



- *Abstract*
  - *Vision-Guided flight stability algorithm*
  - *Detects 'robust' horizons*
  - *At 30 Hz, 99.9% accurate*
  - *Detects horizon estimation errors*

- *Introduction*
  - *Technology was designed for MAV's*
  - *MAV: Micro Air Vehicles*
  - *Study conducted at the University of Florida*
    - *Aerospace Engineering Department*

- *Horizon Detection*

- *2.1 Horizon Detection Algorithm*

- *Two basic assumptions*

1. *“The horizon line will appear as approximately a straight line in the image”*
2. *“The horizon line will separate the image into two regions that will have different appearance; in other words, sky pixels will look more like other sky pixels and less like ground pixels, and vice versa.”*

- *Discussion of First Assumption*

- *The horizon is always...horizontal*
  - *Therefore: image is reduced to 2-D processing in a line parameter space*

- *Discussion of Second Assumption*

- *The division of pixels will be along that horizontal line*
- *Comparison of similar pixels is ‘left to right’*
- *Comparison of Sky vs. Ground is ‘top to bottom’*

- *Two Functioning parts of the Algorithm*
  1. *“For any given hypothesized line, the definition of optimization criterion that measures agreement with the second assumption”*
  2. *“The means for conducting an efficient search through all possible horizons in two-dimensional parameter space to maximize that optimization criterion”*

- *Optimization Criterion*
  - *Two labels: Above Horizon = Sky*  
*Below Horizon = Ground*

*SKY:*  $x_i^s = [r_i^s \ g_i^s \ b_i^s], i \in \{1, \dots, n_s\},$

*GROUND:*  $x_i^g = [r_i^g \ g_i^g \ b_i^g], i \in \{1, \dots, n_g\}.$

*Where r, g, and b represent RGB colors*

$$J = \frac{1}{|\Sigma_s| + |\Sigma_g| + (\lambda_1^s + \lambda_2^s + \lambda_3^s)^2 + (\lambda_1^g + \lambda_2^g + \lambda_3^g)^2} \quad (3)$$

based on the covariance matrices  $\Sigma_s$  and  $\Sigma_g$  of the two pixel distributions,

$$\Sigma_s = \frac{1}{(n_s - 1)} \sum_{i=1}^{n_s} (x_i^s - \mu_s)(x_i^s - \mu_s)^T \quad (4)$$

$$\Sigma_g = \frac{1}{(n_g - 1)} \sum_{i=1}^{n_g} (x_i^g - \mu_g)(x_i^g - \mu_g)^T \quad (5)$$

where,

$$\mu_s = \frac{1}{n_s} \sum_{i=1}^{n_s} x_i^s, \mu_g = \frac{1}{n_g} \sum_{i=1}^{n_g} x_i^g, \quad (6)$$

and  $\lambda_i^s$  and  $\lambda_i^g$ ,  $i \in \{1, 2, 3\}$ , denote the eigenvalues of  $\Sigma_s$  and  $\Sigma_g$  respectively.

- *To meet 'real time' processing constraints:*

Given a video frame at  $X_H \times Y_H$  resolution:

1. Down-sample the image to  $X_L \times Y_L$ , where  $X_L \ll X_H$ ,  $Y_L \ll Y_H$ .
2. Evaluate  $J$  on the down-sampled image for line parameters  $(\phi_i, \sigma_j)$ , where,

$$(\phi_i, \sigma_j) = \left( \frac{i\pi}{n} - \frac{\pi}{2}, 100 \frac{j}{n} \right), 0 \leq i \leq n, 0 \leq j \leq n \quad (8)$$

