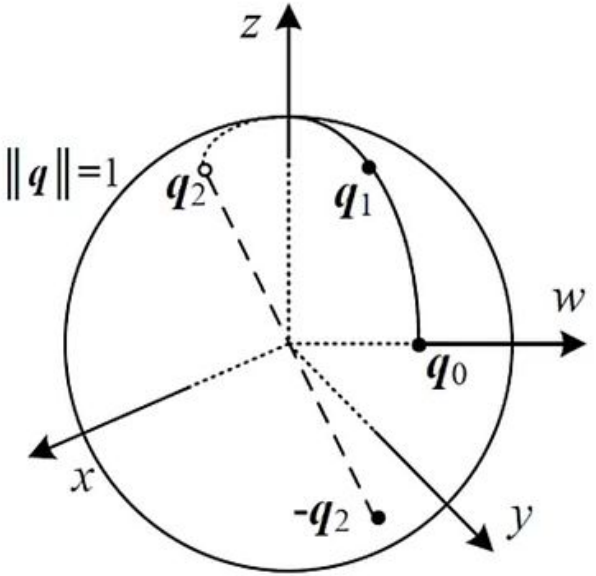
4D Vector representing a rotation

- Quaternion $\mathbf{q} = (q_w, q_x, q_y, q_z)^\top \in \mathbb{R}^4$
- Real and vector part

$$\mathbf{q} = (r, \mathbf{v}),\ r \in \mathbb{R},\ \mathbf{v} \in \mathbb{R}^3$$

- Unit quaternions have $\boxed{\|\mathbf{q}\| = 1}$
- Opposite sign quaternions represent the same rotation
- Otherwise unique

Length of quaternion:
It is just like points on 4D sphere of radius 1.



Richard Szeliski, Computer Vision: Algorithms and Applications
http://szeliski.org/Book/

Quaternions

- Advantage:
  multiplication, inversion and rotations are very efficient
- Concatenation

$$(r_1, \mathbf{v}_1)(r_2, \mathbf{v}_2) = (r_1 r_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, r_1 \mathbf{v}_2 + r_2 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)$$

- Inverse (=flip signs of real or imaginary part)

$$(r, \mathbf{v})^{-1} = (r, \mathbf{v})^{\star} \equiv (-r, \mathbf{v}) \equiv (r, -\mathbf{v})$$

Quaternions

- Rotate 3D vector $\mathbf{p} \in \mathbb{R}^3$ using a quaternion:

Vector

Quaternion

$$(r, \mathbf{v})(0, \mathbf{p})(r, \mathbf{v})^\star$$

Multiplication of the quaternion with the vector and the conjugate of the quaternion.(fast to compute).

- Relation to axis/angle representation

$$\mathbf{q} = (r, \mathbf{v}) = (\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\hat{\mathbf{n}})$$

Regarding 3D rotations:
For concatenating → Rotation
Matrices and quaternions → Suited

Optimize a 3D representation:
axis/angle representation is suited

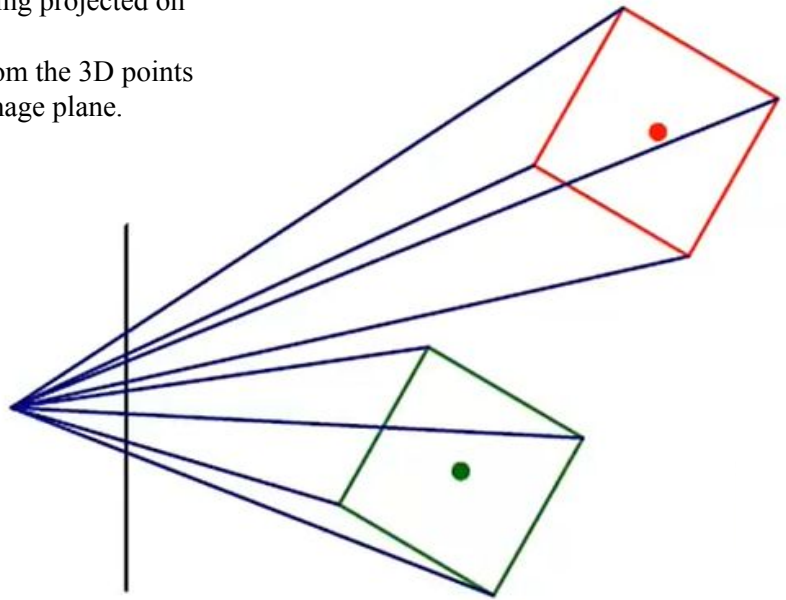
3D Visualization tools:
Linux → ROS → RVIZ
C++ → Library → QGLViewer

