Recall:

1. **Mapping**
   a. Developed code for basic mapping pipeline
   b. Manage to map part of Klaus with images captured at specific poses
   c. Encounter problem when trying to map a close environment
2. **Localization**
   a. Developed code for basic mapping pipeline
   b. The result is wrong
3. **Proposal**
   a. Written a proposal focus on autonomous blimp
   b. Revise the proposal

Plan:

1. **Proposal**
   The proposal will be organized in 4 parts:
   a. Autonomous Blimp Design
   b. Mapping - Structure from Motion (brief)
   c. Localization based on prebuilt feature map (Currently)
   d. Control (brief)

2. **Mapping**
   a. Test on different inputs to find the problem
3. **Localization**
   a. Literature Review
4. **Control**

Astrobee

Related Papers:

Brian Coltin, et al, Localization from Visual Landmarks on a Free-flying Robot, 2016 IROS.
Pyojin Kim, et al, Robust Visual Localization in Changing Lighting Conditions, 2017 ICRA.
Maria G Bualat, Astrobee: A New Tool for ISS Operations, 2018 SpaceOps Conference.
Lorenzo Fluvkiger, Astrobee Robot Software: Enabling Mobile Autonomy on the ISS, 2018.

Source Code:

https://github.com/nasa/astrobee.git

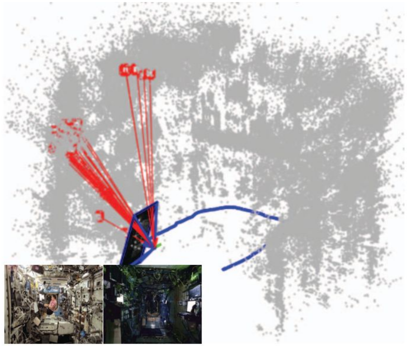

Fig. 1. Astrobee Video:https://www.youtube.com/watch?v=G64Fs8UVUYE

a. Mapping

- Collect Images (Sequence)
- Surf Feature Detection
- Feature Matching
- Tracking Building
- Initial Map Guess
- Incremental Bundle Adjustment
- Global Bundle Adjustment
- Rebuilding with BRISK
- Registration
- Bag of Words Database

b. Localization

EKF

- Visual Observation
  1. Camera Coordinates
  2. Matching 3D landmarks
- Optical Flow (A list of 50 optical flow features is maintained at all times.)
- Handrail Measurements
- IMU
- AR Tags

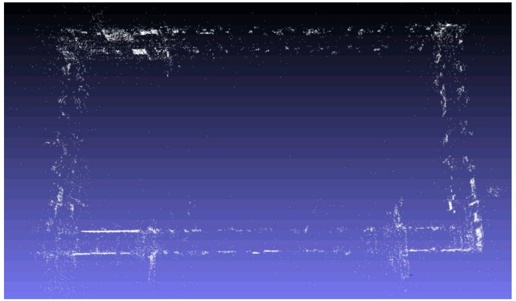

Fig. 2. Prebuilt Map

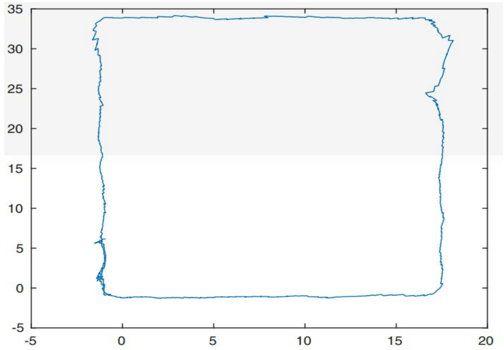

Fig. 3. Localization

Trajectory Estimator (Localization) Basic Pipeline Development:

Pipeline:
Load the mapping result, includes landmark points and their corresponding normalized averaged descriptor values.

```
trajectory_estimator(initial pose, map)
{
    trajectory = [initial pose]
    while(next_frame)
    {
        undistort_image(next_frame)
        get_pose_from_trajectory(trajectory)
        superpoint_extraction(next_frame)
        landmark_projection(pose, map)
        landmark_association(superpoints,
                     projected_landmarks)
        pose_estimate()
        trajectory.append(new pose)
    }
}
```

Top Level Unittest - Result (pass)

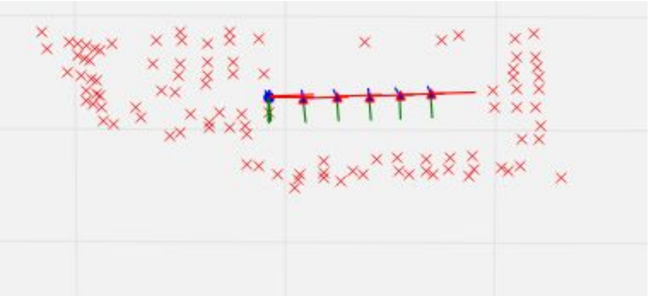

Fig 1. Along the z axis

Input with collected images - Result (wrong)

