Beaver Works Summer Institute 2018
Unmanned Aerial Vehicle

Spencer Ng

Drone Architecture

- Two main control systems
- AeroFC responsible for just flight, very low level (PX4)
- Aero Compute Board is high level, using ROS as middleware

Robotics Operating System (ROS)

- Use `rostopic echo <topic>` to print from a topic
- MAVROS is for AeroFC, ROS is for Aero Compute Controller
- Nodes subscribe and publish to topics
  - Make decisions based on messages sent by other nodes

Localization Techniques

- Need to recognize where the drone is in the room or relative to frame of reference (origin)
- Includes velocity, pitch, roll
- Techniques include
  - Dead reckoning
    * Using gyros, accelerometers, pots to determine how much a plant has moved
    * Not good due to external inputs (e.g. wind)
  - GPS
    * Gives a very good signal
    * Precision only within meters
    * Can't use indoors
  - Radio Frequency/Ultra Wideband
    * Good for controlled environments, needs to be set up
  - Motion capture
    * Fast and reliable
    * Very popular for research due to high precision
    * Only good for dedicated rooms and expensive
    * Unrealistic
  - Visual Odometry
    * How we navigate, with a visual map

- SLAM (Simultaneous Localization And Mapping)
  - Creates a map and tries to localize at the same time
  - Huge issue in robotics today, in terms of accuracy and performance
  - IntelAero is capable, but is difficult to achieve well
  - Active area of research

Reference Frames

- Need to understand quadrotor reference frame with respect to the world
- Either inertial or non-inertial (not accelerating, may be at constant nonzero velocity)
- NED or ENU is the world reference frame
  - $+z$ is down in NED (north east down)
  - $+z$ is up in ENU (east north up)
  - $+x$ is north and $+y$ is east in NED, vice versa in ENU
- Body frame (quadrotor)
  - $+x$ toward front
  - $+z$ down
  - $+y$ toward right "wing"
  - Origin is at center of mass
- Marker frame is attached to AR marker
  - Why tags are so useful
- Transformations can be made mathematically

Convolutions

- Moving average of intensity values in a grid produces edges
  - Smoothing function to suppress noise but makes it blurrier
- Image segmentation based on threshold
  - If above, make white; otherwise, change to black
- Shift invariance
- Linear filters
- To handle boundaries, duplicating parts of the image rather than leaving an edge produces a better result after running blurring kernel
- Kernels are matrices for filters
  - Sobels are for edge detection, starting from a particular direction
  - New pixel intensity is equal to "cross product" of kernel with adjacent pixels, equal to kernel size
- Linear regression - extract lines out of an image
  - Algebraic approach involves computing averages of the points to minimize squared error
  - Machine learning: guess and check from points (RANSAC technique)
    - Two parameters: slope and intercept

      * Draw a line through two points at random, set "thresholds," determine how many "inliers" and outliers
        · Measure distance from every point to that line (either vertical/least squares or normal)
        · Distance is the cost to minimize
      * Rinse and repeat until inliers increase in number, until there's a good line
      * "Hopping around" the slope vs intercept vs squared error graph, for a set number of iterations
      * Convex optimization – not all problems have a definite curve to minimize error
        · Hard to change parameters and descend "cost hill"

- Operations
  - Adding
  - Blending
  - Erosion
  - Dilation
  - Rotation

Computer Vision

- AR tags in use
- A .bag file stores raw sensor data
- More than light and scene properties?
  - Optical illusions and human perception is important
- Edge detection
  - Use intensity rather than color differences
  - Makes us capable of recognizing objects
  - Horizon and vanishing points used for depth perception
  - Surface discontinuity – based on light, depth, or shadows
- Computers better than humans at parallel procssing
- Goal of vision is to turn binary data into something recognizable
  - Many applications
- 3D graph of intensity as a grid of pixels
- Image is still a discrete signal, rather than countinuous
- Intensity ranges from 0 to 255
- Color models are RGB, lab, or HSV
- An image can be a function from $\mathbb{R}^2$ to $\mathbb{R}^M$, where M is the number of colors
  - A position (x, y) is converted into (R, G, B)
- Images formed either perspective projection (has diagonals) or orthographic projection (only parallel lines)
  - Orthographic taken from very far away, then zooming in
- Image formation model for three dimensional world
  - Y and Z axes are meshed togeteher in 2D representation