3D Orientations

- **Note:** In general, it is very hard to “read” 3D orientations/rotations, no matter in what representation
- **Observation:** They are usually easy to visualize and can then be intuitively interpreted
- **Advice:** Use 3D visualization tools for debugging (RVIZ, libqglviewer, …)

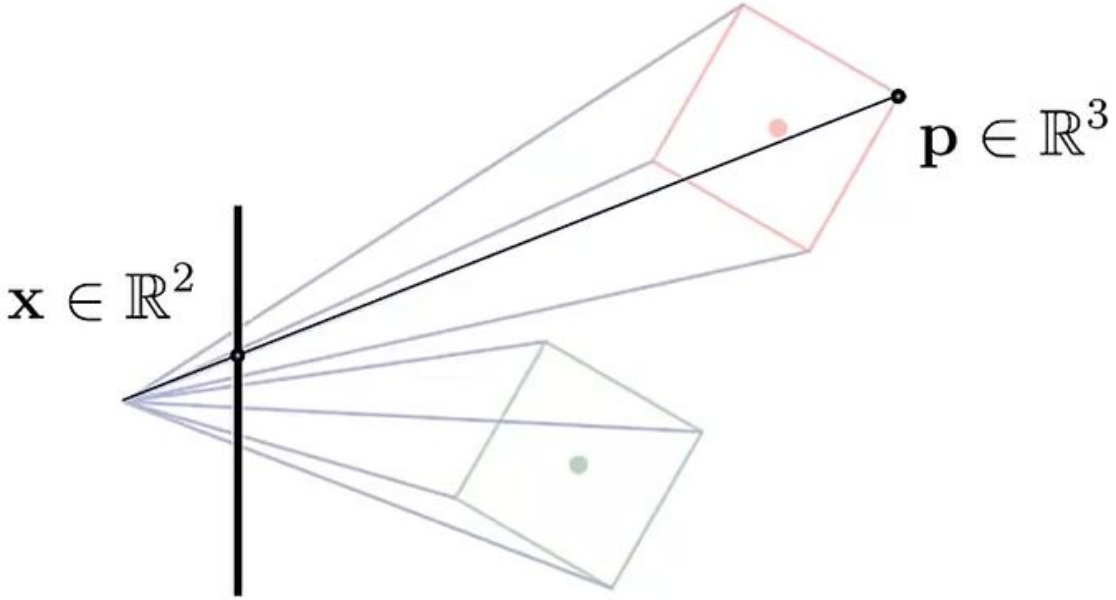
3D to 2D Perspective Projections

For example, if we have objects being located in the real world in 3D, and we have a camera actually observing these objects, and then, these objects being projected on 2D image plane of the camera.
So, we need to find away to compute from the 3D points in the world to 2D coordinates on the image plane.



Richard Szeliski, Computer Vision: Algorithms and Applications
http://szeliski.org/Book/

3D to 2D Perspective Projections



Richard Szeliski, Computer Vision: Algorithms and Applications
http://szeliski.org/Book/

3D to 2D Perspective Projections

The most used model for most cameras that have normal lenses

- Pin-hole camera model

We just remove the last component of our homogeneous vector to obtain a 2D homogeneous vector of the point.

$$\tilde{\mathbf{x}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tilde{\mathbf{p}}$$

- Note: $\tilde{\mathbf{x}}$ is homogeneous, needs to be normalized

$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix} \quad \Rightarrow \quad \mathbf{x} = \begin{pmatrix} \tilde{x}/\tilde{z} \\ \tilde{y}/\tilde{z} \end{pmatrix}$$

The real coordinate on the image plane

After conversion from 3D to 2D, the output is in meters.
But in cameras, we need to convert the position in meters to a position in pixels.
For that, we apply some scaling and offset.
The matrix that does scaling and offset is called Intrinsic matrix (k).
This matrix consists of 5 parameters:

Focal length (Fx and Fy)

Specifies the opening angle of the camera → The larger this focal length, the smaller the opening angle of the camera and vise versa

