Autonomous Air Vehicle Racing

Line Following

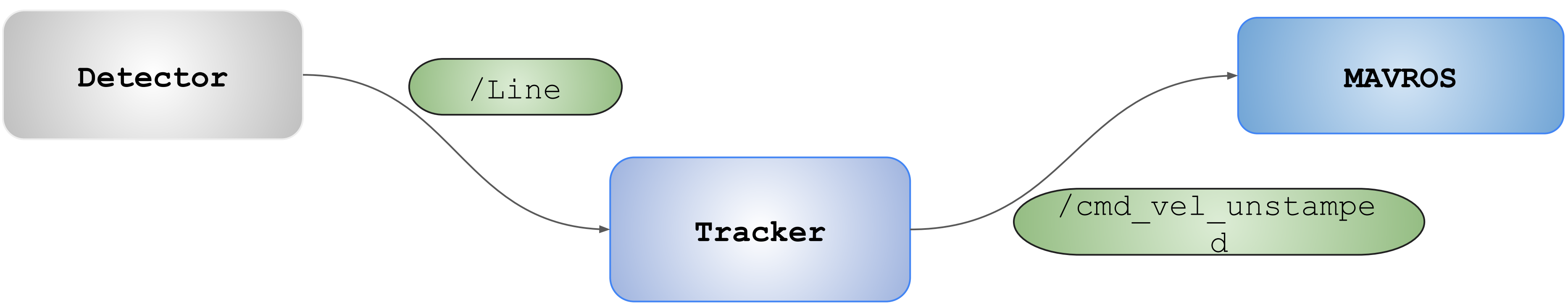
July 23rd, 2025

Outline

- How your code should flow
- Detector.py
  - writing custom msg files
  - finding the line (review)
  - parameterizing the line
  - **publish debugging images**
  - publishing the msg
- Tracker.py
  - finding the error between where you are and where you want to be
  - using that error to decide on corrected velocity commands
  - publish velocity commands
  - publish debugging images

Structure of Your Code

- **Line detection in 2 nodes:**
  - **Detector**
    - **Find the line, contour it, use linear regression to find the direction/vector of the line**
  - **Tracker**
    - **Use the Line published from Detector to tell the drone to move**



Detector

MIT | BEAVER WORKS
Lincoln Laboratory | School of Engineering

Same as 04_Downward.ipynb from last week!

Line msg

```
float32 x
float32 y
float32 vx
float32 vy
```

- Custom msg
- Holds information about your parameterized line
- **(x, y):** coordinates of a point on the line
- **(vx, vy):** vector that is collinear to your tracked line

- ***Hint:*** use the cv2.fitLine function for getting these parameters
- return None if no line or insufficient line

Losing the Line

- When you don't see a line, what happens?
  - Keep drifting from last command
  - Stop and land
  - Stop and await further instruction
  - Somehow navigate back to the line (Don't do this)
- How to tell tracker.py when you've lost the line (*hint:* use your msg)
  - Pass NaN for all the parameters
  - Add an optional (or not) boolean flag
  - What professional programmers use → **A nested msg**

Tracker

BEAVER WORKS
Lincoln Laboratory | School of Engineering

Angular Rates

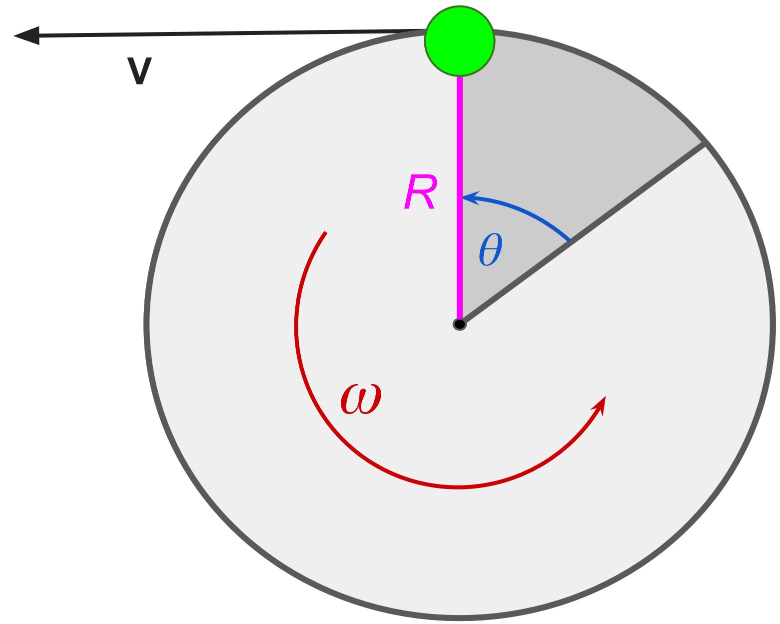
$$\omega = \frac{\Delta\theta}{\Delta t}$$

