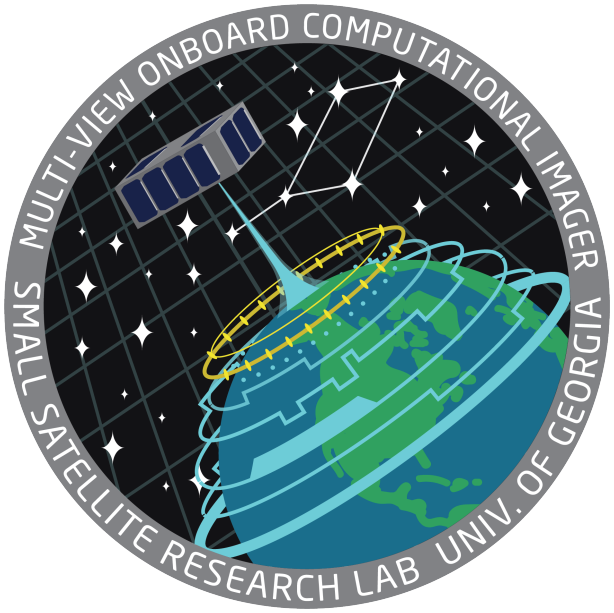


Multiview Onboard Computaional Imager

Algorithm Theoretical Basis Document

Caleb, Adams
CalebAshmoreAdams@gmail.com

LastName2, FirstName2
first2.last2@xxxxx.com

LastName3, FirstName3
first3.last3@xxxxx.com

November 15, 2018

Contents

Chapter 1

Introduction

1.1 File Formats

1.2 Feature Detection

1.3 Feature Discription

1.4 Feature Matching

1.5 2 View Reprojection

1.6 N View Reprojection

1.7 Surface Reconstruction

Chapter 2

Feature Detection

Chapter 3

Feature Extraction

Chapter 4

Feature Matching

4.1 Overview

4.2 2 View Matching

4.3 N View Matching

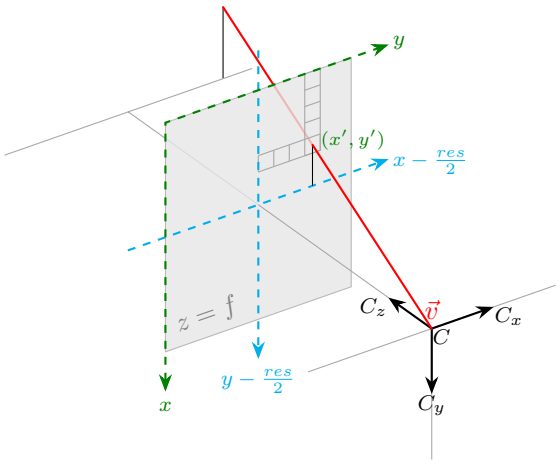
Chapter 5

Two View Reprojection

5.1 Overview

The goal with the 2-view reprojection is to take the pairs of matched points from the SIFT steps and move them from $\mathbb{R}^2$ into $\mathbb{R}^3$ so that equations of lines can be generated from the focal point of the camera into each matched point. Then, we want to find the minimum distance between those lines and choose the midpoint of that line segment at our reprojected point.

5.2 Getting Equations of Lines



The goal here is to generate a parametric equation of a line given camera position and orientation coordinates C, the camera focal length f, and the position of a coordinate in $\mathbb{R}^2$ on the image plane. We wish to generate vector v that can be used to make the parametric equation. The 2-view reprojection takes the matched points between 2 images and places them into $\mathbb{R}^3$. To place each set of points into $\mathbb{R}^3$ some trigonometry and and matrix transformations need to take place. The first step to moving a keypoint into $\mathbb{R}^3$ is to place it onto a plane in $\mathbb{R}^2$. the coordinates (x', y') in $\mathbb{R}^2$ require the size of a pixel $dpix$, the location of the keypoint (x, y), and the resolution of the image $(xres, yres)$ to yield:

$$x' = dpix(x - \frac{xres}{2}) \qquad y' = dpix(y - \frac{yres}{2})$$

This is repeated for the other matching keypoint. The coordinate (x', y', z') in $\mathbb{R}^3$ of the keypoint (x', y') in $\mathbb{R}^2$ is given by three rotation matrices and one translation matrix. First we treat (x', y') in $\mathbb{R}^2$ as a homogenous vector in $\mathbb{R}^3$ to yield $(x', y', 1)$. Given a unit vector representing the camera, in our case the spacecrafts cameras, orientation (r_x, r_y, r_z) we find the angle to rotate in each axis $(\theta_x, \theta_y, \theta_z)$. in our simple case we find the angle in the xy plane with:

$$\theta_z = \cos^{-1} \frac{(\begin{bmatrix}1 & 0 & 0\end{bmatrix} \cdot \begin{bmatrix}r_x & r_y & r_z\end{bmatrix})}{\sqrt{\begin{bmatrix}r_x & r_y & r_z\end{bmatrix} \cdot \begin{bmatrix}r_x & r_y & r_z\end{bmatrix}}}$$