Camera center

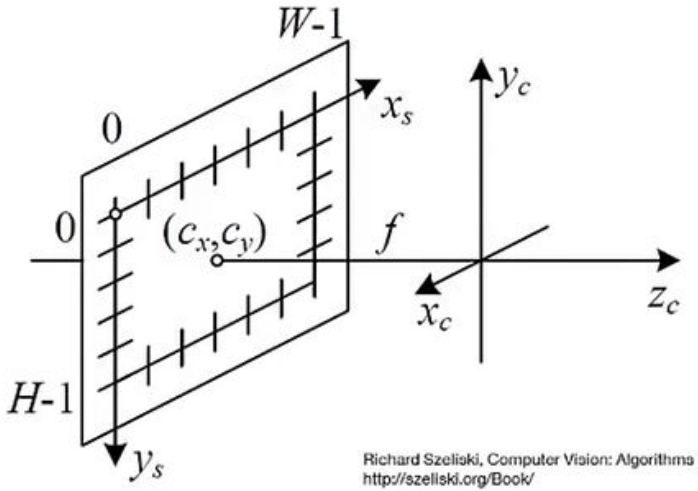
Is the middle of the image, but it doesn't necessarily have to be there, it depends on how exactly the lense if mounted and where exactly the sensor is mounted.

Skew factor (s)

Determine whether or not the pixels on the image array are actually orthogonal or not.
[In most modern cameras, no skew. So, (s) can be assumed to be 0].

Camera Intrinsics

- So far, 2D point is given in meters on image plane
- But: we want 2D point be measured in pixels (as the sensor does)



Richard Szeliski, Computer Vision: Algorithms and Applications
http://szeliski.org/Book/

Camera Intrinsics

- Need to apply some scaling/offset

$$\tilde{\mathbf{x}} = \underbrace{\begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}}_{\text{intrinsics } K} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\text{projection}} \tilde{\mathbf{p}}$$

- Focal length f_x, f_y
- Camera center c_x, c_y
- Skew s

Camera Extrinsics

- Assume $\tilde{\mathbf{p}}_w$ is given in world coordinates
- Transform from world to camera (also called the camera extrinsics)

Given in world coordinates (3D)

Where the camera is located in 3D space. The same as conversion from global to local coordinate system of the camera.

$$\tilde{\mathbf{p}} = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \tilde{\mathbf{p}}_w$$

- Projection of 3D world points to 2D pixel coordinates:

$$\tilde{\mathbf{x}} = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R & \mathbf{t} \end{pmatrix} \tilde{\mathbf{p}}_w$$

Sensors

Sensors

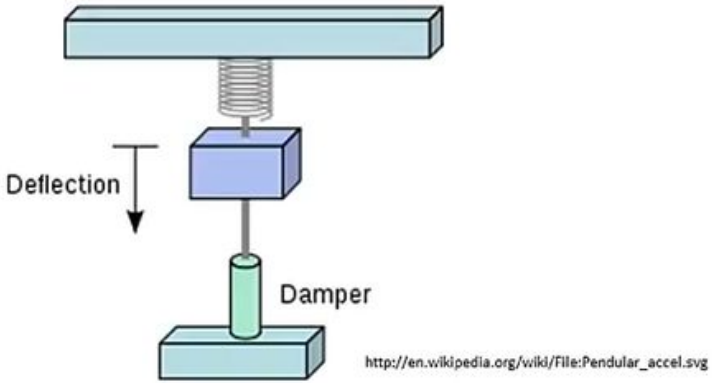
IMU is used to compute the inertial motion of the quadcopter and that's typically used for altitude stabilization.

- IMUs (inertial measurement units)
  - Accelerometers
  - Gyroscopes
- Range sensors (sonar) → To measure the distance from the floor and controls that.
- GPS → Important especially for outdoors applications
- Cameras

For estimating the horizontal speed if you have an optical flow sensor looking downwards.
OR for a forward camera to detect landmarks in the world.