Angular Velocity is a vector value

$$v = |\omega| R = \frac{\Delta x}{\Delta t}$$

Linear velocity is a scalar value

- Which direction is positive in the angular system?



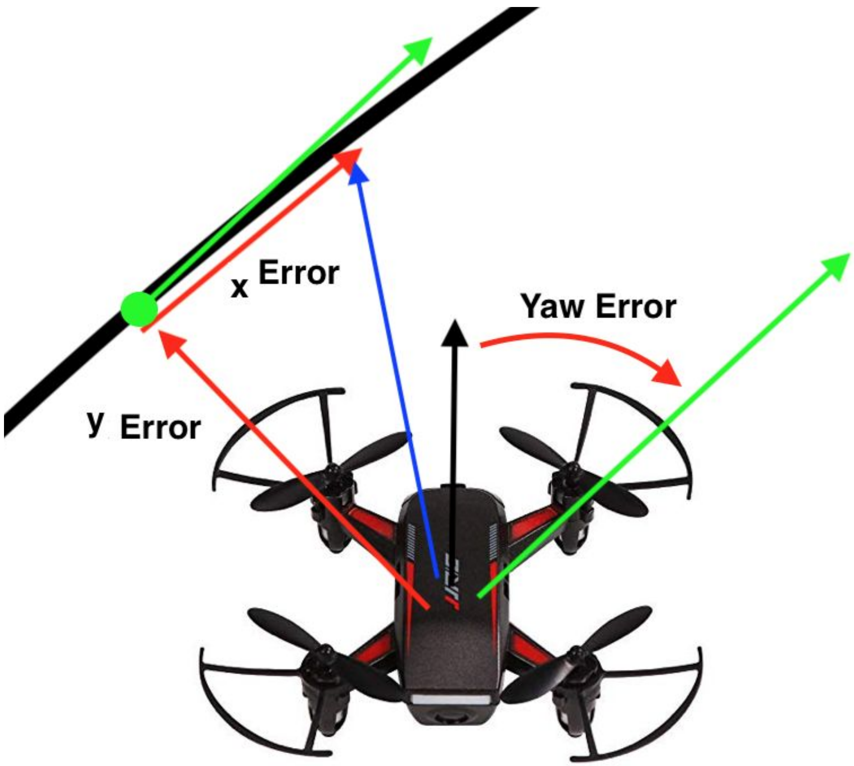
Angular System

Linear System

Error Measurements

- How would you break down an error measurement/error measurements from the downward camera into components?
- How would you write a controller using those errors?

Error Measurements



- Proportional Gains
- I and D controllers more difficult to implement

(x and y in DC frame)

Helpful Vector Math

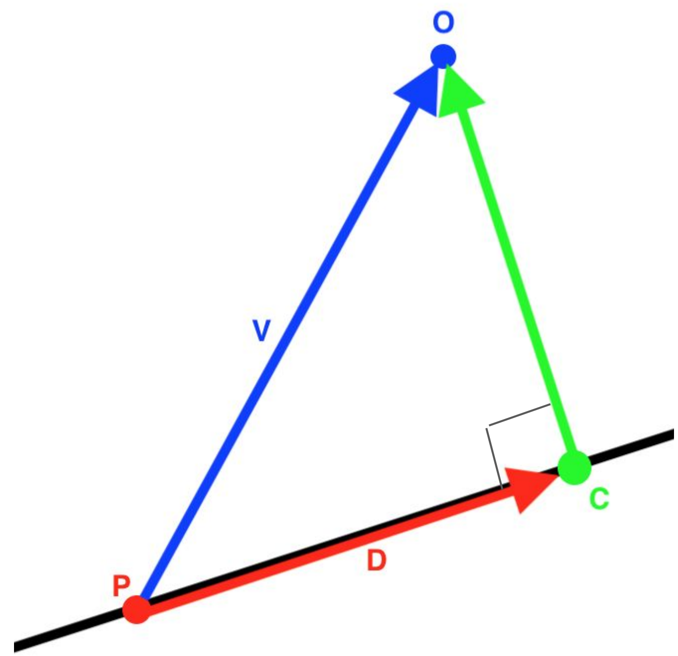
- From Line messages you get an x value, a y value, and the slope

