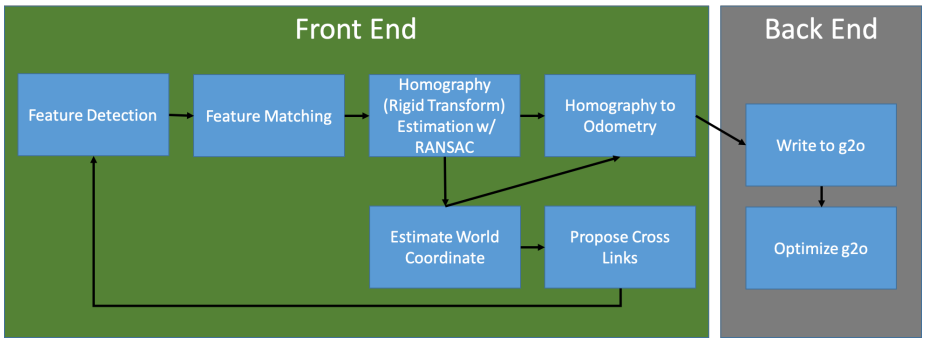
Visual Odometry Pipeline

The goal of this project was to build a visual odometry pipeline and validate the implementation using the Pizarro Dataset. While the full pipeline was still a bit buggy, all of the major components of the pipeline were implemented.

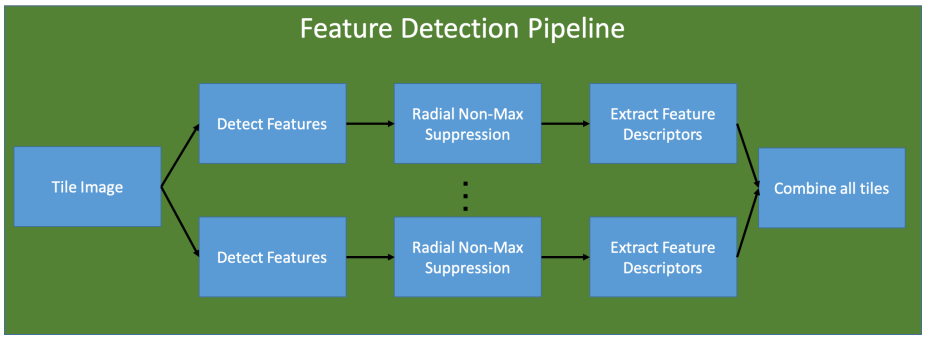


The entire pipeline can be split into a front and back end. The front end is responsible for feature detection, feature matching, and homography estimation. Once a homography between two images is calculated, an odometry measurement is extracted from the rotation and translation components of the homography matrix. Because we assume/know the motion of the robot was fairly level and straight, we can make the assumption of our images being rigid-body transforms, allowing only translation and rotation of the image. Therefore, the odometry is extracted from a homography matrix of the following form.