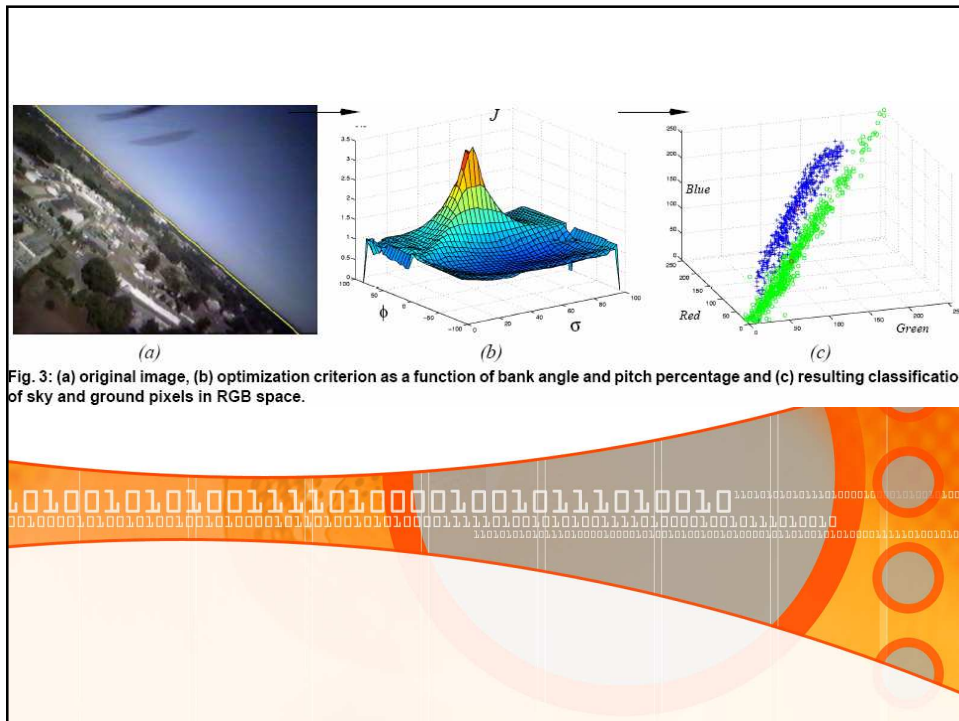
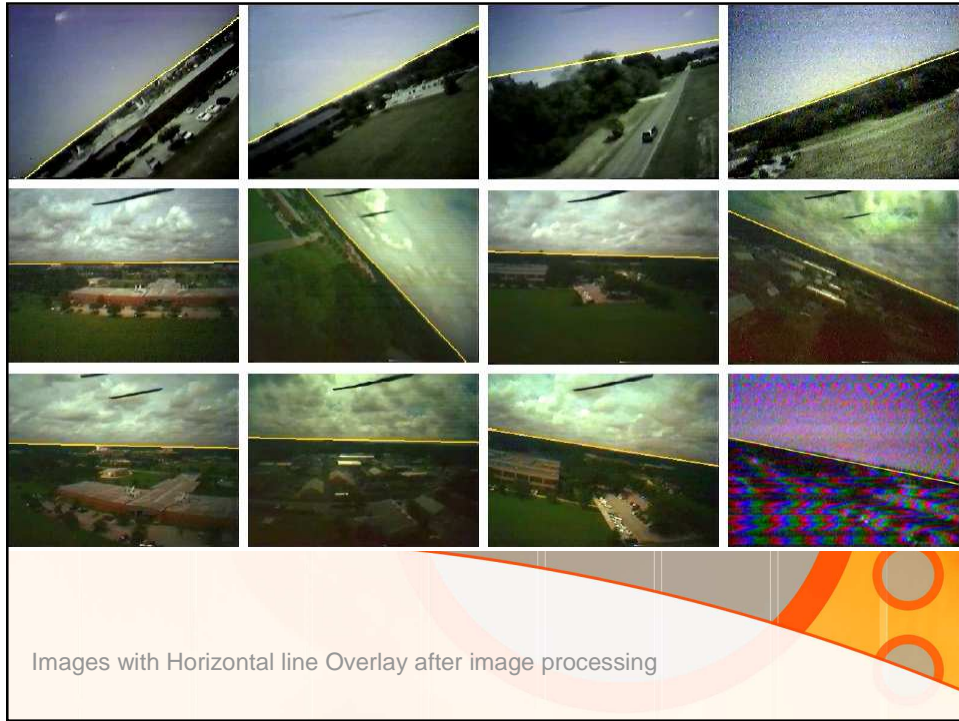
3. Select  $(\phi^*, \sigma^*)$  such that,

$$J|_{\phi = \phi^*, \sigma = \sigma^*} \geq J|_{\phi = \phi_i, \sigma = \sigma_j}, \forall i, j. \quad (9)$$

4. Use bisection search on the high-resolution image to fine-tune the values of  $(\phi^*, \sigma^*)$ .

- *Computing Power for Algorithm testing*
  - *30 Hz run speed*
  - *900 Mhz x86 Processor*
  - *Down-sampled image of  $XL \times YL = 80 \times 60$  res*
  - *Search resolution of  $n = 36$*
- *Slightly reduced performance at  $XL \times YL = 40 \times 30$   
 $N = 12$ , and  $XH \times YH = 160 \times 120$*

- *Why process this image in 'real' time?*
  - *Although the horizon may not change for a sizeable time delay, errors in the system will be accounted for*
    - *In other words, although the image will not change for several seconds (something we can take into account when allocating processing time and resources to) if there is an error, it will be corrected immediately*
  - *HOWEVER...*
    - *If we grab and process an image every second, or every two seconds, and use the average of those values every 5 seconds, we may still have an accurate model without the use of 'real' time processing power*