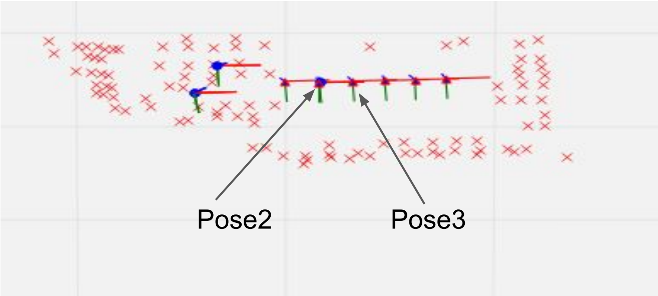

Fig 2. Along the z axis

Red crosses are landmarks. Poses (triangles at the pose origins) are poses generated through the mapping pipeline, which are the ground truths in this unittest.

Inputs:
a. The map - Landmark Object (3 lists: landmarks, descriptors, keypoints)
b. The first image from the input image set of the mapping pipeline

Output:
a. A trajectory of one pose (circle) - list

Inputs:
a. The map
b. 6 images collected along the x axis from Pose2 to Pose3.

Output:
a. A trajectory of six poses (circle)

Analysis:
a. too few landmarks
b. Parameters effect - matched feature key point distances, matched descriptors L2 distances
c. The normalized averaged descriptors may cause matching problem and required further analysis.

Localization:

Ideas:

a. Descriptors type
b. Map data can be stored in KD tree for future fast matching
c. Descriptor Matching(the type of distance)
d. Filter Outlier

Matching Progress

i. Use keypoint position to filter landmark points that can not be projected in the current view
ii. Find feature matches
iii. Filter feature matches with essential matrix
iv. Some people store color information

Mapping:

Reconstruct Library with 4X8 Images

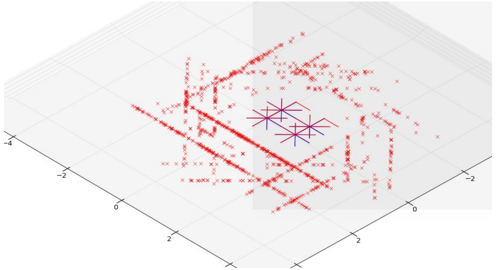

Fig 1. Top Down

Reconstruct Library with 2X3 Images

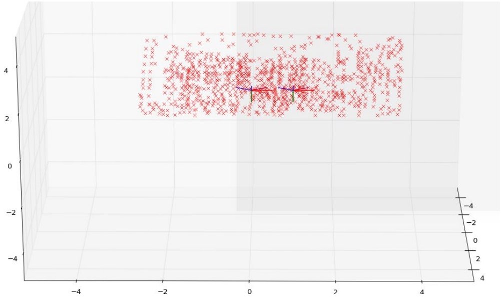

Fig 2. Forward

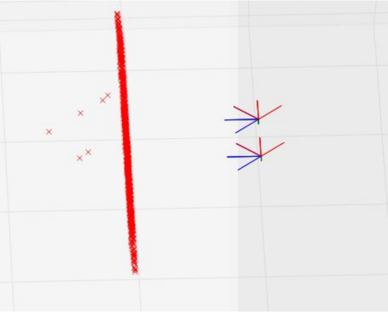

Fig 3. Top Down

Problems:

1. **Jing C++ code**
   a. Has problem when two images do not have any matches
   b. Have problem with large size images
2. **GTSAM error**
   a. Only run Optimization

```
# Optimization
optimizer = gtsam.LevenbergMarquardtOptimizer(graph, initial_estimate)
sfm_result = optimizer.optimize()
```

   b. Run Optimization and Marginalize

```
# Optimization
optimizer = gtsam.LevenbergMarquardtOptimizer(graph, initial_estimate)
sfm_result = optimizer.optimize()
# Check if factor covariances are under constrain
marginals = gtsam.Marginals(  # pylint: disable=unused-variable
    graph, sfm_result)
```

```
RuntimeError:
Indeterminant linear system detected while working near variable
8070450532247928845 (Symbol: p13).

Thrown when a linear system is ill-posed.  The most common cause for this
error is having underconstrained variables.  Mathematically, the system is
underdetermined.  See the GTSAM Doxygen documentation at
http://borg.cc.gatech.edu/ on gtsam::IndeterminantLinearSystemException for
more information.
```