$$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

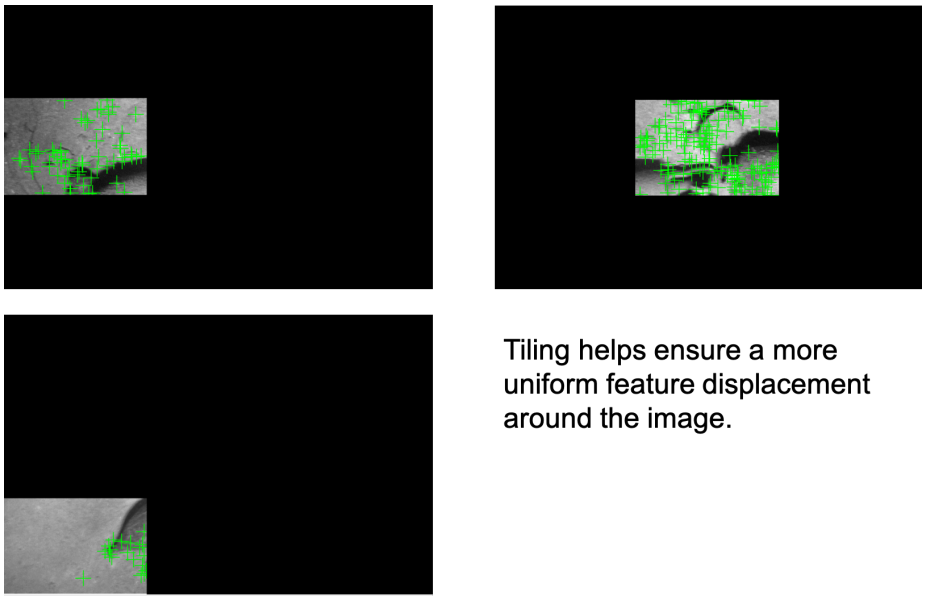
Once odometry measurements are calculated from the homographies, this data can be written into a G2O compatible format which is used to optimize a non-linear least squares problem, hopefully improving the estimations of our vertices. Once the initial forward pass of the data set is complete, we can continue to propose additional crosslinks between images which were not originally found. In our case, we made the assumption that sequential images were linked (in the forward pass). After the first pass we can attemptto estimate overlaps between images and propose new cross links to be added. These links are then fed into G2O to further optimize our graph.

Feature Detection Pipeline



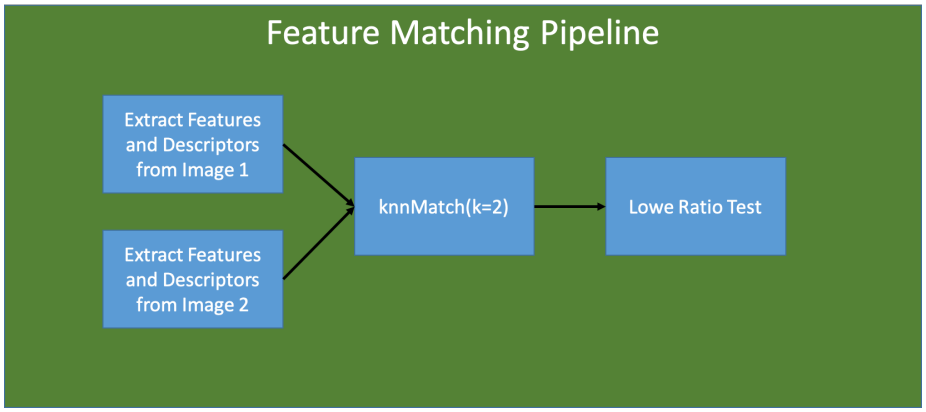
The feature detection pipeline has five main steps. The first component of pipeline is tiling. Tiling is important because it ensures that we get a set of well distributed features across our image. This is especially important for this data set which is fairly sparse and devoid of features. For example, image 0623.

Tiling splits the image into chunks which are examined for features independently. In each tile, we take a maximum of N features. After the initial features are detected. We run a radial non-maximum suppression to eliminate features that are too close together. This helps space features out and prevent overlaps which can make feature matching much more difficult down the road.



Once we've aquired a set of interesting features, we can compute the associated descriptors at each feature to be used in the matching pipeline. The features from all of the tiles are then combined at the end to complete the total set of features for the entire image.

