This will be a centralized system which Drones, Users and Monitors can connect to in order to autonomously fly Drones to a target, avoiding NoFlyZones and in the future other Drones. The current project target is to devise a system which provides the navigation for Drones automatically, but cannot prevent collisions reliably.

Note: Inability to pre-empt collisions actually may be of benefit as it removes the serialization constraint from the pathfinding (in other words, the paths of the Drones no longer affect each other). This means that, due to this independence, processing of flight paths can happen in parallel, VASTLY increasing scalability of the flight planner as well as reducing path finding time itself.

Systems:

There are 4 components to the system: The Server, Drones, Users and Monitors. Each Drone, User or Monitor can connect to the Server in order to obtain data, change preferences and set flights. The Server will automatically find a path for the Drone to follow around NoFlyZones using an A\* Search.

Drone -> Simply a client attached to a Drone for example which receives a set of waypoints to follow when a User (The owner) sets a flight for the Drone. Drones will have to authenticate themselves with the server to prevent hijacking of the route and other problems. This will be done by creating a Drone Credential table in which each DroneID has a corresponding key. In this way Drones can be created by a user, DroneID and Password written to a file, placed onto a storage medium e.g. MicroSD card and placed into a Drone. This password could be unique for each drone in case it is lost. Further features could include overriding of Drone information in order to enable swapping of SD cards between Drones (This is low priority though and will probably not be implemented).

User -> This can be a human or automated system. The User will use calls to the server to login and manage Drones, NoFlyZones and Monitor Permissions. The User cannot control the objects belonging to other Users unless they have permission to, this not only prevents security issues but isolates each process serving each User, allowing to concurrent writes to the database without issues. A User can add a Monitor to watch the position and paths of its Drones. Additionally a User can set their flights to be visible to the public or not, as well as giving specific Monitors permission to view the position and paths of the Drones.

Monitor -> Can be seen as a watcher. A Monitor can view all the public flight data and all non-public flight data which it has been given permission to view. This could be used to create a visualization of the current flight situation and/or provide customers a real time view on when the Drone will arrive at its destination.

Analysis:

An original idea was to modify each created path to avoid others to they never intersect and cannot collide. However any created paths would be dependent on all other paths created, meaning that paths cannot be created in parallel which would drastically reduce throughput of the server. Furthermore searching a database for any possible intersecting lines would take too long and grow in time when a server is used greatly. Therefore paths will not be designed explicitly not to intersect, instead they will be offset by a random amount in each axis, reducing probability of collision. It will be assumed that the hypothetical drones will be equipped with sensors which enable them to detect an object in front of them and be able to avoid it.

The Coordinate system will be using x, y, z coordinates where x, y are measured in degrees (normal GPS coordinates) and z in metres above the ground, measured e.g. by laser range finder. This will prevent the drone colliding with the ground if a map based offset is incorrect or GPS measurement of altitude is inaccurate.

A connection to the server will not be guaranteed for drones in flight, therefore to ensure a constant quality of service, it will not be possible to send commands to the drone during flight. Instead the drone will only send information about its current status to the server, not attempt to search for commands. Additionally it will be necessary for the drone to be able to recover from a failure in connection and re-establish a connection to the server once possible to transfer data. This data will include details such as current coordinates, battery level and other items.

Objectives for AATC NEA

1. A centralised system will allow Users, Monitors and Drones to connect and will provide them with an interface to organise and monitor Drone flights automatically with a shortest (or close to shortest) path found around areas where drones are not desired.
   1. The server software will connect to a database server to manage concurrent access to various tables/records simultaneously. These tables will include Users, Drones, Monitors, Monitor permissions, flights, flight waypoints and no fly zones.
   2. The individual instances will run in separate processes as python cannot disable the GIL in the threading module. This will allow the program to scale better to higher user numbers and prevent interference between individual instances e.g. if one instance is running the search algorithm it will not affect the other instances.
   3. Creating instances by forming connections will take place in separate processes for each type of connection and any smaller systems, so the connection of one type will not interfere with other types. The different connection types will be placed on different ports e.g. 8000, 8001, 8002…
   4. The DB interface on the server software will use the python DB-API so database servers can be switched if necessary.
   5. The server software will only allow access to all functions once the client has logged in. Until the client logs in the only functions available will be login and add new client and exit.
   6. The graph for the search algorithm will be stored in files on the server.
      1. The search algorithm will be able to access details about the nodes such as cost by using an interface from a graph object.
      2. The nodes will be split into multiple files which are identified by their NodeID. Each block will contain an adjustable number of nodes, when a search requires the node the block with that node it loaded by the graph object and returned to the search algorithm. This will reduce amount of memory required by the algorithm.
      3. The search space will be a 3D graph of nodes connected to their neighbours. This could allow calculation of various node details given a few details about the graph.
      4. The graph will be able to map input search coordinates to the next node which is closer in all dimensions to the origin of the graph.
   7. The server will save the flight created and the waypoints in the database so each flight contains start time , distance, predicted end time etc. and the flight waypoints will contain each point the drone has to follow
2. A User will be the equivalent of a human or algorithm which owns drones, request flights, create no fly zones. This user will connect to the server via sockets and request various details about the state of the server.

