Core Functions of FastGeoRef

The package's five core functions can be accessed via command-line entry points, which launch dedicated GUIs for user interaction.

**1. Data Generation:**

• **GUI Command:** fastgeoref.run\_datagen()

• **Summary:** This script is used to **extract image data (animal and background patches) from drone videos** to build training datasets for machine learning models. It enables users to manually select and crop regions of interest (ROIs) from video frames, saving them with structured filenames for future model training.

• **Inputs (via GUI):**

    ◦ **Filename:** The name of the video file (e.g., DJI\_0001).

    ◦ **Video path:** The directory containing the video file.

    ◦ **Output directory:** The folder where the cropped images will be saved.

    ◦ **Window resize factor:** A numeric value by which the original video is resized for display on screen (e.g., 2 for a 4K video on an HD screen).

    ◦ **Frame rate:** The frame rate of the video.

    ◦ **Image width (px), Image height (px):** The original dimensions of the video in pixels.

    ◦ **Grabsize (px):** Defines the half-size of the square bounding box (e.g., 15 will result in a 30x30 pixel cropped image in the resized frame). It should be set to ensure the entire animal is covered within the box.

• **Outputs:**

    ◦ **Cropped image patches (.png files):** Saved in the specified output directory. Filenames indicate the video source, timestamp, and image count.

• **User Interaction during Workflow:**

    ◦ **Spacebar:** Pauses/plays the video and enters annotation mode.

    ◦ **Mouse Click:** Extracts an image patch around the clicked object and saves it.

    ◦ **z:** Rewinds the video by 5 seconds.

    ◦ **x:** Fast-forwards the video by 10 seconds.

    ◦ **Esc:** Exits the program.

**2. Model Building:**

• **GUI Command:** fastgeoref.run\_modelbuild()

• **Summary:** This script facilitates the **training of a Convolutional Neural Network (CNN) for object/animal detection**. It handles data preparation, including augmentation to balance dataset classes, preprocessing, defining a four-layer CNN architecture, and training the model. The trained model is then saved for inference in subsequent steps.

• **Inputs (via GUI):**

    ◦ **Training data path:** The directory where training images are saved. This directory **must contain two sub-folders named animal and background** with the respective image patches.

    ◦ **Save model path:** The full path and filename where the trained CNN model will be saved (e.g., cnn\_model\_balanced.h5py).

    ◦ **Image size:** The target dimension (in pixels) to which images will be resized for model training (e.g., 75 for 75x75 pixels). This should be set as **grabsize \* window resize factor \*2**.

    ◦ **Batch size:** The number of samples per gradient update during training (e.g., 64).

    ◦ **Epochs:** The number of times the entire training dataset is passed forward and backward through the neural network (e.g., 25).

• **Outputs:**

    ◦ **A trained CNN model (.h5py file):** Capable of detecting animals (binary classification: yes/no) and saved at the specified path.

**3. Object Detection and Tracking (Automated):**

• **GUI Command:** fastgeoref.run\_auto\_tracking()

• **Summary:** This script automates object detection and tracking in video files using a pre-trained CNN model. It first identifies potential object locations as "blobs" or "keypoints" and then uses the CNN to classify these regions. Detections with high confidence scores are then tracked using a YOLO-based tracker, and the results are saved as an output video and a CSV file.

• **Inputs (via GUI):**

    ◦ **Video File:** The full path to the input video file (.mp4, .avi).

    ◦ **Model File:** The path to the folder containing the pre-trained CNN detection model.

    ◦ **Output Directory:** The directory where the processed video and tracking data will be saved.

    ◦ **Window resize factor:** Value to resize video for display.

    ◦ **Grabsize:** Bounding box size for extracting cropped images for model inference. **Must be the same value used during data generation**.

    ◦ **threshold\_value1, threshold\_value2:** Minimum and maximum pixel intensity values for grayscale thresholding to identify regions of interest. Set to 0 and 255 respectively if unsure.

    ◦ **threshold\_step:** The increment value for thresholding.

    ◦ **minAreaThresh, maxAreaThresh:** Approximate minimum and maximum pixel area (in pixels) occupied by each animal in the video, used to filter keypoints.

◦ **detection\_start\_time**: The time in seconds from where animals detections/tracks are needed

◦ **frame\_rate**: Frame rate in seconds or fps of the video

    ◦ **YOLO Tracker Parameters (internally set):**

        ▪ max\_age: Maximum duration (frames) after which an undetected ID is dropped (e.g., 60).

        ▪ track\_threshold: Minimum average confidence of a track to be considered valid (e.g., 0.6).

        ▪ init\_threshold: Minimum confidence in a new detection required to start a track (e.g., 0.6).

        ▪ init\_nms: Threshold to initialise a track based on its overlap with another (e.g., 0.0).

        ▪ link\_iou: Threshold to link detections based on proportion overlap (e.g., 0.01).

• **Outputs:**

    ◦ **A processed video (\*\_det.avi):** Showing detected objects marked with coloured dots and unique ID numbers.

    ◦ **A CSV file (\*\_tracks\_raw.csv):** Containing detailed tracked object coordinates, confidence scores, and other tracking information.

    ◦ The output can also be visualised in real-time during processing if enabled in the script.

**4. Geo-referencing:**

• **GUI Command:** fastgeoref.run\_georef()

• **Summary:** This script is crucial for converting pixel-level positions of detected objects into their **real-world geographic coordinates (latitude and longitude)**. It takes the raw tracked detections from the automated tracking script (objtrackauto.py) and integrates drone flight log data to ensure accurate geo-referencing, adjusting for potential time lags and height differences.