Accelerometers

- Measures all external forces acting upon them (including gravity)
- Acts like a spring-damper system
- To obtain inertial acceleration (due to motion alone), gravity must be subtracted



http://en.wikipedia.org/wiki/File:Pendular_accel.svg

Gyroscopes

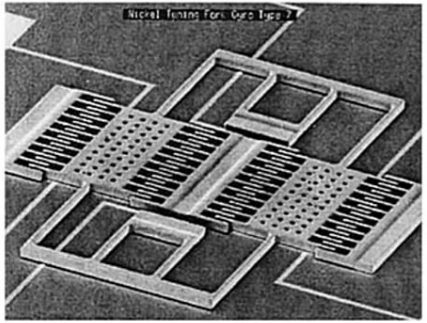
- Measures orientation (standard gyro) or angular velocity (rate gyro, needs integration for angle)
- Spinning wheel mounted in a gimbal device (can move freely in 3 dimensions)
- Wheel keeps orientation due to angular momentum

http://en.wikipedia.org/wiki/File:Gyroscope_operation.gif

http://en.wikipedia.org/wiki/File:Foucault%27s_gyroscope.jpg

MEMS Gyroscopes

- Vibrating structure gyroscope (MEMS)
  - Based on Coriolis effect
  - “Vibration keeps its direction under rotation”
  - Implementations: Tuning fork, vibrating wheels, …



Bernstein, J., An Overview of MEMS Inertial Sensing Technology, Sensors, Feb. 2003.

Inertial Measurement Units (IMUs)

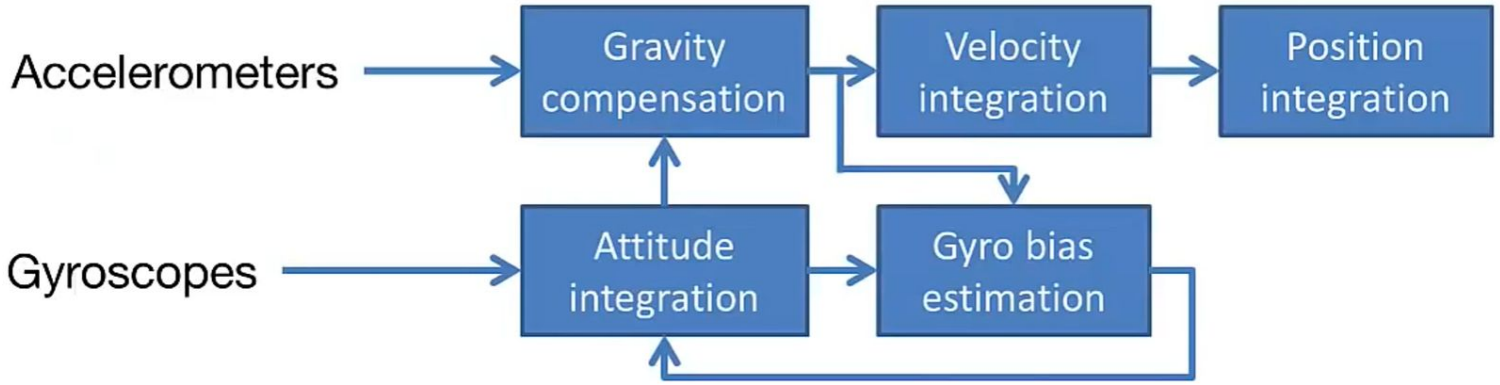
- 3-axes MEMS gyroscope
  - Provides angular velocity
  - Integrate for angular position
  - Problem: Drifts slowly over time (e.g., 1deg/hour), called the bias
- 3-axes MEMS accelerometer
  - Provides accelerations (including gravity)

One gyroscope provides only one angular velocity. So, we need to integrate 3 gyroscopes overtime to obtain the absolute angular positions.

The readings from gyroscope (especially small one) will be very noisy, the angular position will drift overtime, this drift value depends on the quality of the gyroscope.

Through IMU → You can integrate the estimated speed (from accelerometer) and estimated position (from gyroscope).
This is done in an algorithm or in an implementation called IMU Strapdown algorithm. The data is got from the accelerometers and gyroscopes.

IMU Strapdown Algorithm



Explanation for IMU Strapdown algorithm

1. With the gyroscopes, we are getting the angular rates in all the 3 directions. Then, the first computation block here is just integrating those readings to obtain the attitude of the quadrotor, just by adding them up.
   [angular velocity around x-axis + angular velocity around y-axis + angular velocity around z-axis = attitude integration]

2. When you have the attitude, you know in which direction you will except a gravitational force. So, you can use that to compensate for gravity from the acceleration readings, which means that after this gravity compensation, you have the pure acceleration that is applied to the IMU at the moment.