Future software will generate rotations for all planes in an identical way. Now, given a rotation in each plane $(\theta_x, \theta_y, \theta_z)$ we calculate the homogeneous coordinate $(r_x, r_y, r_z, 1)$ in $\mathbb{R}^4$ using linear transformations. The values (T_x, T_y, T_z) represent

a translation in $\mathbb{R}^3$ and use camera position coordinates (C_x, C_y, C_z), the camera unit vectors representing orientation (u_x, u_y, u_z), and focal length f:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) & 0 \\ 0 & \sin(\theta_x) & \cos(\theta_x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} C_x - (x_r + f * u_x) \\ C_y - (y_r + f * u_y) \\ C_z - (z_r + f * u_z) \\ 1 \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ T_z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix} = \begin{bmatrix} x'_r \\ y'_r \\ z'_r \\ 1 \end{bmatrix}$$

The above process should happen for both points that have been matched. This should result in 2 homogeneous points that we will call $\begin{bmatrix} x_0 & y_0 & z_0 & 1 \end{bmatrix}^T$ and $\begin{bmatrix} x_1 & y_1 & z_1 & 1 \end{bmatrix}^T$. Each point has a corresponding camera vector, which is already known thanks to the camera coordinates, $\begin{bmatrix} C_{x0} & C_{y0} & C_{z0} & 1 \end{bmatrix}^T$ and $\begin{bmatrix} C_{x1} & C_{y1} & C_{z1} & 1 \end{bmatrix}^T$. From this we can make parametic lines L_0 and L_1 with the parametic variables t_0 and t_1:

$$L_0 = \begin{bmatrix} x_0 - C_{x0} \\ y_0 - C_{y0} \\ z_0 - C_{z0} \end{bmatrix} \begin{bmatrix} t_0 \\ t_0 \\ t_0 \end{bmatrix} + \begin{bmatrix} C_{x0} \\ C_{y0} \\ C_{z0} \end{bmatrix}$$

$$L_1 = \begin{bmatrix} x_1 - C_{x1} \\ y_1 - C_{y1} \\ z_1 - C_{z1} \end{bmatrix} \begin{bmatrix} t_1 \\ t_1 \\ t_1 \end{bmatrix} + \begin{bmatrix} C_{x1} \\ C_{y1} \\ C_{z1} \end{bmatrix}$$

The Host functions are simple and will not be addressed here, refer to standard CUDA memory management. For the Kernel, we must insure that we have a few global variables about the camera data/parameters availible to the GPU. These variables are the focal length foc, the feild of view fov, and the resolution of the imager res. This assumes that the cameras are in the xy plane, which can only be assumed for this special 2-view case. It may be helpful to read the section on file formats, which is after the main introduction.

In the future, only the first 3 components, the position coordinates, of C_0 and C_1 are used. Vectors v_0 and v_1 are used to parateterizeds lines.