• **Inputs (via GUI):**

    ◦ **Video Path:** Directory of the video file.

    ◦ **Video Filename:** Name of the video file (without extension).

    ◦ **Tracks Path:** Directory of the raw tracked detections CSV file (\*\_tracks\_raw.csv).

    ◦ **Track Filename:** Name of the raw tracked detections CSV file (without extension).

    ◦ **Logs Path:** Directory of the drone flight log CSV file.

    ◦ **Log Filename:** Name of the flight log CSV file (e.g., flight\_log.csv).

    ◦ **grabsize:** Must be the same value used when generating training data.

    ◦ **time\_off:** The time difference (in seconds) by which the drone flight log lags from the drone video time.

    ◦ **height\_ft\_off:** The height difference (in feet) between the drone's take-off point and the area where animals are being tracked.

    ◦ **log\_freq:** The logging frequency of the flight data (e.g., 5 for 200ms logging frequency).

    ◦ **h\_pix, v\_pix:** Horizontal and vertical pixel resolution of the video (e.g., 3840, 2160).

    ◦ **h\_fov, v\_fov:** Horizontal and vertical field of view of the drone's camera when recording the video (e.g., 70, 40).

    ◦ **get\_flight\_details and pix\_to\_pos functions:** These essential functions are imported from tracking\_functions.py.

• **Outputs:**

    ◦ **Geo-referenced Detections CSV (\*\_tracks\_geo-referenced.csv):** Contains object IDs, pixel positions, timestamps, and their corresponding geographic coordinates (latitude and longitude).

**5. Manual Object Tracking and Geo-referencing:**

• **GUI Command:** fastgeoref-objtrackmanual

• **Summary:** This script offers a manual alternative for tracking objects, useful when automated detection might be challenging or for specific objects of interest. Users can manually click on objects in a video, and the script records their pixel coordinates and timestamps. Crucially, it then proceeds to **geo-reference these manual selections** directly, converting them into real-world coordinates using drone flight log data.

• **Inputs (via GUI):**

    ◦ **Video Path:** Directory of the video file.

    ◦ **Video Filename:** Name of the video file (without extension).

    ◦ **Log Path:** Directory of the drone flight log CSV file.

    ◦ **Log Filename:** Name of the flight log CSV file (e.g., flight\_log.csv).

    ◦ **Tracks Output Folder:** The directory where the raw and geo-referenced tracks will be saved.

    ◦ **Window resize factor:** Value to resize video for display (e.g., 2.5 for better visualization of 4K footage).

    ◦ **Frame rate:** The frame rate of the video (e.g., 30 fps for the reference video).

    ◦ **Image width (px), Image height (px):** The original dimensions of the video in pixels (e.g., 3840, 2160 for 4K).

    ◦ **time\_off:** Time offset (in seconds) between drone flight log and video time.

    ◦ **height\_ft\_off:** Height difference (in feet) between the drone's take-off point and the tracking area.

    ◦ **log\_freq:** The logging frequency of the flight data (e.g., 5).

    ◦ **h\_pix, v\_pix:** Horizontal and vertical pixel resolution of the video.

    ◦ **h\_fov, v\_fov:** Horizontal and vertical field of view of the drone's camera.

    ◦ **get\_flight\_details and pix\_to\_pos functions:** These essential functions are imported from tracking\_functions.py.

• **Outputs:**

    ◦ **Raw Detection File (\*\_tracks\_raw.csv):** A CSV file containing manually clicked pixel coordinates and their corresponding video timestamps.

    ◦ **Geo-referenced File (\*\_tracks\_geo-referenced.csv):** A CSV file containing the real-world geographic coordinates (latitude and longitude) of the manually detected objects.

• **User Interaction during Workflow:**

    ◦ **Spacebar:** Pauses/plays the video and allows manual point selection.

    ◦ **Mouse Click:** Records the pixel coordinates of the clicked object.

    ◦ **z:** Rewinds the video by 5 seconds.

    ◦ **x:** Fast-forwards the video by 5 seconds.

    ◦ **Esc:** Exits the program.

--------------------------------------------------------------------------------

Helper Functions:

This script, though not directly callable as a GUI command, contains crucial helper functions utilised by georef.py and objtrackmanual.py for geo-referencing and tracking.

• **get\_flight\_details(filename, pth, pth\_log, filename\_log):**

    ◦ **Purpose:** Extracts flight data from DJI Mini 3 Pro video subtitle (.srt) files and flight log (.csv) files. It matches the correct flight log to a video file and returns two dataframes: df\_video (timestamps from the SRT file) and df\_flight (flight parameters from the log file).

    ◦ **Usage Notes:** May require adjustments for SRT/log formats from different drone models. It incorporates 'true' flight start time for corrected timestamps.

• **pix\_to\_pos(data, df\_flight, srt\_dt, time\_off, height\_ft\_off, log\_freq, h\_pix, v\_pix, h\_fov, v\_fov):**

    ◦ **Purpose:** Converts pixel positions of objects in a video frame to geo-coordinates. It synchronises flight data with video timestamps using an offset, estimates the object's distance from the drone, and calculates geo-referenced latitude and longitude.

    ◦ **Usage Notes:** Accounts for drone height, gimbal pitch, and camera field of view. It also checks for drone movement (gimbal, yaw, horizontal speed) at each timeframe to identify potential sources of geo-referencing error.

• **yolo Class for Object Tracking:**

    ◦ **Purpose:** Implements a Kalman filter-based object tracking system adapted from Rathore et al., 2023. It tracks moving objects using bounding boxes and estimates object motion over time, supporting non-maximum suppression (NMS) to filter overlapping detections. This class is integral to the automated tracking process in objtrackauto.py.