How to create a new drone wrapper

Drone wrappers are the connection point between the multi-drone platform and a drone's native API. Hence for the platform to interact with a drone type a wrapper is created to translate messages between the platform and the drone.

A new drone wrapper can be created by following these steps:

1. Copy the wrapper template file under */wrappers/object.cpp*
2. Rename both the file name and the first parameter of the DRONE_WRAPPER(...) line to the new drone's identifier. E.g. if the drone identifier is 'foo' the file will be renamed to '*/wrappers/foo.cpp*' and the first param 'DRONE_WRAPPER(foo, …)'.
3. Any required parameters on drone initialization can be declared by adding parameters to the DRONE_WRAPPER line. E.g. say the drone 'foo' requires an IP address and port number on initialisation, then the DRONE_WRAPPER line will be 'DRONE_WRAPPER(foo, ip_address, port_number)'. When a user declares a drone for a mdp session using the 'add_drone' program, the user will be required to fill in these parameters. The values for these parameters will then be given back to the drone as a vector of strings in the 'on_init(std::vector<std::string>)' function in the order they were provided in the DRONE_WRAPPER line. For drone 'foo' this means that the first element of the vector corresponds to the given 'ip_address' and the second the 'port_number'.
4. Next fill out the virtual functions so that they perform the given task using the drone's native API. Positions given are in absolute terms relative to the world's origin point in meters. Velocities are given in meters per second relative to world coordinates. All durations are given in seconds. Yaw is given in degrees, yaw rate in degrees per second.
5. Additional function details can be found in the doxygen documentation.
6. Once completed the wrapper will automatically be included in the platform's next compilation given it lies in the */wrappers/* folder and is named correctly as described above.

Tips:

- If the drone's native API does not support velocity commands for instance, it may be necessary to approximate this functionality using position commands. (crazyflie does not have high-level velocity functionality so modified position messages were used instead)
- If the new drone type requires all drones of that type to communicate through a central point, consider writing a drone type server as a separate ROS node by which wrappers act as clients to that server node (or find a premade ROS package). This is how the crazyflie wrapper works.
- The on_motion_capture function is called every time the drone platform receives a new message from optitrack. If the new drone does not have any inbuilt communication with optitrack it may be necessary to compare the drones position in this motion capture frame to ensure that the drone is where it is expected to be. If

there are differences then micro adjustments can be sent to the drone to align positions.

- The ‘hover’ command is evaluated internally into a on_set_position command where pos is the drone’s current position.
- If you would like to use ROS functions in the drone wrapper, the rigidbody’s node handle can be retrieved through the ‘get_ros_node_handle()’ function.
- A number of drone variables are available in the drone wrapper such as ‘currentPose’, ‘currentVelocity’, ‘homePosition’, ‘timeOfLastUpdate’, etc.