Automatic Startracker Optimization Framework

Andrew Tennenbaum
Dr. John Crassidis (Advisor)
University at Buffalo (SUNY)
• University at Buffalo Nanosatellite Laboratory
  – Volunteer only student research lab
  – AFRL UNP and NASA USIP

• Three missions under development:
  – GLADOS (Glint Analyzing Data Observation Satellite): Gathers data from glinting objects in GEO using multiband cameras (RGB & IR)
  – SORA (Spectral Observation for Reflectivity Analysis): Collects lightcurve data on objects in LEO using a guider camera and a spectrometer
  – Linksat: Characterize the radio environment in LEO using an SDR
Background

- GLADOS & SORA precise attitude info to perform mission

- Simplest way to do this – startracker
  - Compare star patterns in an image to DB of known star positions
  - Use this as a reference to determine orientation

- Commercial startracker -> $$$$

- Developing startracker -> difficult
• Build an open source framework which will simplify & automate process of developing startrackers from off the shelf cameras

• Lower the barrier of entry for cubesat missions wishing to incorporate startrackers

• Still working out the details of when and how the framework will be made available
Framework Components

- **Problem:**
  - Create the best startracker possible given a specific camera
  - Determine its performance

- **Best means:**
  - Highest chance of obtaining a match
  - No false positives

- **Can break the task down into 3 phases:**
  1. Calibrating camera
  2. Generating constellation database
  3. Testing performance under realistic conditions

- **Exact details depend on the startracker algorithm that we are using**
Startracker Algorithm

- Two most common startracker algorithms:
  1. Pyramid: robust against false positives
  2. Geometric voting: robust against false negatives

- Pyramid algorithm: identifies groups of stars in image using pregenerated constellation database
- Geometric voting: selects most likely ID for each star based on distances to surrounding stars

- Framework designed with pyramid in mind
  - Robustness again false positives more useful
  - Can be extended to work with other algorithms
• Calibration Phase: optimize startracker algorithm for specific camera

• Take >3 different images of the stars taken with our camera and find:
  1. Background noise
  2. Field of view size
  3. Average star position error
  4. Relationship between the brightness of stars in our image and stars in our database

• For the last 3 steps, need to know which stars we are looking at in uncalibrated images!
To accomplish this, use astrometry.net
  - Open source tool
  - Used in astronomical community

On some flight computers, possible to perform automatic recalibration
  - Compensate for sensor degradation, etc
Background Subtraction

- Startrackers need to filter out:
  - Background noise
  - Dead pixels

\[ SNR = \sqrt{N} = \mu \]

- Typically subtract a darkfield image from the raw camera images

- Able to reconstruct by taking median of different images of the night sky

- Can be done using only images that are available in orbit
• Take star survey such as the Hipparcos catalog, filter out stars that are:
  – Too dim
  – Too close together
  – Highly variable
  – Subject to parallax effects (too close to us)

• Update positions of stars using the current year
  – Compensate for stellar motion

• Generate constellation database
  – always at least one feature in our field of view at all times
1. Validate against a set of test images taken with our camera
   - We can solve them?
   - Results agree with the results from astrometry.net?
   - This stage can be performed in flight!

2. Hardware in the loop camera image simulator
   - Use computer with graphics card
   - Display realistic moving starfields
   - Solve with the startracker in realtime
   - Can be used to validate attitude control algorithms which depend on the startracker!
Results

- Completed the process of calibration, DB generation, validation using:
  - Consumer grade webcam
  - Gigevision Smartek flight camera

- Using either camera we are able to solve the lost in space problem in:
  - 30 milliseconds on an old laptop
  - 300 milliseconds on our intel edison flight computer

- Already quite fast, but should be possible to improve by an order of magnitude by optimizing the algorithm
Using OpenCV centroiding:

bg_sample/w1.png
Time: 0.0321140289307
DEC=60.4956492831
RA=179.157306008
ORIENTATION=178.979274135

....

bg_sample/w8.png
Time: 0.0311989784241
DEC=76.0048732316
RA=-179.516300048
ORIENTATION=179.343238004
andrew@spaace:~$
Questions?