- **3.1 Extreme attitude detection**
  - If the MAV is pointing straight up or down, no horizon will be visible...
    - Bearings will still need to be retained
  - Two Options
    1. utilize recent images and reorient
    2. utilize recent 'horizon line' estimations

- **Examples:**
  - If last four images show an increasing split of Sky and Ground, where Sky % < Ground %, we know the craft has turned downwards
  - If last four horizon estimates show the horizon line suddenly 'dropping' in altitude (Y=10 to Y=7, to Y=4...etc) we know the craft has turned upwards

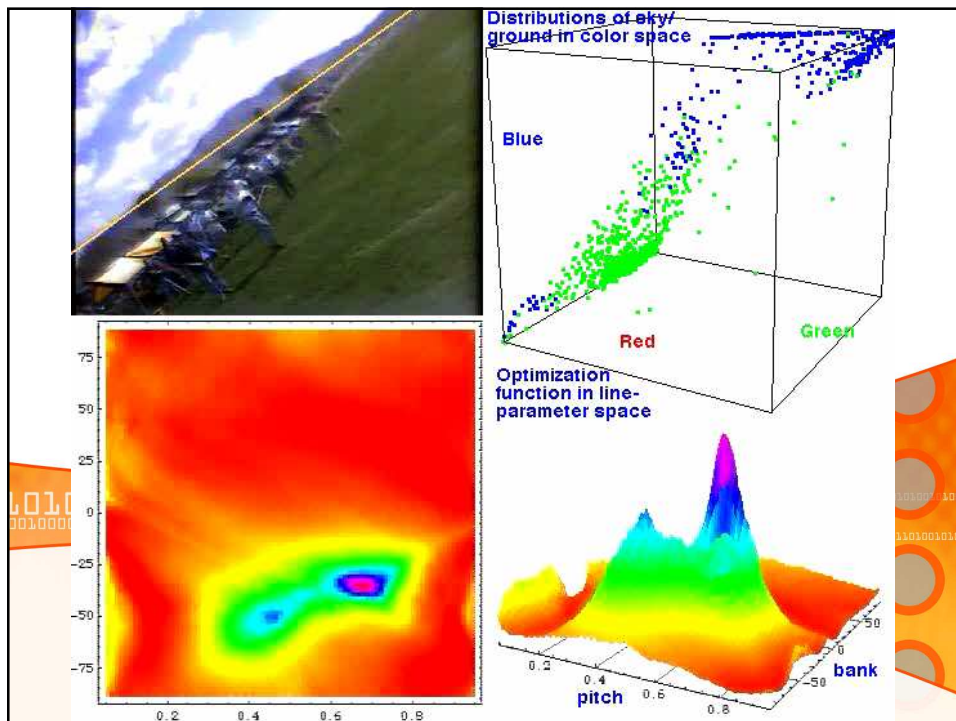
**Table 1: Extreme attitude detection**

<i>case</i>	<i>condition</i>	<i>conclusion</i>
1	$D_1 < D_2$ and $D_3 > D_4$	valid horizon present
2	$D_1 > D_2$ and $D_3 > D_4$	all ground
3	$D_1 < D_2$ and $D_3 < D_4$	all sky
4	$D_1 > D_2$ and $D_3 < D_4$	upside down

- *Relevance*
  - *If we are climbing or descending a steep hill*
  - *If an object is blocking our view*
  - *If the weather conditions are overcast, such that the 'grey' of the sky and the 'grey' of an urban environment start to mesh*
    - *But if the weather starts out overcast, what do we have to compare to?*

- *Weather conditions for the Urban Challenge*
  - *The algorithm will have to be fine tuned to look for slight color variations in a potentially color-grey scale image*
    - *Color Grey Scale: When a color image displays mostly grey, black or white pixels*
  - *If color definition is set to 'overcast-urban' the algorithm will use a 'finer' color variation to decipher horizon location*

- *After extreme altitude and error detection...*
- *Passed through a Kalman Filter*
  - *“Provides an optimal estimate of a system's current state, given a dynamic system model, a noise model and a time series of measurements”*
  - *Removes high-frequency noise, eliminates radical single frame errors*
  - *Elements unnecessary small control surface deflection*





*For more information, visit <http://mil.ufl.edu/~nechyba/mav/>*

**Thank You**

A decorative footer graphic consisting of a horizontal band with a gradient from light orange to dark orange. The band contains a pattern of white binary code (0s and 1s) and several overlapping circles of varying sizes and shades of orange and grey.