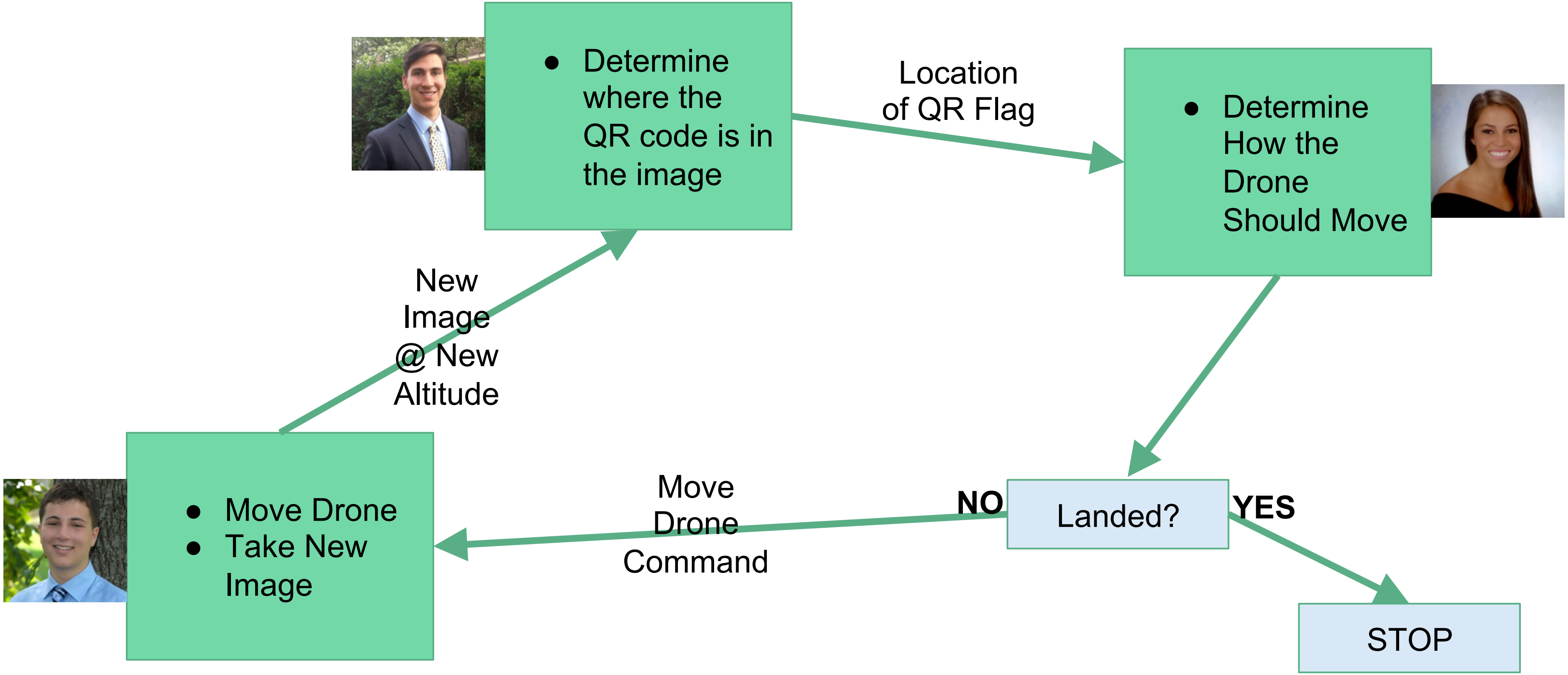
# Penn Prime Air

Matt Schulman
Josh Pearlstein
Steffi Maiman

# We Have Focused on Harnessing Optics for Uber-Accurate Landing Capability

- **Uber-Accurate Landing**
  - **Went forward with GPS + Optics Approach**
  - **Designed Software System Components**
    - **Interfacing with Drone**
    - **Recognizing Location of QR Flag/Poster in Image**
    - **Algorithm to Update Location of Drone**
- Integration of Software System Components
- Testing and Refinement
- Drone Routing

# Drone Landing Software Design Flowchart



- Determine where the QR code is in the image

Location of QR Flag

- Determine How the Drone Should Move

Landed?

