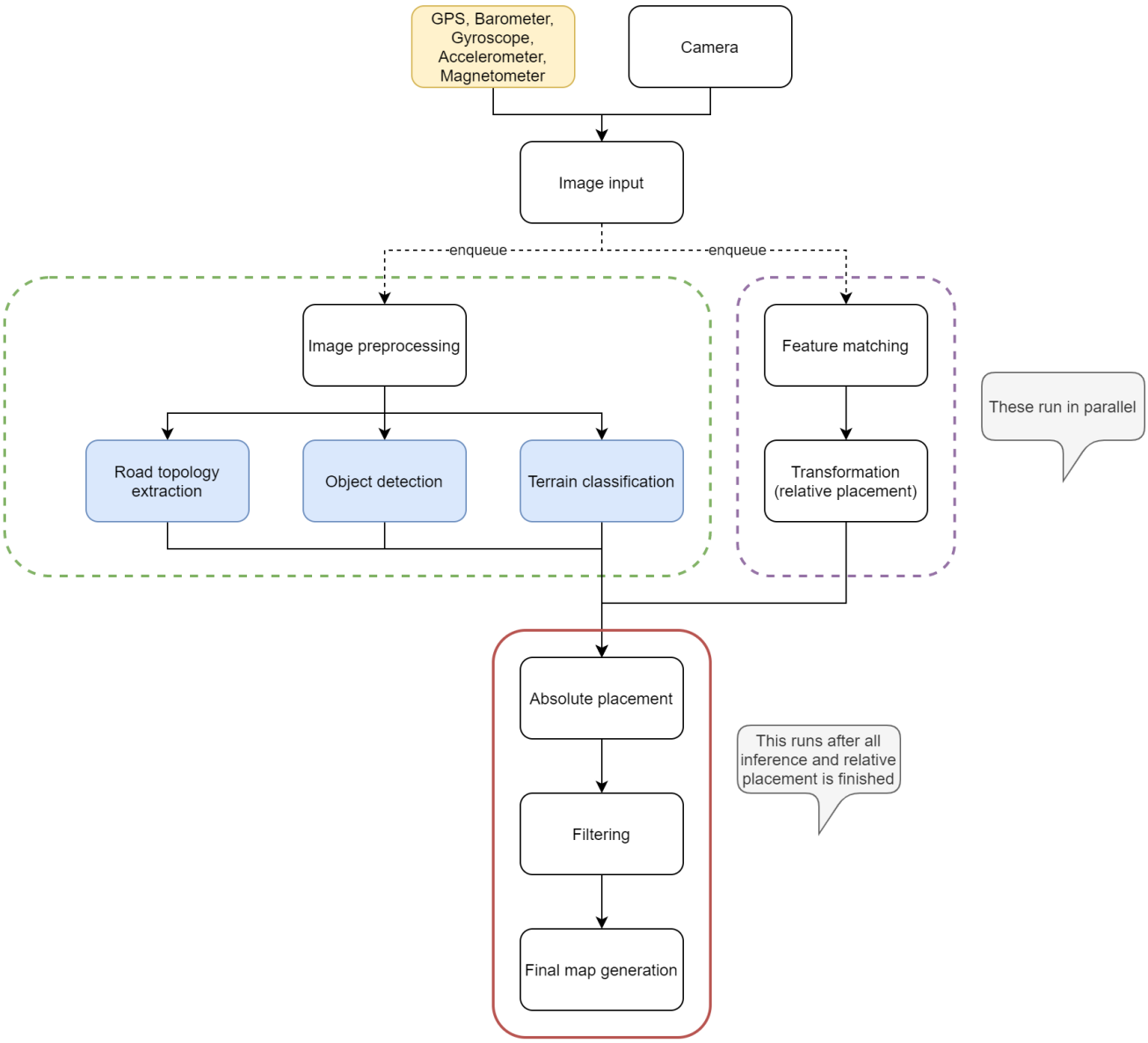

Figure 7: Secondary mission flowchart
(block color indicates the device used: white – Raspberry, yellow – Arduino, blue – NCS)

2.4.2.1 Core

The main event loop responsible for communication between other modules (indicated by arrows in figure 7). Uses `asyncio` and `multiprocessing` for handling asynchronous tasks. Dropping the autonomous flight objective allowed us to take, stitch, and infer from photos completely independently.