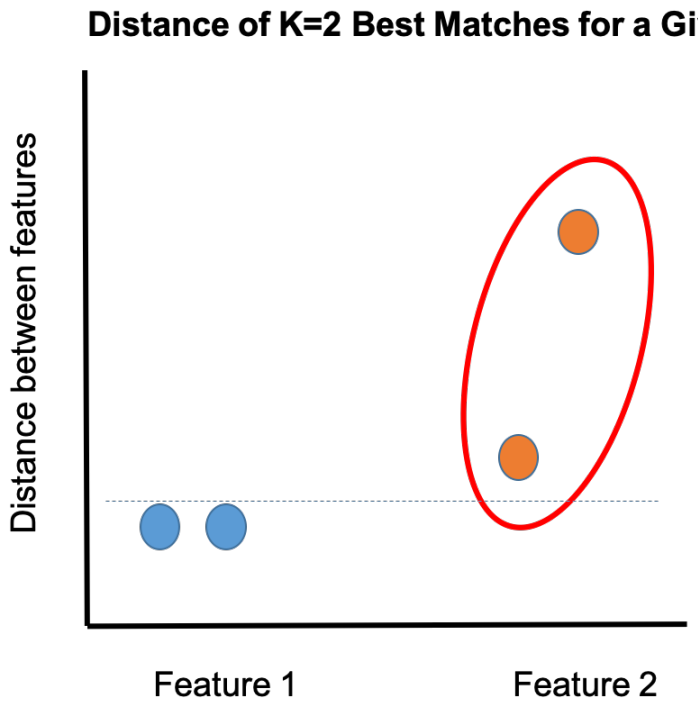
Feature Matching Pipeline



With features detected for every image, we proceed to attempting to match a set of features in each image. Given the two descriptors, we can use the OpenCV feature matchers which propose the best match for a feature where the "best" match is the match that minimizes some distance function between the two patches. Unfortuantely, this doesn't always return valid matches, especially for

features which are not super unique. When the matcher is created, you have the opportunity to specify `crossCheck=True` which only takes a match if the best match from feature A to B is the same as the best match from B to A, i.e. both parties agree that they are the best match for the other. Experimentally, this did not do much to improve the match rate.

An alternative is to use the Lowe Ratio test to eliminate matches which weren't "unique" enough to be considered a good match. The Lowe Ratio test compares the ratio of the top two potential matches for a feature and only admits matches where the "Best" option is significantly better than the rest.

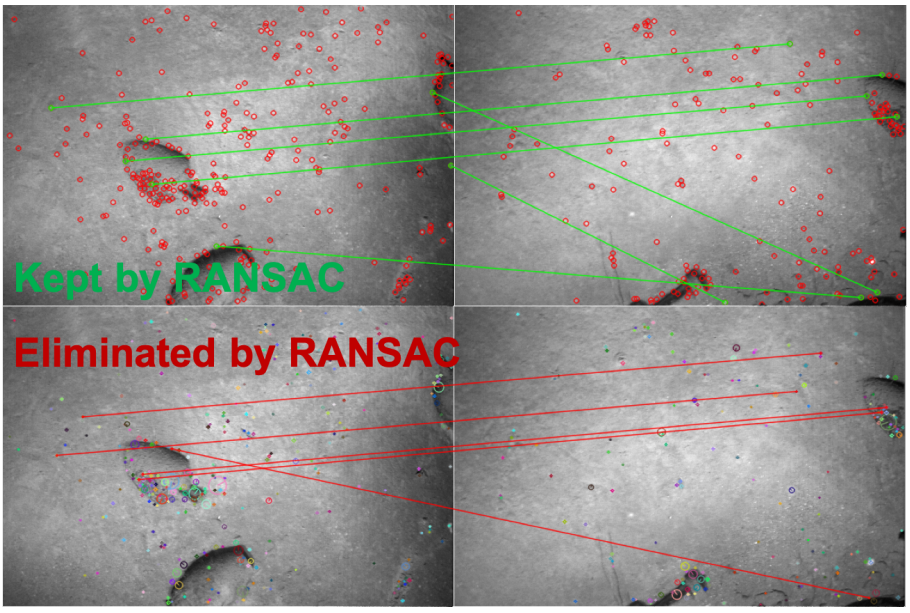


In this example, we see two potential feature matches between two images. Feature 1 shows a feature that has two very good matches to another image, i.e. the distance between the feature and the proposed match is very low. Feature 2 show a feature match with a slightly higher distance than the matches in feature 1, but the next best match is significantly worse than the first. If we were to simply take the "N best features" (computed by minimum distance value), we would choose to take feature 1. However, considering the that feature has TWO very "good" matches in feature two, how confident are we that the "best" match is really the best match? Alternatively, Feature two exhibits characteristics that are much better in determining a match. Feature 2 is much more confident in its "Best" match than its second best match.