  - X is preserved, in the same orientation

- Edge classification
  - Figure segmentation
  - Occlusion edges
    - Only the foreground
  - Contact edges – touching the ground
    - Y is 0 because the object is touching the ground
    - Z value is still unknown

- Accidental image – distorts perception of axes, loses info

- Differential geometry - using $(N-1)$-D images to visualize N-D objects

- Systems and filters
  - Goal is to extract feature (edges, blobs) from images
  - Denoising, in-painting, etc
  - Linear filter in 1D
    - A set of n values in an array g
    - Another signal (sort of like modulating in AM) is introduced to filter/transform that signal
    - Used to create an overall transformative function, $f(n) = H(g(n), s(x, y))$
  - Translation invariant filter – do same operation at every point in the image to filter it
    - If input image is filtered by m samples, output also translated by m samples
  - Convolution
    - Key operation for machine learning/embedded (local) vision
    - Sliding/multiplying a set of values across another set of values to produce a transformation
    - Same as cross correlation

- Characterizing edges – place of rapid change in image intensity
  - First derivative of intensity vs. Position graph (aka gradient profile) shows spikes/peaks
  - Noise causes derivative/gradient profile to be unreadable, as there are many edges
  - Finite differences is susceptible to noise
  - Smooth/filter by combining the input signal and a bell curve (Gaussian distribution) $f(x) = s(b(x))$
  - To detect change in intensity, just take the derivative and use convolution after smoothing
  - Power rule: $\frac{d}{dx}x^n = n \cdot x^{n-1}$

- Deep learning
  - Convolution Neural Networks (CNN) - collections of basic mathematical operations
  - Training begins with discriminator and generator, which learn from each other
  - Called "deep" because transformations keep on going further down

- Recurrent Neural Network (RNN)
  - Keep on transforming the image; feed output back into input

Color Segmentation

- RGB color space – visualized as 3D graph, values 0-255 for each channel
- HSL/HSV - polar coordinates, visualized as cylinder
  - Saturation is radius
  - Lightness/value is height
  - Hue is degree/direction
- Color thresholding – cutoff at particular value
- Color masking

States

- State vector contains 12 variables for complete and unique description of quadrotor
  - $\dot{\vec{\zeta}} = f(\vec{\zeta}, \vec{u})$
  - x, y, z for position (could be very noisy)
  - v_x, v_y, v_z for velocity (often calculated or estimated)
  - ϕ, θ, ψ for roll, pitch, and yaw
  - p, q, r for pitch, roll, and yaw "velocities"

$$\vec{\zeta} = \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \\ \phi \\ \theta \\ \psi \\ p \\ q \\ r \end{bmatrix}$$

- $\vec{u}$ contains angular velocities for four rotors $(\omega_1, \omega_2, \omega_3, \omega_4)$
  - Derive acceleration/force using $F = m \cdot \vec{a}$ and $\tau = I \cdot \dot{\omega}$

$$\vec{u} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \rightarrow \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \rightarrow \begin{bmatrix} f \\ \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix}$$

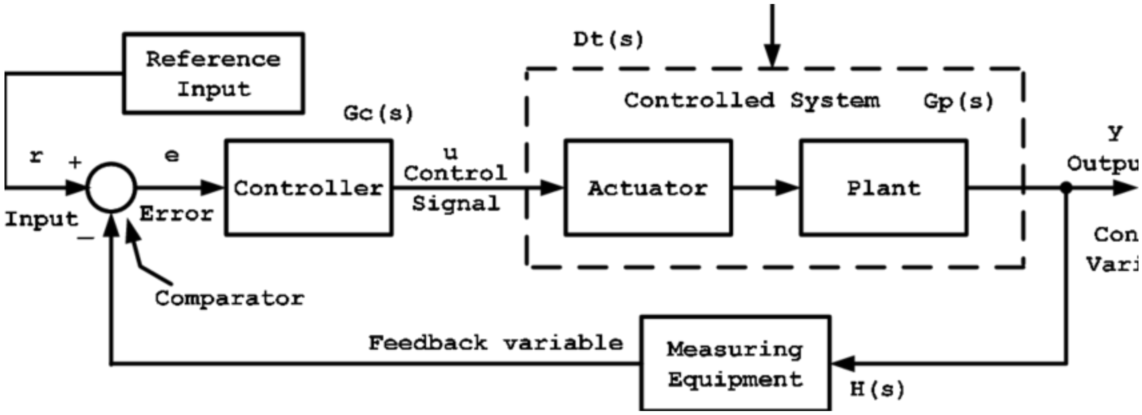
- Above is the low level control, where f is the sum of all forces on four rotors, τ_n is the torque on the center of mass on the quadrotor, in all three directions
- Rightmost vector is what's really used
- Physics is mostly handled by the flight controller, which we just send velocity commands to via MAVROS