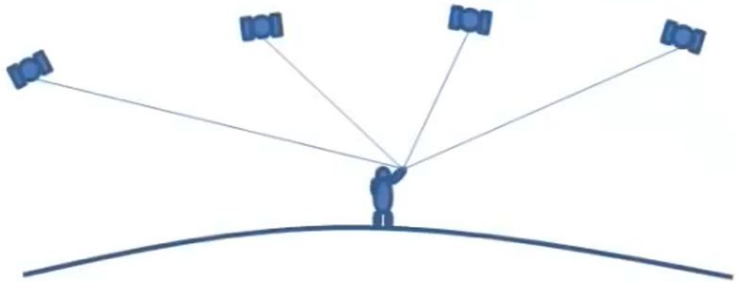
3. When you integrate the acceleration, then you obtain the velocity of the device, and when you integrate that again, you get the position.

It should be noted that when you integrate twice such noisy readings then the result will be extremely noisy and wrong. So, the position integration of MEMS IMU is not useful over larger periods of time. But, you can usually get rather good predictions for the next 100 ms and may be even up to 1 sec, but after that the readings will be extremely inaccurate.

4. This estimated acceleration after the gravity compensation then typically run through a low pass filter to see if there is any acceleration remaining in a certain direction and this is then usually because the attitude is not completely right. This can be compensated by estimating the bias of the gyroscope and then feeding it back into the attitude integration to compensate for such a bias over larger periods of time.

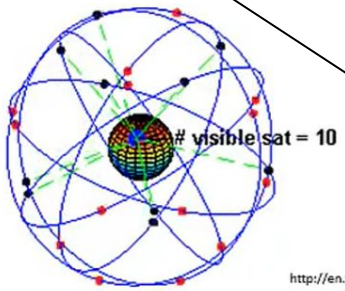
GPS

- Every satellite transmits its position and time
- Receiver measures time difference of satellite signals
- Calculate position by intersecting distances (pseudoranges)



GPS

- 24+ satellites, 12 hour orbit, 20.190 km height
- 6 orbital planes, 4+ satellites per orbit, 60deg distance
- Satellite transmits orbital location (almanach) + time
- 50bits/s, msg has 1500 bits → 12.5 minutes



http://en.wikipedia.org/wiki/File:ConstellationGPS.gif

Because all satellites share the same frequency of transmission, the transmission rate is extremely low ≃ 50 bits/sec

That's why a cold start of the receiver takes a very long time before you get the first fix.

GPS

GPS has 2 different modes of operations, the normal one is used by the most smart phones

1. Position from pseudorange
   - Requires measurements of 4 different satellites
   - Low accuracy (3-15m) but absolute

Because the receiver has to estimate both the current time and its position on earth.

2. Position from pseudorange + phase shift (RTK/dGPS)
   - Very precise (<1mm)
   - Position is relative to a reference station

Requires additional base station or reference station

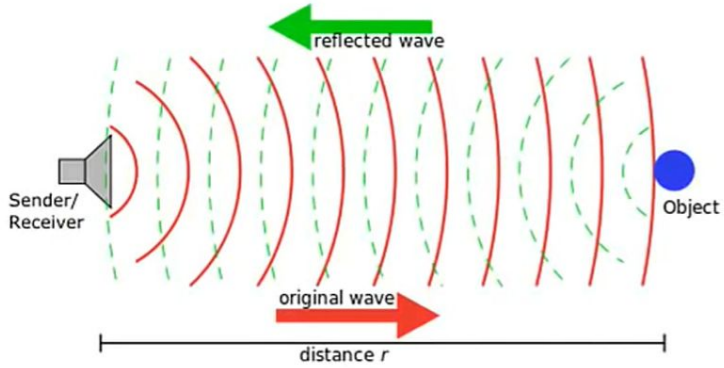
Between the receiver and the base station

This way delivers accurate results, they are not absolute values, but relative with respect to your reference station. Such systems are called real time kinematic GPS or differential-GPS systems.

Ultrasound Range Sensors

The idea is to emit a signal and measure the time the signal needs to return to the sender/receiver.