2.4.2.2 Image input

For camera input we are using the `webrtc` module (as a wrapper for the *OpenCV's* camera interface). For performance reasons photos are taken if and only if the CanSat moves enough horizontally. Camera distortion is corrected using *Open CV's* `cv2` module as outlined in its documentation [1]. Each image is associated with its metadata, i.e. the GPS, barometer, gyroscope, accelerometer, magnetometer measurements and the timestamp. These are later used in placing the image and the AI's predictions on the map.

This module is also responsible for selecting images suitable for inference – too panned or distorted ones would cause unexpected inference results and possibly break the image stitcher.

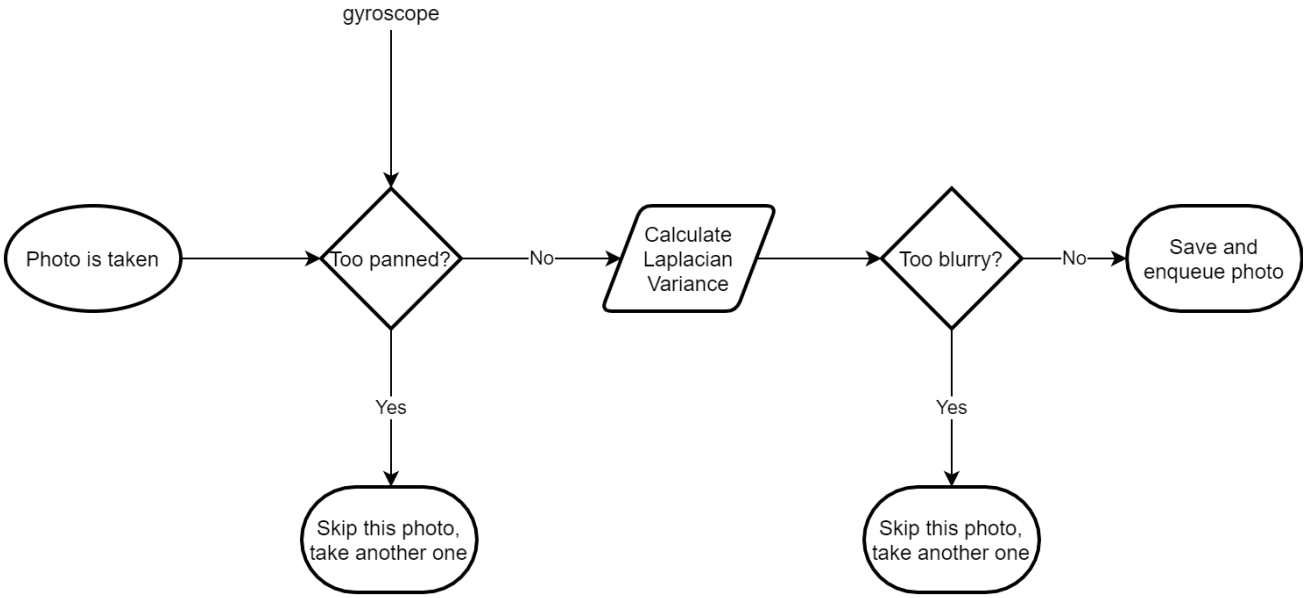

Figure 8: Deciding whether a photo is suitable for inference

2.4.2.3 Image stitcher

After additional consideration to our approach we decided to increase the accuracy of position measurements by stitching the images together using existing popular methods. The photos taken by the CanSat's camera are in a specific order and have some position and altitude hints as previously assumed, but they are imperfect and using only this information to coordinate the relative position proves to be insufficient. However, stitching the photos allows for a much more credible guess.

The basis of our computer vision code is the OpenCV module and the codebase is mainly written in Python to allow quick development and easier integration with the machine learning models, which are also mostly developed in this language. In fact, the `cv2` module provides an out-of-the-box stitching method (`cv2.Stitcher`), but our tests proved it to be unusable for our usage – for more than a couple images the time spent on stitching was extremely long and this is unacceptable for usage on a microcomputer. What is more, the expected number of usable (especially non-blurry) photos taken is on the order of a hundred rather than ten, and the built-in algorithm is not designed to allow for such an use (e.g. the memory usage sometimes becomes too large, especially for our available hardware). What is more, if bad photos are inputted into the algorithm, it becomes extremely unstable and the output quality worsens significantly. This behaviour is somewhat erratic and seems to depend on the algorithm input.