- Unit vector tangent to the line pointing in the positive x direction:
    - Same as the positive x direction of the bu frame
    - Unit vector = (a, b) = (vx, vy) / |(vx, vy)|
    - Check if vx is positive and if not negate both vx and vy
    - Normal vector = (b, -a)

Helpful Vector Math

- How to find point on the line closest to the center of the image:
  - **V** Vector from some point P on the line to the center point O
  - **U** = Unit vector in direction of line (vx, vy)
  - **D** = Vector from P to the point C on the line we are looking for
  - **V•U** = magnitude of **D**
  - **D** = |**D**|***U** = (**V•U)*U**
  - C = P **+ D** = P + (**V•U)*U**
- You can use this point to calculate distance to line (y error) and as a base point for finding points extended a certain distance in front on the line

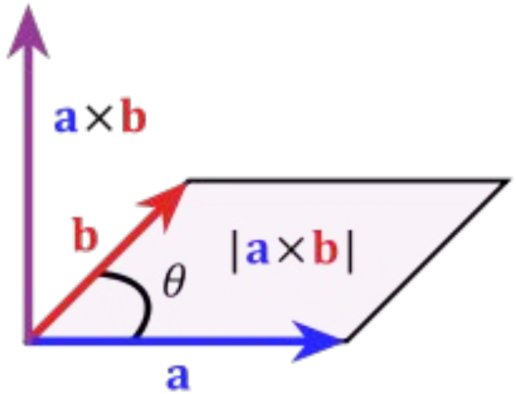
$$proj_{\mathbf{v}}\mathbf{a} = \frac{\mathbf{a}\cdot\mathbf{v}}{\left\|\mathbf{v}\right\|^2}\mathbf{v}$$



Helpful Vector Math

- Find angle between x-axis and the tangent vector to the line via cross product
    - Cross Product: a × b is defined as a **vector** c that is perpendicular (orthogonal) to both a and b, with a direction given by the right-hand rule and a magnitude equal to the area of the parallelogram that the vectors span.
    - Using cross product for theta: $|\mathbf{a} \times \mathbf{b}| = |\mathbf{a}|\,|\mathbf{b}| \sin(\theta)$

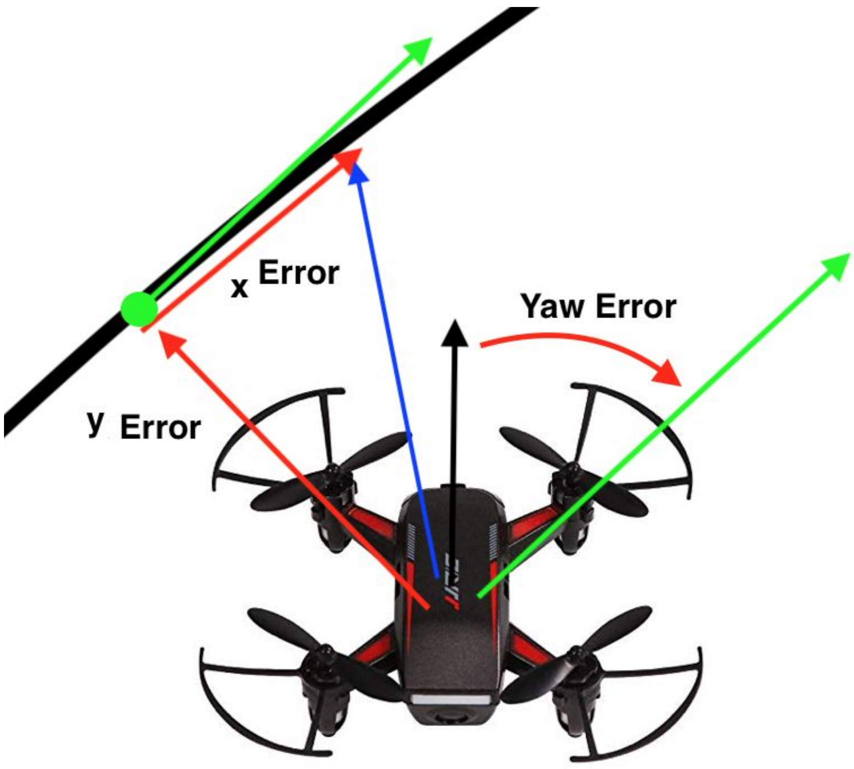
        - $|\mathbf{a}|$ is the magnitude (length) of vector **a**
        - $|\mathbf{b}|$ is the magnitude (length) of vector **b**
        - θ is the angle between **a** and **b**