- Emit signal to determine distance along a ray
- Make use of propagation speed of ultrasound
- Traveled distance is given by speed of sound (v=340m/s)



http://en.wikipedia.org/wiki/File:Sonar_Principle_EN.svg

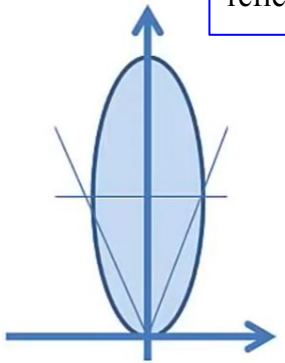
$$d = \frac{v\Delta t}{2}$$

Ultrasound Range Sensors

- Range between 12cm and 5m
- Opening angle around 20 to 40 degrees
- Problems: multi-path propagation, absorption
- Lightweight and cheap

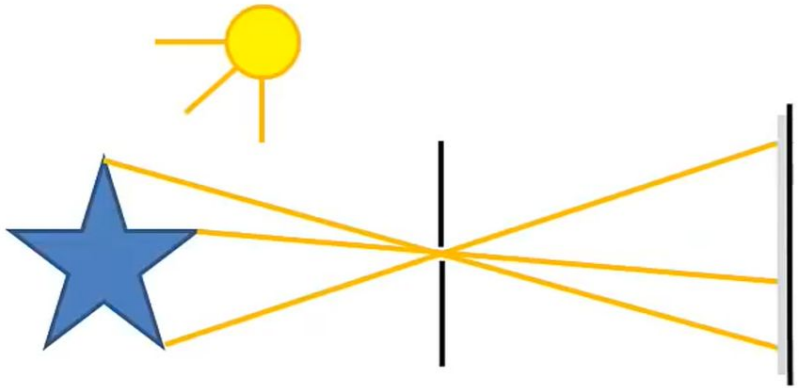
This is better for things like stones or better reflecting surfaces (not grass for example).

http://www.parallax.com/product/28015



Cameras

- Pinhole Camera
- Lit scene emits light
- Film/sensor is light sensitive



The idea of camera sensor

The idea is you have a world, and the world emits some light because it is lit by the sun or any other light source. Then, the rays from the object pass through the pin-hole and then fall on the film or light sensor which is light sensitive and that we can reach out electronically.

Most cameras don’t use pin-hole idea, but have a lens that focuses the light on the film or sensor.

Advantages:
Larger aperture, which means that more light is passed on to the sensor, and that helps during image acquisition.

Disadvantages:
They typically induce some radial distortion of the image.
It is bad for lenses that have large opening angles. It is not a problem for the webcams, but if you getting may be above a 100 degrees opening angle, these deviations become visible very strongly. And the larger it gets the worst it gets.

Lens Distortions

- Radial distortion of the image
  - Caused by imperfect lenses
  - Deviations are most noticeable for rays that pass through the edge of the lens

Lens Distortions

- Radial distortion of the image
  - Caused by imperfect lenses
  - Deviations are most noticeable for rays that pass through the edge of the lens
- Typically compensated with a low-order polynomial

$$\hat{x}_c = x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$
$$\hat{y}_c = y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$

R→ radius or the distance from the optical center of the camera or the image.
K1 & k2 → coefficients(kappa_1 and kappa_2). They refer to how strong this deviation is on your lens.

Camera Calibration

For being able to use the camera models, we need to determine all the coefficients in the models

- Pinhole model

$$\tilde{\mathbf{x}} = \begin{pmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tilde{\mathbf{p}}$$

Fx and Fy → Focal length
Cx and Cy → Optical center

- Distortion model

$$\hat{x}_c = x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$
$$\hat{y}_c = y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$

- How can we determine these parameters?

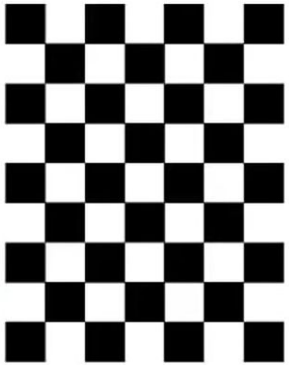
How to determine these coefficientS?

This is done by using a checkerboard to the camera and use that to establish correspondences between 3D points in the world and 2D points on the image.

And from such a set of observations, the coefficients of the camera model can be determined. Every observation between 2D point and 3D point produces 2 constraints. One for x-axis is in the image and one for y-axis in the image, and then use that to solve the intrinsic parameters, like the focal length and the optical center, but also for distortion coefficients K1 and K2.

Camera Calibration

- Collect n corresponding observations between
  - 3D points $\mathbf{p}_1, \ldots, \mathbf{p}_n$ (3D location in world coordinates)
  - 2D points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ (on the image plane)
- Typically using a calibration board



Camera Calibration

- Every observation $\mathbf{x}_i, \mathbf{p}_i$ produces two constraints
- Solve for f_x, f_y, c_x, c_y + distortion coefficients κ_1, κ_2

The image in lower left shows the calibration toolbox in ROS for monocular camera.