Figure 9: Example stitching process output for three of our aerial images.

Figure 10: Chaotic results of `cv2.Stitcher`, probably caused by lower-quality frames.

Another argument that speaks for using a custom approach is the fact that our data has specific properties: the position has a hint (that allows for checking if the results are faulty or serves as an initial guess) and the ordering of the photos allows us to assume that consecutive images have a significant common area. Additionally we get more control over the stitching process, e.g. we can process the images in steps. This caused us to investigate relevant algorithms in the area of computer vision.

The standard approach for image stitching is composed of three major steps: extracting keypoints in the image via feature extraction algorithms, matching the keypoints via their descriptors with nearest-neighbour approaches, and lastly computing a transform that relatively moves the matched keypoints between first image and the second. This is enough for us to recover data regarding the positions of the images. As an additional step we can then blend the photos together using one of the possible methods.

For the first step (feature extraction) we use the novel ORB algorithm (see [2]) for finding keypoints as well as the standard SIFT algorithm for computing descriptors [3]. The approach was also briefly tested for ORB & SURF. ORB was chosen because of its efficiency and SIFT for the quality of output. Both of the mentioned algorithms are implemented in the OpenCV library. The number of extracted keypoints is important – too little means the homography might have too little inliers, while too many means that too much noise and unimportant points will be returned, and additionally the time of computation rises. The returned descriptors concisely represent the direct vicinity of any keypoint, allowing us to match it with other ones.

The second step is straightforward thanks to the nearest neighbours problem being widely researched. The keypoints from both sets are matched according to descriptors. In case of SIFT they are 128-dimensional real vectors. We use the FLANN [4] heuristic algorithm (implemented in OpenCV as well) which yields good results in practice. Additionally, we use a common heuristic called *Lowe's ratio test*. In essence, for each keypoint two of it's nearest neighbours are computed. Let us denote their descriptor-distances to the point by d_1 and d_2 ($d_1 < d_2$). The ratio test is based on checking whether $d_1 < Ld_2$, where L is the coefficient commonly set to to a value in the interval $[0.6, 0.8]$. This allows us to partially filter "false positive" or noisy matches.

Figure 11: Example image with marked keypoints. Note that areas with no significant contrast contain no or few keypoints.

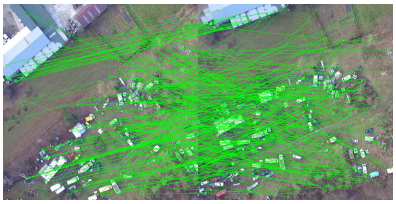

Figure 12: Example image (same as above) with lines connecting matches with another image next to it. Note the occasional emergent behaviour and some false positives in non-contrasting areas.

For the third step the classic approach uses the computed matches and treats each pair of two points (a_i, b_i) as viewed from different perspectives from two cameras in 3D space. This can be formally described as assuming that there exists a homography matrix $\mathcal{H}$ such that $b_i = \eta(\mathcal{H}a_i)$ (treating a_i, b_i as augmented vectors of the form $[x, y, 1]$), where η is the normalisation function $\eta([x, y, k]) = [x/k, y/k, 1]$. A model for minimizing the loss in this parameter optimisation problem is constructed by a OpenCV-supplied function `cv2.findHomography`. In this case, the RANSAC method (as described in [5]) helps to account for *outliers* (which we occasionally called false positives intuitively), meaning matches that fail to conform to any good homography that works for other points (the *inliers*).

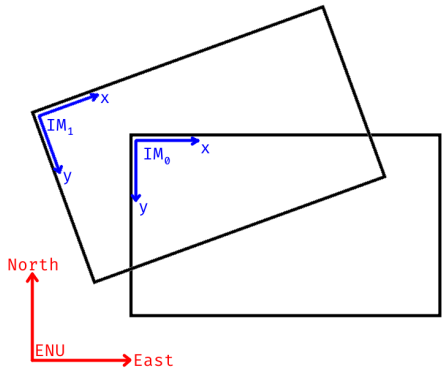

Figure 13: The IM and ENU coordinate systems on a diagram. It should be noted that their orientation is different.