OR

- atan(vy/vx) is the angle between x-axis and line.
- Use atan2(vy, vx) to avoid angle wrapping problems.
    - Atan2 takes 2 arguments and thus takes into account the sign of y and x to give the correct quadrant output
    - Result is always between -pi and pi.

Error Measurements



- You can then calculate these point(s),vector(s), and angle(s) to find numeric values for errors
- Use those values in your P controllers

(x and y in DC frame)

Commanding Drone

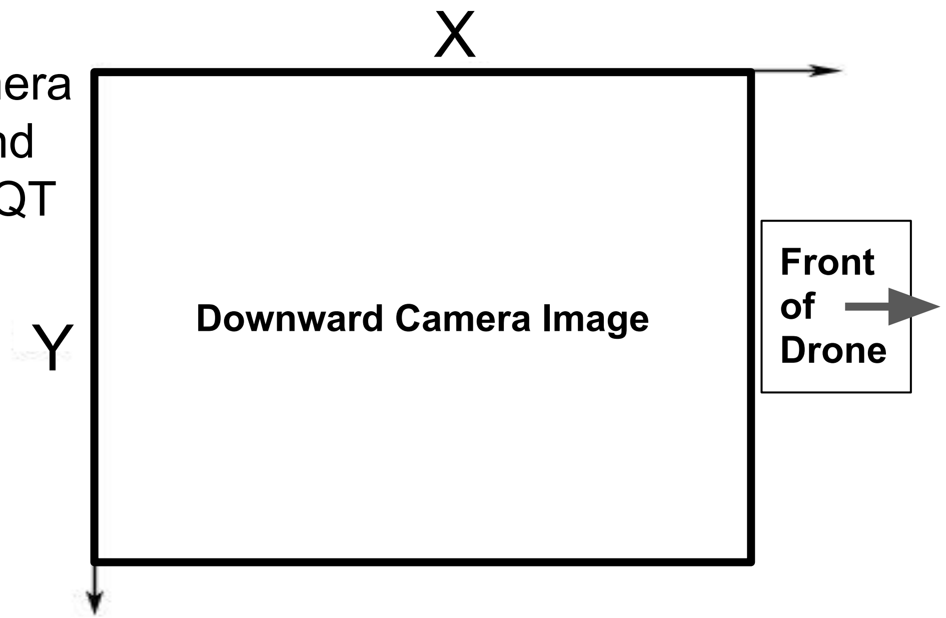
- Take Downward Camera velocities you calculated and transform them to publish as a velocity setpoint to PX4

```
vx, vy, vz = coord_transforms.get_v__lenu((self.vx__dc, self.vy__dc, self.vz__dc), 'dc', self.quat_bu_lenu)

_, _, wz = coord_transforms.get_v__lenu((0.0, 0.0, self.wz__dc), 'dc', self.quat_bu_lenu)
```

```
msg = TrajectorySetpoint()
msg.velocity = [vx, vy, vz]
msg.yawspeed = wz
```

OpenCV Downward Camera image format and as viewed on RQT ImageView:



Visualizing Vectors with OpenCV

```python
# Publish a copy of the image annotated with the detected line
if DISPLAY:
    # Draw the rectangle around the countour
    ...
    # Draw point at (x, y)
    ...
    # Draw vector (vx, vy) located at point (x,y))
    ...

    self.detector_image_pub.publish(color_msg)
```

- `DISPLAY` blocks
    - Use a global constant bool, `DISPLAY`, to tell your code whether you want it to process your images and publish annotated versions of them
    - Why?

Some
Relevant CV Functions

- `cv2.fitLine(...)` – gives x, y, vx, vy in DC frame (NED)
- `cv2.line(...)` – Draws a line on the given image
- `self.bridge.cv2_to_imgmsg(...)` – Converts a cv2 image (black and white or RGB) into the Image msg type used for ROS topics