2.1 A User Connection object will be created to allow changes in the User interface or allowing another program to import the module and control the user.

2.2 The user will have access to various commands including Login, adding user, adding drone, viewing user drones, viewing all public drones, removing drones, adding no fly zones, removing no fly zones, modifying no fly zones, adding permissions for monitors etc.

2.3 The user will be able to grant an expiring permission to a monitor. This permission will allow the monitor to obtain data about the user’s drones and display them.

2.4 The user will able to set if their drones and flights and flight waypoints are visible to other users. This means that other clients will be able to view the current position of the drone, any flights queued etc. If not enabled other clients will be able to verify existence of the drone and that the drone belongs to that user, however not allow access to the details about the drones or any flights.

3. Monitors are instances which can obtain information from the server such as no fly zones and the flights and drones and waypoints of any users which have given them permission or have public visible data enabled.

3.1 Monitors will be able to show the positions of waypoints, drones and no fly zones. These could be represented visually with pyjama.

3.1.1 The pygame monitor would be able to connect to the server software, retrieve information based on previously mentioned conditions and display them as objects in a window. These objects would also have text shown next to them on the details about the object.

3.1.2 The monitor viewer would be able to zoom in and out and pan using the keyboard. Objects outside of the currently viewed zone as well as objects which would be rendered with a size of less than 1 pixel would not be rendered. This will increase the performance of the monitor.

3.1.3 The monitor would only represent objects and text up to a maximum size as this would prevent memory overflow due to extremely large bitmaps.

3.1.4 The monitor would render images and texts from a cache object to reduce performance loss due to redrawing by only creating a pointer of the surface.

3.1.5 The colour of the objects will indicate the type of object so different colours for drones, no fly zones, waypoints and flight starting and ending points.

3.2 Monitors will be able to refresh every X seconds and obtaining any refreshing information and displaying it again.

3.3 Monitors will be able to view the current viewing permissions referring to them and when they expire, however they are not able to create or extend them.

3.4 Monitors will have to sign in in order to view any information as otherwise another client could connect in their place and retrieve information which is not meant to be accessed by others.

4. Drones will be able to communicate with the server via a Drone Connection object. This will allow them to sign in, check for flights, update status and mark flights complete.

4.1 Drones will have to sign in in order to verify they are the correct drone. Otherwise another client could connect, imitating the same drone and obtain flight details via this method.

4.2 Drones will consist of multiple threads including a simulation of a real drone as a reference and a logic thread which communicates which the server.

4.2.1 The drone simulation thread will simulate a drone moving by calculating movement vectors based on current target waypoint, speed and current location. A real drone would be able to obtain current coordinates from a GPS module and send commands to whatever component controls the drone in proportion to the difference in distance for each axis to the next waypoint.

4.2.2 The logic thread will send commands via queues to the simulation thread and receives the information about the status of the drone.

4.2.3 It would be possible to add another thread/object to control devices such as LEDs which could be used to indicate the current state of the drone. Additionally a button or switch could be added to disable flight while active e.g. if drone needed servicing.

5. Garbage collection for the database would take the form of deleting expired monitor permissions, completed flights and their respective flight waypoints. It would take place on a separate process.

6. Transferal of the records on the no fly zone table will take place in another process, this will calculate which nodes are affected and calculate the highest cost for each node and save the graph to files. It could use an A/B method of saving depending on saving interval so reading the node blocks will not be affected by the creation new files.

8. There will be a telegram bot which the user can communicate with the server.

8.1 The bot will respond to user commands, returning the relevant query to the user or returning the result of the command.

8.2 The bot should be able to convert the inputs to the correct types which are required by the command.

8.3 The bot should be able to identify when enough inputs have been entered into the program and execute the command.

8.4 The bot should be able to cope with incorrectly set commands and inputs or the cancelation of the command currently being used.

8.5 The bot should be able to ensure that a chat can only perform the commands it is allowed to do. It should allow the user to login normally, non logged on chats should act like non logged in users.

9. Connection would take place via sockets. Because this is not necessarily encrypted and private information will be transferred including passwords. There for an encryption key will be transferred via the Diffie-Hellman method. Alternatively RSA could be used to authenticate the server to ensure that the client is connecting to the correct server. This key will be used to encrypt and decrypt the information on both ends using AES. AES will be chosen as it is a symmetrical cipher and so the server and client will be able to both use the shared secret key created to encrypt and decrypt. Furthermore in modern CPUs AES may be implemented in hardware, accelerating the encryption / decryption process.