We revised this method after testing to better suit our needs. First we will introduce additional notation to formally describe our knowledge about the system. So far, the first and second steps allowed us to get additional information about the relative positioning of given (key)points on following images. Let us denote the coordinate system of the i-th image as IM_i. The computations performed in the third step give us information about the transformation $\mathrm{IM}_{i+1} \to \mathrm{IM}_i$, as in the placement of pixels on the current photo relative to the previous one. However, we also have access to information from an absolute system we call ENU (East North Up), which is calculated relative to a point on the surface of the planet. Information in this system is calculated from GPS measurements by transforming the geodetic coordinates to ECEF and then finally to ENU. A key property of ENU is that it is a system of coordinates on a plane, similarly to IM. In essence, this plane is the so-called local tangent plane,

which is tangent to the surface of the Earth at a given point.

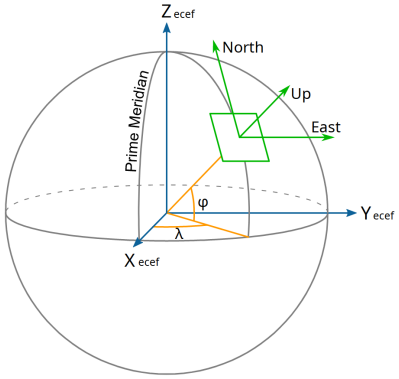

Figure 14: A local tangent plane along with our used coordinates. (Source: Wikimedia Commons)

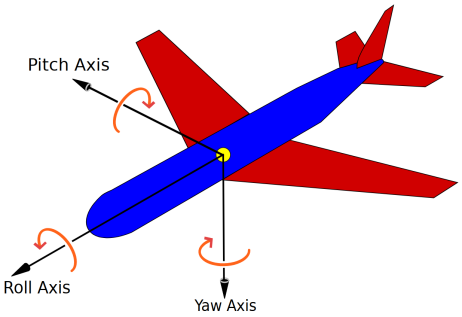

Figure 15: Roll-pitch-yaw rotation axes, for reference. (Source: Wikimedia Commons)

Our available measurements are: GPS and barometer data (directly translates to ENU), gyroscope data (roll & pitch rotations), and compass/magnetometer data (yaw relative to $[0, 1]$ vector in ENU). However, GPS data is only given for a single point, although it turns out that thanks to angle and camera view angle data we have sufficient data to approximate the $\mathrm{IM}_i \to \mathrm{ENU}$ transform. Hence we can compute $\mathrm{IM}_{i+1} \to \mathrm{ENU} \to \mathrm{IM}_i$, where the last transform is an inverse of a previously known one. It can be shown that these transforms are (approximately, disregarding the varying distance to camera lens) linear and given by rotations, scaling and translation, and we call these kind of transforms *even similarities* for short.

We found that the homography-based method used in a following-photo approach induces a large amount of overlaying errors, especially ones deforming the image (that assume that the camera underwent roll and pitch rotations). As such, we assume the relative roll and pitch rotations to near equal to zero during the stitching process during transformation fitting. We do not ignore them altogether: when needed, they are handled properly according to gyroscope outputs and transformed before being inputted as the next photo to be stitched. Formalising our requirements, we need to handle only translation, scaling, as well as planar rotation – we encounter even similarities yet again.

It turns out functions of the form $y = mx + c$ for $m, c \in \mathbb{C}$ (where $\mathbb{C}$ is the set of complex numbers) are bijective to even similarities (in short, the c term is bijective to translation, $|m|$ to scaling, and $\arg m$ to rotation). Since we have a sequence of vectors $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ in the systems IM_{i+1} and IM_i, fitting $\mathrm{IM}_{i+1} \to \mathrm{IM}_i$ is a matter of solving a classic linear regression over the complex numbers – we use the least squares method implemented in `numpy.lstsq` for maximum efficiency. To account for outliers we use RANSAC as in the original method. Our method performs faster and is more stable, since the minimum sample size is 2 instead of 4, and less degrees of freedom allow for more consistent results. Additionally, as shown before we can approximately compute $\mathrm{IM}_{i+1} \to \mathrm{IM}_i$ and hence can use this as an initial fitted "consensus" in the method, where instead of random samples we can apply a random error to this transformation. Thanks to this the method will never deviate too far from a ground-zero truth, as the errors from relative computations are cancelled by the ones coming from absolute measurements.