Experimentally, using the Lowe Ratio test to eliminate bad features performed significantly better than using the straight "best feature" function.

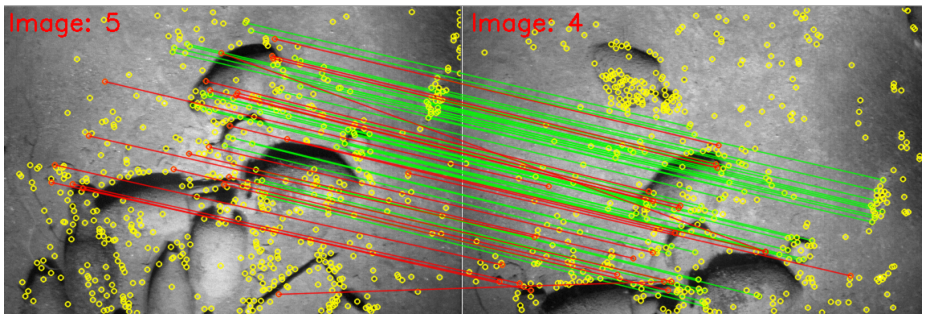
Homography and RANSAC

Feature matches in hand, we can calculate a homography between two images. While we initially started with the `cv.findHomography` command with RANSAC, we ran into a couple of edge cases which caused poor performance.



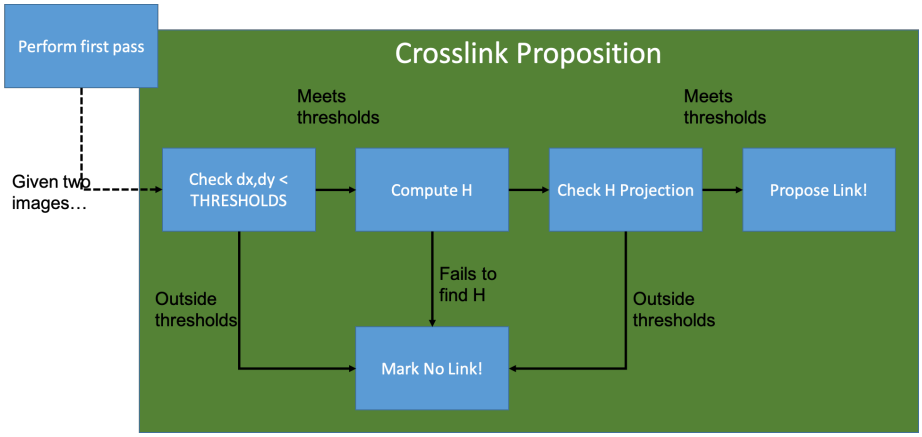
In this example, we see a number of "bad" matches were kept by RANSAC while good matches were being thrown away. While we weren't able to ultimately determine why this was the case, the leading hypothesis was that the `cv.findHomography` is looking for and eliminating matches to achieve a full projective transform as opposed to a simple rigid body transformation. This combined with our low number of matches could mean that the proposed matches/elminations are a better match for the found transform, as opposed to the desired/"intuitive" rigid body transform we see.

Instead of searching for a full projective transform, OpenCV had a function `cv.estimateRigidTransform` which only searches for a 2D rigid body transform.



Here we see that the matches (and eliminations) make more sense and ultimatley find a satisfactory transformation.