YES

STOP

NO

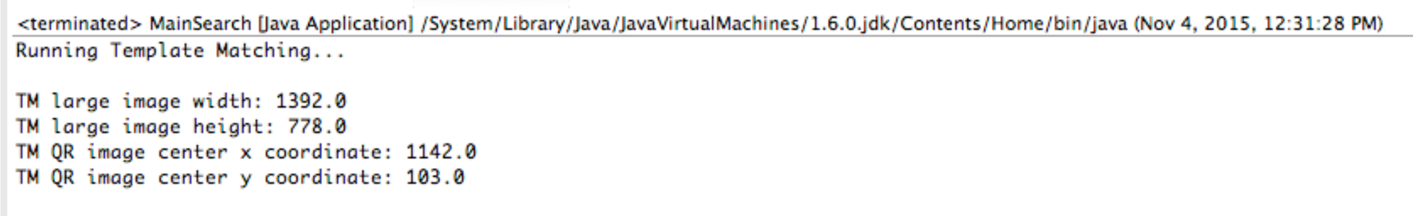
Move Drone Command

- Move Drone
- Take New Image

New Image @ New Altitude

# QR Flag Optics Leverages OpenCV Computer Vision

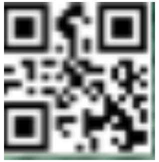
- OpenCV: BSD license → free for both academic and commercial use
    - common infrastructure for computer vision applications
- Progress: Template Matching in Java



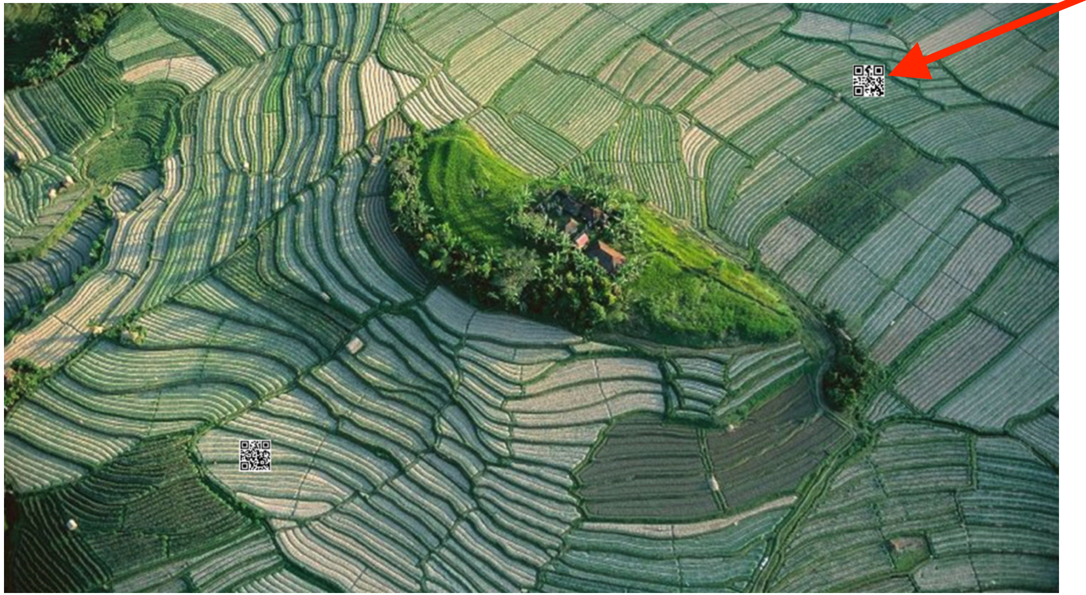
```
<terminated> MainSearch [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Nov 4, 2015, 12:31:28 PM)
Running Template Matching...

TM large image width: 1392.0
TM large image height: 778.0
TM QR image center x coordinate: 1142.0
TM QR image center y coordinate: 103.0
```

