**GUI documentation**

The GUI is designed to make it easier for a user to process data, without needing knowledge of the underlying python code. This documentation links the GUI to the underlying code, so that the user can edit the backend if required. This can be used in conjunction with the comments in the code itself, which I have aimed to make as self-explanatory as possible.

**Buttons and associated functions:**

**Generate XYZ** invokes the create\_xyz\_basic() method of DataProcessor. This performs a basic generation of X and Y coordinates for the currently loaded array. It simply starts at 0 in the bottom left corner of the array, and as it moves along columns it increments the X coordinate, and as it moves up the rows it increments the Y coordinate. The array is then therefore a N by M by K matrix where K the K dimension provides access to each coordinate (X,Y,Z) and the temperature; angle is also in there too. So the array is a stack of 2D matrices essentially. For .LAS format, this array needs to be flattened, and this is done automatically at the end of create\_xyz\_basic() by the flatten\_array() method of DataProcessor. This turns each variable stored in the K dimension into a 1D array, using np.ravel(). The result is therefore a K by (M times N) array, such that each column of the array has a X, Y, Z coordinate, a temperature, and an angle. The resulting array is plotted in a 3D plot and displayed in the GUI. **IMPORANT NOTE**: This flattened array is the final format which would be required to produce a .LAS file, however, more complexities are introduced when needing to consider variable speed of the car, and changing inclination/orientation of the instrument. This will require much further though, as elements in a temperature matrix cannot simply be assigned a temp/distance value.

**Thoughts:**

Thought for dealing with rotation etc – have a large array held in the buffer at all times during processing – maybe 2 or 3 times the size of each loaded. As the data swings around due to pitch/roll/yaw of car you can geometrically map the data points back to old data points, and forward to new ones in the empty array. If we already have a data point associated with a certain place in the array we can omit the data, if we don’t then we can start to populate the empty array. The result may be a messy sweeping but probably will end up looking ok.

I would have to set the resolution of the matrix to the slowest speed encountered, or to a resolution I am most happy with? Then populate that array with the array of saved data, iteratively. Can calculate the resolution by

Easiest solution to reconstruction may initially be to assume that the lidar and thermal scanner are on the same optical axis (not separated by 11 cm. In the grand scheme of things this small error may not be so significant. This will greatly simplify the reconstruction, as lidar points can be directly mapped to thermal camera points – working out exactly how to shift the lidar to match the thermal, with pitch/roll/yaw and speed changes during acquisition may be very difficult.

Or… Create a 3D matrix, with enough points for the resolution I’m interested in. I can then map points to this matrix using my position relative to the matrix origin and start to populate the matrix. When may data starts to move out of the scope of this matrix I build another, it might have a different orientation, so I would need to reset this and give a descriptor to its orientation relative to the last matrix. OR, once I have populated the the first matrix as much as possible, I can work out the GPS coordinates and altitude of each point, and therefore flatten it down into LAS pointcloud data.