Crosslink Proposals



After finding the initial estimation of the first 6 images, we can begin searching for cross links bewteen images. In our initial pass through, we work with the priori that two sequential images must have overlap and therefore, will match. This allows us to make the image sequence chain `0 -> 1 -> 2 -> ... -> 5`. To improve our estimation, we also want to propose a number of cross links between images to add constraints on image states.

The first metric for link proposal is to test whether the bounds of the image fall within a resonable distance of another image based on the pose estimates from the first foward pass through the image chain. For example, the bottom left corner of image 0 starts at (0, 0). To check if another image may overlap with the first image, we take the coordinates of the the second image and test if the absolute distance between the two points are below an acceptable threshold. The corresponding point on image 2 (the first non sequential image for image 0) falls around the point (20, 250). Here we see a delta x of ~20 pixels and a delta y of ~250 pixels. We know the size of an image to be 576x384 so we can set our

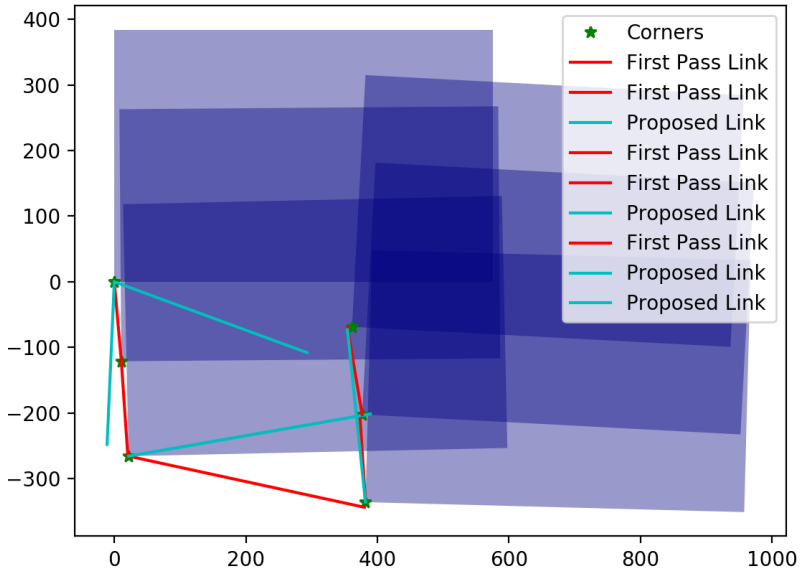
thresholds accordingly. Based on these numbers, it is very resonable for image 2 to have some overlap with image 1. If the distance between the two images was too great, we would mark the link as having been too far and attempt to match the next image.

The seconds stage of the cross link proposal pipeline is to attempt a feature match between the two images. If there are enough good feature points between the two images, we will find be able to find a homography. If not, we'll mark it as not having been matched.

Finally, we'll check the distance of the last projection. This last check acts as a santity check to make sure we don't have a completely wrong projection (wrong meaning a bad match). Using the estimated homography we project the old image into its new position comparing where it fell to the first projection. If the image position differs by more than some threshold, we call the projection bad and mark it as a failed match.

The structure of this problem is that detecting a link from image N to M is the same as detecting the reverse link. As a result, we only have to test the first N/2 images against every image that comes after it.

