**ADSB Data Overview**

Overview

ADSB data in the specified region is recorded every 5 seconds, capturing information about known aircraft in a specified area. The data is presented in JSON format, with each aircraft identified by a unique hex number. A Python script extracts various parameters from the ADSB data for analysis.

The following parameters are extracted from the ADSB data:

* **Hex:** ICAO Identifier, a unique identifier for each aircraft.
* **Flight:** Flight name or aircraft registration.
* **Alt\_baro:** Aircraft barometric altitude (feet).
* **Alt\_geom:** Aircraft geometric altitude (feet).
* **Gs (Ground Speed):** Speed of the aircraft over the ground.
* **Track (True Track):** True track over the ground.
* **Baro\_rate:** Rate of change of barometric altitude (feet/minute).
* **Lat:** Latitude.
* **Lon:** Longitude.

For more information about ADSB data fields, refer to [ADSB Exchange](https://www.adsbexchange.com/ads-b-data-field-explanations/).

**Coordinate Systems of AirSim / Unreal Engine**

All AirSim API uses NED coordinate system, i.e., +X is North, +Y is East and +Z is Down. All units are in SI system. This is different from the coordinate system used by Unreal Engine. In Unreal Engine, a cartesian coordinate system is used and the length unit is in centimeters instead of meters. It appear Airsim can handle the appropriate conversion from the NED system to the cartesian system. The starting point of the vehicle is always coordinates (0, 0, 0) in NED system when employing Airsim in Unreal Engine. The vehicle is spawned in Unreal environment where the Player Start component is placed.

**Proposed solution for conversion between coordinates**

There are two common solutions to this problem in python, using the UTM library or the PyProj library to employ The WGS84 ellipsoid. In Unreal Engine There are also two potential solutions, one being the orginGeopoint function and the other the georeferencing system plugin.

**SOLUTION IN UNREAL ENGINE**

**Unreal Engine Plugin**

I believe this may be the best solution from my understanding we can use lat, long, and altitude in unreal engine through this plugin that will make the appropriate conversions to unreal engine coordinates.

<https://docs.unrealengine.com/5.3/en-US/georeferencing-a-level-in-unreal-engine/>

[iwer/GeoReference: Unreal Engine Plugin to provide a spatial reference between geo-coordinates and game-coordinates. For UE5 help see this [Gist](https://gist.github.com/iwer/fc4e34a2a491461d606d826b72664845)](https://github.com/iwer/GeoReference)

A diagram of a diagram

Description automatically generated

(lots of information on this plug in that is easier to understand in the above documentation)

A screenshot of a computer

Description automatically generated

**OriginGeopoint**

There is a setting called OriginGeopoint in settings.json which assigns geographic longitude, longitude and altitude to the Player Start component.

There was not much documentation on this function in unreal engine; thus, this would be something I need to try in unreal engine myself.

Currently, "OriginGeopoint" in settings.json is the Lat/Lon location of the world coordinate system origin of the Unreal level (0,0,0) while it is described in the documentation as the Lat/Lon location of the PlayerStart/Vehicle.

[Settings - AirSim (microsoft.github.io)](https://microsoft.github.io/AirSim/settings/#origingeopoint)

[Fixed OriginGeopoint GPS error by ahmed-elsaharti · Pull Request #3147 · microsoft/AirSim (github.com)](https://github.com/microsoft/AirSim/pull/3147)

[PX4 HIL OriginGeopoint Issue (Airsim 1.2.2) · Issue #2575 · microsoft/AirSim (github.com)](https://github.com/microsoft/AirSim/issues/2575)

[Georeferencing a Level | Unreal Engine 4.27 Documentation](https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/Georeferencing/)

**Potential problems**

There is limited documentation on the orginGeopoint setting so this would need to be experimented with.

I am unsure how the plugin relates to AirSim and I would need to try the plugin myself to fully understand its capabilities.

**SOLUTIONS IN PYTHON**

**WGS84 ellipsoid Conversion**

WGS84 uses mathematical equations to translate geodetic to XYZ coordinates. The coordinate origin of WGS 84 is meant to be located at the Earth's center of mass. In Python, this calcualtion can be done through the pyproj library where you would create a pyproj.Geod object with the WGS84 as a reference. Following this you can use the fwd method to perform the associated calculations for the WGS84 ellipsoid which would be the following equations.

* x = (N + altitude) \* cos(latitude) \* cos(longitude)
* y = (N + altitude) \* cos(latitude) \* sin(longitude)
* z = (N \* (1 - e^2) + altitude) \* sin(latitude)
* Computing the Radius of Curvature N = a / sqrt(1 - e^2 \* sin(latitude)^2)
* Eccentricity (e) of the Ellipsoid e^2 = 1 - (b^2 / a^2)

<https://pyproj4.github.io/pyproj/stable/api/geod.html>

[XYZ Conversion | Tools | National Geodetic Survey (noaa.gov)](https://www.ngs.noaa.gov/TOOLS/XYZ/xyz.shtml)

[coordinates - Converting LLA to XYZ - Stack Overflow](https://stackoverflow.com/questions/18759601/converting-lla-to-xyz)

**UTM**

A common solution is to convert location information in latitude and longitude to Universal Transverse Mercator coordinate system (UTM). This is especially useful for large dense arrays in a small area. The zyntax is in the form utm.from\_latlon(LATITUDE, LONGITUDE) and it returns the Easting, Northing, Zone\_Number, Zone Letter. Then an additional step is taken to convert this to xy coordinates. One problem with the UTM library is the z component (altitude) must be calculated and added separately to the x, y coordinates. This can be done but would require further research to understand how to properly adjust the altitude.

**Potential problems**

I am unsure how the conversion of UTM and WGS84 will relate to the coordinates of unreal engine. They may need further scaling to directly relate to unreal engine.