Algorithm 1 Device Kernel

Let: $dpix = foc * \tan \frac{fov/2}{res/2}$
Let: $i = blockIdx.x * blockDim.x + threadIdx.x$
procedure GENERATELINES(R2POINTS, R3CAMERAS)
    $C_0[6] = (R3cameras[0], R3cameras[1], R3cameras[2], R3cameras[3], R3cameras[4], R3cameras[5])$
    $C_1[6] = (R3cameras[6], R3cameras[7], R3cameras[8], R3cameras[9], R3cameras[10], R3cameras[11])$
    $x_0 = (R2points[i] - res/2.0)$
    $y_0 = (-R2points[i+1]) + res/2.0)$
    $x_1 = (R2points[i+2] - res/2.0)$
    $y_1 = (-R2points[i+3]) + res/2.0)$
    $kp_0 = [x_0, y_0, 0.0]$
    $kp_1 = [x_1, y_1, 0.0]$
    $\theta_{x0} = \cos^{-1}(\frac{kp0 \cdot \begin{bmatrix}1 & 0 & 0\end{bmatrix}^T}{\sqrt{\|kp_0\|}})$
    $\theta_{x1} = \cos^{-1}(\frac{kp1 \cdot \begin{bmatrix}1 & 0 & 0\end{bmatrix}^T}{\sqrt{\|kp_1\|}})$
    $kp_0 = kp_0 \begin{bmatrix}1 & 0 & 0 \\ 0 & \cos(\frac{\pi}{2}) & -\sin(\frac{\pi}{2}) \\ 0 & \sin(\frac{\pi}{2}) & \cos(\frac{\pi}{2})\end{bmatrix} \begin{bmatrix}\cos(\theta_{x0}+\frac{\pi}{2}) & -\sin(\theta_{x0}+\frac{\pi}{2}) & 0 \\ \sin(\theta_{x0}+\frac{\pi}{2}) & \cos(\theta_{x0}+\frac{\pi}{2}) & 0 \\ 0 & 0 & 1\end{bmatrix}$
    $kp_1 = kp_1 \begin{bmatrix}1 & 0 & 0 \\ 0 & \cos(\frac{\pi}{2}) & -\sin(\frac{\pi}{2}) \\ 0 & \sin(\frac{\pi}{2}) & \cos(\frac{\pi}{2})\end{bmatrix} \begin{bmatrix}\cos(\theta_{x1}+\frac{\pi}{2}) & -\sin(\theta_{x1}+\frac{\pi}{2}) & 0 \\ \sin(\theta_{x1}+\frac{\pi}{2}) & \cos(\theta_{x1}+\frac{\pi}{2}) & 0 \\ 0 & 0 & 1\end{bmatrix}$
    $kp_0[0] = C_0[0] - (kp_0[0](C_0[3] * foc))$
    $kp_0[1] = C_0[1] - (kp_0[1](C_0[4] * foc))$ ▷ kp0 now represents the keypoint location in $\mathbb{R}^3$
    $kp_1[0] = C_1[0] - (kp_1[0](C_1[3] * foc))$
    $kp_1[1] = C_1[1] - (kp_1[1](C_1[4] * foc))$ ▷ kp1 now represents the keypoint location in $\mathbb{R}^3$
    $v_0 = kp_0 - C_0$ ▷ line L_0's vector
    $v_1 = kp_1 - C_1$ ▷ line L_1's vector

5.3 Minimum Distance Between Skew Lines

Now that we have lines L_0 and L_1, the challenge is to find the points s_0 and s_1 of closest approach. First, we must test the assumption that our lines are skew, meaining they are not parrallel and do not intersect. To start to think of this we take the forms of L_0 and L_1 and simplify them by thinking of them as parametic vectors where C_0 and C_1, represent the camera position vectors and v_0 and v_1 represent the vector was previously calculated from the subtraction of match coordinates with the camera vector. We make the simple equations:

$$L_0 = v_0 t_0 + C_0 \qquad L_1 = v_1 t_1 + C_1$$

To make sure that the lines are not parallel, which is unlikely, we must verify that their cross product is not zero. if $v_0 \times v_1 = 0$ then we have a degenerate case with infinitely many solutions. As long as we know this is not the case we can proceed. We know that the cross product of the two vectors $c = v_0 \times v_1$ is perpendicular to the lines L_0 and L_1. We know that the plane P, formed by the translation of $L1$ along c, contains C_1. We also know that the point C_1 is perpendicular to the vector $n_0 = v_1 \times (v_0 \times v_1)$. Thus, the intersection of L_0 with P is also the point, s_0, that is nearest to L_1, given by the equation:

$$s_0 = C_0 + \frac{(C_1 - C_0) \cdot n_0}{v_0 \cdot n_0} \cdot v_0$$

This also holds for the second line L_1, the point s_1, and vector $n_1 = v_0 \times (v_1 \times v_0)$. with the equation:

$$s_1 = C_1 + \frac{(C_0 - C_1) \cdot n_1}{v_1 \cdot n_1} \cdot v_1$$

Now, given to points that represent the closest points of approach, we simply find the midpoint m:

$$m = \begin{bmatrix}(s_0[x] + s_1[x])/2 \\ (s_0[y] + s_1[y])/2 \\ (s_0[z] + s_1[z])/2\end{bmatrix}$$

5.4 Least Square Approximation for Two Lines

Chapter 6

N/Multiview Reprojection

Chapter 7

Pipelines

Chapter 8

Sources