First Results

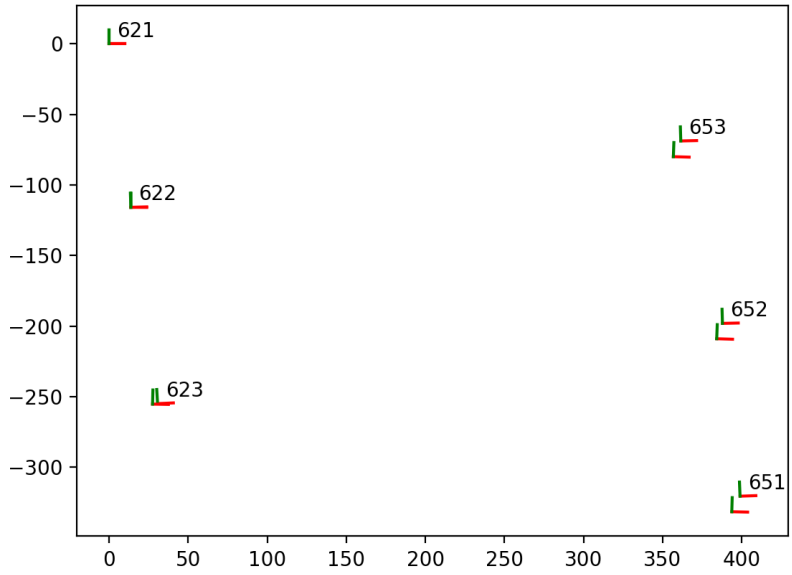


This image shows the first set of results of the image passing. Unfortuantely, I had trouble with the `cv.warpPerspective` code warping images into negative

pixel space. For sake of time, I just applied the transformations to rectangles and plotted the links and the proposed links. The red links show the links found between the first six images. The blue links represent potential image links between non sequential images. In this pass, we found potential links between the 0->2, 0->5, 2->4, and 3->5. These links are fed into the g2o file used to optimize the relationships between images.

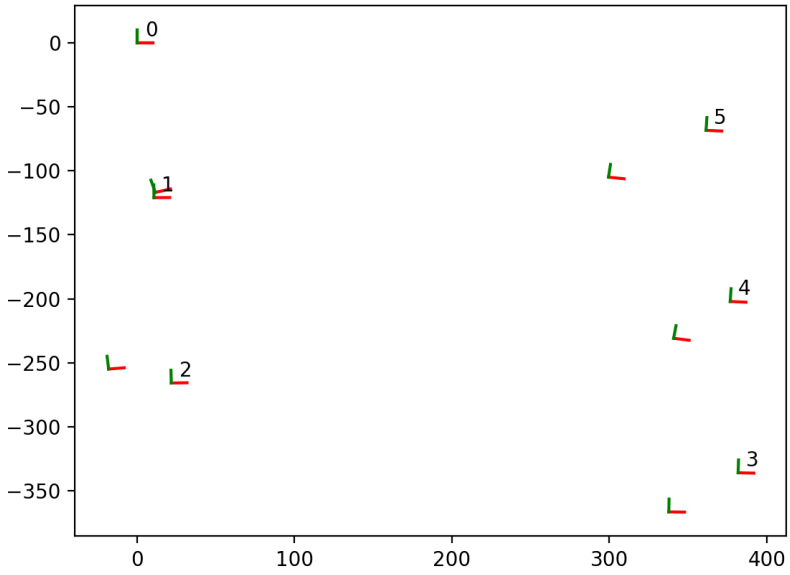
g2o Optimizations

Here are the results of running the results through g2o. The first image is the result of optimizing the given matlab mat files on the six image data set.



Here we see g2o behaving exactly as we would expect it to, with the images further away from the origin being pulled in towards the base image column. This shows that g2o is correcting for the error/drift in the image.

The second image is the result from the six image data set using the cross links found between the image. These results show a much larger pull on the right column images towards the first set. Additionally, we see image 2 also experienced this pull. This is likely because of the link 0->2 which placed a new link between the images to the left of the original image.



Future Work

Unfotuantely this is about as far as I had gotten with this dataset after spending a tremendous amount of time trying (unsuccessfully) to get the stitching to working in the negative space properly.

Items that need to be addressed are:

- Greater feature detection through toggling SIFT detector settings to allow more features in for more roboust feature matching
- Full/robust panorama generation code from images... at points this was actually working, even though it likely shouldn't have been...
- Plotting the 29/168 image data sets.
- Covariance calculation/propagation: right now I'm still using the idenity matrix for covariance. This should be updated to better reeflect the uncertainty.