$$\dot{\vec{x}} = \frac{d^w \vec{x}_B}{dt}$$
$$\ddot{x} = mg\vec{z}_w - u_1\vec{z}_b$$
$$\dot{R}_{BW} = R_{BW}\vec{\Omega}_{BW}$$
$$\dot{\vec{\Omega}}_{BW} = \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} - \vec{\Omega}_{BW} x J_B \vec{\Omega}_{BW}$$

Controls & Estimation

- Three aspects toward autonomous control
  - State estimation
  - Planning
  - Controls
- Use the same closed control loop with controller (feeds inputs), UAV (provides outputs from sensors), and observer (state estimation)
- Problem is that sensors have noise
  - Inertia measurement unit (IMU) - measures acceleration and velocity, but prone to drift over time
    - Error also accumulates over time
  - GPS - current position, but noisy and often has large error due to lack of precision
- Need to develop mathematical model to figure out the state
- Simple state estimator - average values of particular variable derived from three sensors, depending on your confidence in them
- Mathematical model should be derived to calculate uncertainty also
  - Only an approximation
  - Subject to uncertainties
- If there's no uncertainty, then $y = \hat{y}$
- State observer
  - Help observe current state of rotor and brings actual results toward ideal mathematical model
- Kalman Filter
  - Optimal estimation algorithm to detect location and speed in noisy measurements
  - First used in 1960 for Apollo project
  - Used when indirecct measurement is available, but not direct measurement
  - Combines measurements and predictions
  - Two step process:
    - Prediction - system model used to calculate prior state
    - Update - use measurements to calculate and refine the prediction
    - Final output is a weighted process, depending on your confidence of model ($x = \alpha_1 x_1 + \alpha_2 x_2$)
  - Can only used for linear systems
- Nonlinear state estimators needed for quadrotors
  - Use an extended Kalman Filter - one for each linear variable

Feedback Control



- Terminology
  - **Plant** - system to be controlled, the "physics" of the problem (e.g. UAV)
  - **Sensor** - measures quantities
  - **Actuator** - controls the plant (e.g. motor)
  - **Controller** - processors sensor data to drive actuator
  - **Control law** - algorithm used by control processor to compute actuator signal (math)
- Example: hot plate temperature
  - Equation $T = T_{\text{amb}} + \alpha(P + D)$ is used for output, where P is input power, D is external disturbance, α is in °F/W
  - Using algebra, manipulate to solve for P, given a T_{desired} - $P = \frac{T_{\text{des}} - T_{\text{amb}}}{\alpha}$, assuming D is 0 W.
  - Error is $e = T - T_{\text{des}}$, goal is to have this be zero
  - In open loop control, have to manually adjust, with trial and error
  - Closed loop control equation: $P = \frac{T_{\text{des}} - T_{\text{amb}}}{\alpha} - k(T - T_{\text{des}})$
    - Increase power if too cold, lower power if too warm
    - Feed the error back into the system
    - The k term can be optimized with trial and error
  - In the end,
  $$\text{closed-loop error} = \frac{\text{open-loop error}}{1 + \alpha k}$$
  - Error is significantly less in closed-loop control when getting to a steady state
- Slide camera example
  - A camera moves along a fixed track with velocity $\vec{v}$ and camera pitch angle θ, and its target position x_{des} is at an angle γ with respect to θ
  - $\dot{x} = \vec{v}$ and $\dot{\vec{v}} = K \sin\theta$
  - Can detect $\gamma = -(\tan^{-1}(\frac{x - x_{\text{des}}}{h}) + \theta)$ based on sensors in the camera frame, where x is the current 2D position of the drone along the ground and h is its height
  - Letting $u = \theta$ (controller to plant value), goal is to find $\theta(t)$ to control the drone that makes $\gamma \to \gamma_{\text{des}}$
    - $\gamma_{\text{des}} = 0$ is the general goal
  - In a simplified version, $\theta = 0$ is is effectively ignored, and goal is to drive $\dot{x} = \vec{v}$ and find $\vec{v}(t)$
    - $\gamma = \tan^{-1}(\frac{x - x_{\text{des}}}{h})$, the angle between x_{des} and x with respect to the drone
    - Find an optimized $u = \vec{v}$ that drives $\gamma \to \gamma_{\text{des}}$
    - For a velocity command, $\vec{v} = K_p \gamma_{\text{err}}$
    - For an acceleration command, $\vec{a} = K_p \cdot \frac{d}{dt}\gamma_{\text{err}}(t)$
    - Integration of $\gamma_{\text{err}}(t)$ can be used to fix induced noise when sending velocity commands (e.g. due to wind)