2.4.2.4 Stitching enhancement

Finally, various methods for image blending and output composition can be used. This is be important only in the final visualisation phase. State-of-the-art methods include seam finding-based methods, but since their implementation seems to be scarce we gave up using them. We tested approaches based

on weighted averages and "covering" each pixel with a single image (overwriting). The second of the approaches is lightweight and as fast as possible, and it produces acceptable results as long as the stitching remains precise – otherwise *cracks* may appear. The weighted average approaches seek to remedy the appearance of cracks. Weights give us a smoothing-out or blurring effect, which in some cases gives much better results. In our main tested weighted average approach (occurence-weights) a weight of 1 is given if and only if the image is opaque at a given point, and 0 otherwise. However, the occurence-based weighted average often performs badly, producing mainly blurry images or "ghosts" (a double-see effect). This comes in contrast to the overwrite method where areas of it look very good but the cracks between different images sometimes produce a undesirable contrast. In an attempt to fix this effect using other averages was considered, but the issue turned out to be too complex and we decided to keep using covering.

Figure 16: Example stitched images.

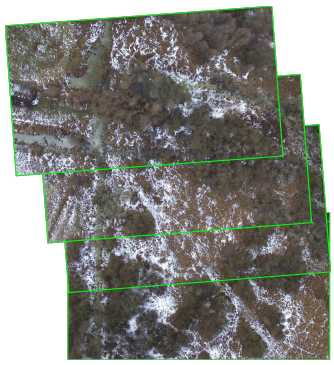

Figure 17: Same stitching output, but with drawn contours.

To increase the output quality of the covering method we came up with techniques for picking the used photos and the order in which they should be drawn. In essence, we want to draw images that cover the most previously uncovered area first, and only keep images that cover enough new ground. In order to simplify the necessary computational geometry, we intuitively draw a fine square grid on top of the output image and compute the squares covered entirely by any given image. We use these "tiles" as the concept of area instead of individual pixels. Thanks to not handling each pixel separately this proves to be quite fast and provides good results when there are many photos on the same area. We omit the details of used computational geometry algorithms, but they are entirely within the scope of classical research (problems such as segment intersection or checking for points in a polygon). Calculations are performed over the integers so that no precision is lost and performance is optimal.

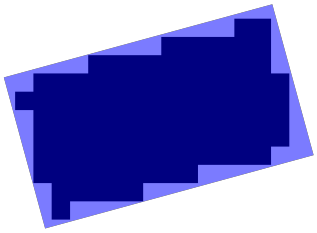

Figure 18: An example "grid ghost" – the blue area is covered by the image, with dark blue containing grid squares.

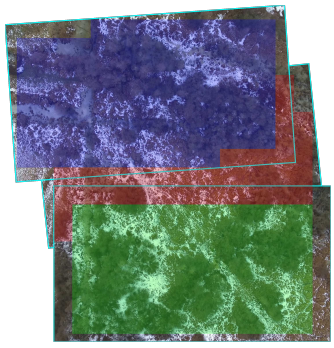

Figure 19: Stitched images with covered grid squares of each drawn, according to a computed ordering.

So far, the output image uses the IM_0 system. However, using an absolute system like ENU for this proves to be useful. We can achieve this quite easily by fitting IM_0 position hints in ENU in an even similarity linear regression, in the same fashion as previously, although a robust method is not needed this time. Afterwards we can use the fact that transforming from geodetic to ECEF and to ENU is invertible and overlay geographic coordinates over the output map (although they are inherently imprecise they serve as a good-enough approximation). It should be noted that using ENU here assumes that the curvature of the Earth is negligible around the photographed area.

In order to improve the quality of input images we used techniques for camera calibration and lens distortion reduction. This is handled entirely by `opencv`. Example output for a fisheye lens camera (which has a large amount of distortion) is shown below.

Figure 20: Example image of a chessboard with a fisheye (wide view angle) camera. Notice lines appear curvy.