## INPUTS:

## OUTPUT:



Template



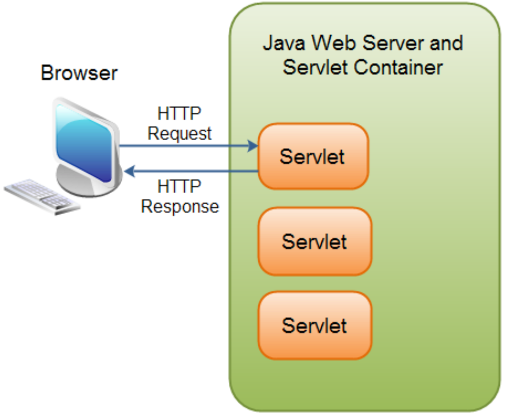
Source

(1142,103)

# The OpenCV Features2D Framework Will Make the Optics QR Flag Recognition More Robust

- 1st Next Step: Features2D Framework
  - “Template matching...is not good when your object is rotated or scaled in scene”
  - Goals with Features2D Framework
    - 1) Avoid worrying about expected # of pixels in QR code
    - 2) Handle rotated QR codes seamlessly
- 2nd Next Step: Java Servlet API

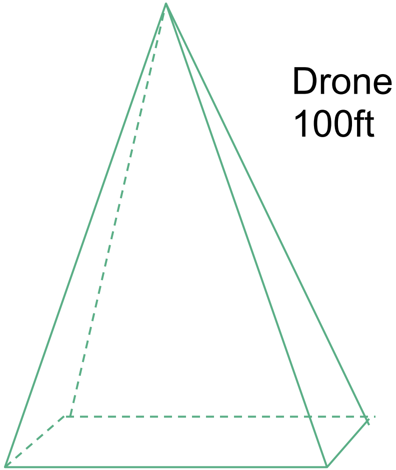
stackoverflow



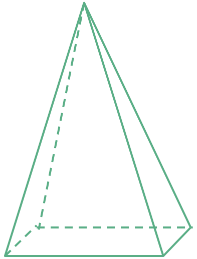
# Drone Movement Algorithm

- Algorithm comprised of 3 components:
  - Recursive function for movement
    - Uses a calculated pixel-to-foot ratio
    - Drone is moved to location above QR code
    - Lowers drone to proportion of altitude
    - will be using an experimented zoom ratio (between 0 and 1) depending on the image quality
  - Landing function
    - Called when the drone reaches an altitude threshold ~3-5 feet
  - New Image function
    - Better accuracy
    - Process new image at every altitude change
    - Re-calculate pixel-to-foot ratio

# Drone Movement Algorithm Walkthrough Example



Drone Altitude:
100ft



Drone Altitude:
(100*zoomRatio) ft

Original Image:
width = x
height = y

‘Zoomed’ Image:
width = x * zoom
height = y * zoom

# Drone Interfacing

- Currently interfacing with drone through an iPhone
  - Next steps include moving that interface to drone hardware
- Fully implemented all drone movement
  - Adding GPS waypoints
  - Full Gimbal 360 control
  - Speed, altitude controls
- Currently: Drone takes image sends to iPhone which sends to remotely hosted image processing server which sends results back to iPhone for control commands
- Next steps: Drone takes image, processes on board, decides which commands to take

# High Level Next Steps

- **Integration of Software System Components**
  - Refine Optic Component
    - Template Matching → Feature2D
    - Rotated Images Compatibility
  - Java Servlet API for Optic Component
    - N64 → image conversion
    - QR creation from QR code
    - Possibly look into writing optics openCV algorithm in ios app to avoid API calls
  - Test with Drone
    - Integration pains?
    - Ideal magnitude of Drone movement commands (tradeoff between speed & accuracy)
    - Refinement of Drone movement algorithm
- Testing and Refinement
- Drone Routing