PID Control

$$u(t) = \underbrace{K_P \cdot e(t)}_{\text{current}} + \underbrace{K_D \frac{d}{dt} e(t)}_{\text{future}} + \underbrace{K_I \int_0^t e(t)}_{\text{past}}$$

- Choosing gains
  1. Increase K_P until system starts to oscillate around the target, with other gains at 0
  2. Increase K_D to damp motion/oscillation
  3. Increase K_I to get rid of steady state error
- P is almost universal in all controllers
  - May cause overshoot and oscillation, however
- I resolves steady-state error
  - Risks causing saturation
- D predicts future to avoid overshoot
  - Noise can exaggerate derivative commands

Optical Flow

- Motion field - 2D projection of 3D motion
- Optical flow field - 2D, apparent motion; requires reflection
- Despite 6 degrees of freedom, can only capture a 2D field
- Example: matte sphere rotating has motion field, but not an optical flow field
- Process of visualization
  - Two frames are given as input
  - Differences are computed and visualized
- Applications
  - Video stabilization
  - Denoising
  - Super resolution
- Scanning locally (small window) for changes is more effective
- Goal is to find differences between two frames
- Need to find corners and determine if they are same
  - "Flat"region- no change in all directions
  - "Edge" - no change along edge direction
  - "Corner" - change in all directions (in uniform color shape)
  - Differences due to aliasing in images
- Windows - spacial and temporal
  - u is translation in x direction, v is "wiggle" in y
  - Not moving window around, but see how contents will change over time

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \; [\underbrace{I(x+u,y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

- Change in appearance for the shift, as a function of pixel intensities
- Trial and error required to determine appropriate u and v
- Window function is a weighted sum of the pixels in the window
- u and v are $(0,0)$ if the image is the same color throughout
- Small but nonzero shifts in u and v are interesting to find out, key to edge detection
- Taylor series - approximation of a function by taking the derivative throughout its input
- Product rule: $\frac{d}{dx}f(x)g(x) = f'(x)g(x) + f(x)g'(x)$
- For an image corner,

$$M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \approx \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

- Eigenvalues help define shape of an ellipse, in terms of stretch/compression factors
  - λ_1 is the direction of the fastest change, independent of λ_2
- Harris Detector Algorithm
  - Has translation and rotation invariance
  - No scalar invariance - zooming in on a corner reveals multiple "edges"
  - Detects glare and lighting changes easily, yet still identify object
- "Phase diagram" based on actual phase gives probablistic, fast realtime estimation of object
- Optical flow has many methods
  - Pixel difference for every pixel
  - Corner detection
- Only is apparent motion of object
  - Problem occurs when motion vectors of individual pixels are in different directions
  - The pixel correspondence problem - how to estimate pixel motion
- Assumptions made in computing optical flow
  - Brightness/intensity of pixel is constant between frames (brightness constancy)
    * Problem when lighting changes
    * $I(x,y,t) = I(x+u,y+v,t+1)$, once we know u and v
  - Solving equation with Taylor series, we get $I(x+u,y+v) = I(x,y,t+1) + \frac{\partial I(t)}{\partial x}u + \frac{\partial I(t)}{\partial y}v$, but only applicable for small motions
  - Derivation from smoothness is Gaussian
- Aperture problem - if the window is too small, apparent motion direction may not be the actual direction
  - Solution is to pretend adjacent pixels have the same (u,v) in a window with n pixels

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} I_{t_x}(p_1) & I_{t_y}(p_1) \\ \vdots & \vdots \\ I_{t_x}(p_n) & I_{t_y}(p_n) \end{bmatrix}$$

- Spacial smoothness - detect a whole surface in space, based on edges