Figure 21: Image after undistortion according to previously conducted calibration. Notice the lines of the chessboard are straight. The image is then clipped around the inner area – the external parts are artifacts.

It is worth noting that we experimented with gradient descent-based global approaches as well, used in state-of-the-art stitching algorithms. However, they are slow and could be used only after the entire flight on another, faster, computing machine, and so we do not use them. We also experimented with stitching relatively to multiple previous images instead of a single one, but this did not help increase the quality by much, though severely hurting performance.

2.4.2.5 Stitching prediction data

We plan to use the image processing neural networks on the source images and then use the fact we have control over computing the transformations to stitch it separately using previously designed methods, but for instance using a weighted average blending instead, since the ghosting effect should be less significant in this case. This comes as a direct advantage to exposing the interfaces by writing parts of the code from scratch.

Some neural networks are only designed to works at given pixel-to-meter ratios. Using known camera view angles and altitude measurements we can compute this ratio at any given moment and scale the image appropriately. This does not need to be precise and is a matter of simple measurements.

The input images are tessellated into squares, which are given as input to the neural networks (since their input resolution is much smaller than the original resolution), and the outputs are blended properly using averages or similar methods. Afterwards classical approaches (e.g. nearest-neighbours) for increasing the continuity of the predictions can be used to improve the result quality.

2.4.2.6 Map generation

Figure 22: A stitching result from 20 images (note that larger gaps are harder to overcome algorithmically since the common area is smaller). The greedy ordering is used.

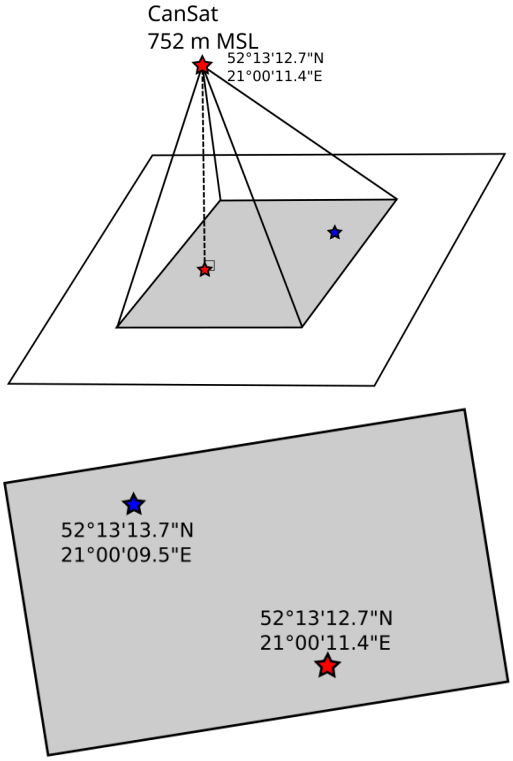

Figure 23: Image projection

We are going to take as many clear-enough images as possible, in order to improve the stitching quality (a large common part yields better results), and stitch them in a streaming model based on the previous images (in order to achieve maximum efficiency). Afterwards, the output map will be saved in a compressed form and software for viewing it will produce a proper rendering – this does not have to happen on the CanSat since the data needs to be collected manually anyway. Additionally, we can fit our ENU and rotation data against produced stitches, checking for the size of the error (failure to fit implies that something went horribly wrong).

The issues arising during generating the final map were mostly already solved as shown previously in this report. For example, placing a geographic reference on the map is solved by the $IM_0 \to ENU \to$ geographic transforms. Similarly, the output can be computed either in one go or our procedures allow to compute it in any pixel (or, in fact, ENU) region. Every image can be represented after the full

stitching process as a pair [offset, transformed image]. The procedure of drawing the map requires choosing a region $[x_{\min}, x_{\max}, y_{\min}, y_{\max}]$ and some simple rectangle math to draw the proper images in the right places. Some pixels are transparent (after rotation, the image does not fit in a simple isothetic rectangle) and to increase the efficiency we just assume black pixels to be transparent instead of introducing an alpha channel.

We note that the approach of using ENU only works for regions where the curvature of the Earth is negligible. For longer flights, this can be remedied by splitting the map into arbitrary regions where the local tangent plane approximation is good enough. However, for our use this is not required.