AIDERS OSDK + ROS Documentation

This system was build for a DJI Matrice300 drone to be run from NVIDIA Jetson Xavier NX with Ubuntu 18.04 on the 4.9.253-tegra Kernel. It provides an interface to communicate, exchange data and control a UAV system. Some key focuses of the project are to be **all-encompassing, fault-tolerant, robust, provide remote access and diagnostics, and have minimal setup time.**

This project's documentation is broken down to the following files:

README.md – Introductory documentation and basic setup
CONTRIBUTING.md – Contributions and acknowledgments

aiders_osdk_documentation.pdf – In-depth explanation of modules and processes

AutostartMultimasterOSDK.sh

The entry point of the project. The script responsible for launching the application modules and setting up the execution environment and variables.
First, it closes previous instances, sets up required variables and sources the users .bashrc file, and checks that devel/ and build/ directories were created by running catkin_make. After that it starts launching all the ROS packages, more on those later. Finally, it starts our own modules. This shell script never closes on its own in order to keep child processes alive.

autostartService.sh

The project provides a systemctl service, since many times a display will not be available. The autostart service can be installed using the **autostartInstaller.sh** inside the *aiders_osdk/autostart/* directory. The README.md provides instructions on how to enable/disable/debug the service, if someone is unfamiliar with systemctl.

When enabled this service will call **autostartService.sh** at start-up. This process will setup a hotspot using the Jetson's Wifi module for debugging access when a display is not available, await for 3G and VPN access and then finally call **AutostartMultimasterOSDK.sh** which will launch the project.
This requires a separate 3G dongle to be plugged in the Onboard-Computer, as the service specifically looks for a 3G network connection before calling **AutostartMultimasterOSDK.sh.** In addition the user is responsible for providing and placing a VPN file in *aiders_osdk/configs/*, this file should adhere to the following naming convention;

matrice300_$machine_id.ovpn

getActiveIP.py

Provides an easy and consistent interface for any module or script in this project to obtain the device's IP address. If many connections/IP are available, the script will follow this priority (from high to low):

- Vpn
- 3g
- Wifi
- *Ethernet*

ROS

In this section, the ROS packages and the features each provides will be examined. The project's dependence on each individual package will also be explained.

Roscore
collection of nodes and programs that are pre-requisites of a ROS-based system. You must have a roscore running in order for ROS nodes to communicate

roscore

rosbridge_suite
provides a JSON API to ROS functionality for non-ROS programs. Generally required if the application is communicating with an android/mobile application that does not have access to ROS.

roslaunch rosbridge_server rosbridge_websocket.launch

master_discovery_fkie
Discovers the running ROS Masters in the local network. The local network is defined by the ROS_IP and ROS_MASTER_URI variables which are set by AutostartMultimasterOSDK.sh. There are two separate versions with different names for Ubuntu 18.04 (melodic) and Ubuntu 20.04 (noetic). The following bash code provides an example that will work on both versions

```
ros_version=$(rosversion -d)
if [ "$ros_version" = "melodic" ]; then
        rosrun master_discovery_fkie master_discovery
elif [ "$ros_version" = "noetic" ]; then
        rosrun fkie_master_discovery master_discovery
 fi
```

master_sync_fkie
Synchronizes the local ROS master to the remote masters discovered by master_discovery_fkie node. Without this module running, remote topics or services will not be available to the local system. Like **master_discovery_fkie** there are two separate versions for Ubuntu 18.04 and 20.04.

```
if [ "$ros_version" = "melodic" ]; then
        rosrun master_sync_fkie master_sync
elif [ "$ros_version" = "noetic" ]; then
        rosrun fkie_master_sync master_sync
fi
```

connectionHandler
Handles handshaking and finding the AIDERS platform in the ROS network. If this script is disabled, the drone will never appear in the platform interface.
Depends on **master_discovery_fkie** as it provides the ‘/master_discovery/linkstats’ topic. The script uses the IPs published in the specified topic, identifying the platform by the ports open on each IP. Next it publishes a handshake message to the following topic; '/droneIds', awaiting an acknowledgment from the dronename+'/handshake' topic. When that is done the handshake process is considered complete.

metric_publisher
General purpose script for collecting data and publishing them as ROS messages. When started the script will loop at a given rate and will prepare and publish data packets for use by the rest of the system or stored into a database by the **databaseHandler.** Data collection and publishing sometimes needs to be done in separate threads as collection and publishing of certain packets might need more than the alloted time, slowing down the actual publishing rate.

telemetryTranslator
Module responsible for sharing positioning data from the C++ layer to the python3 layer of the system. Additionally it enirches the shared packets with some metadata like uptime, drone state, and angle of gimbal of the camera, etc.. This module uses positioning_system_priority script. Which subscribes to all available sources of Positioning Data, and ranks them realtime based on a priority and a confidence factor. Allowing the drone to switch from one positioning system to another real time seamelessly without operator interferance.

positioning_system_priority
This system is responsible for finding the most trustworthy source of positioning data during the operation of the UAV. Sources of telemetry can be subscribed to and assigned a priority, then any system that needs to acquire positioning data can expect the prioritiser to return coordinates from the source it deems the most valid and correct. The validation of the sources is based on the number of satellites and/or expected publishing frequency. These validation methods can also be turned off for sources of positioning which aren’t expected with a regular frequency or that don’t use number/strength of satellites.

This script publishes 4 topics:
/PSP/Telemetry
    The latest telemetry package.
/PSP/Source
    The source PSP currently uses and deems the most trustworthy
/PSP/Priority
    The priority of the currently used Source
/PSP/Confidence
    A confidence value ranging from 0.0 to 1.0

databaseHandler
A system for logging ROS messages into a database with synchronization capabilities. The handler subscribes to topics defined by the user inside the subscribeTopics() function, directing incoming packages to the savePacketCB() callback function which serializes the ROS messages and converts them to a list of simple values which are then stored into a database via query.
Synchronization is done via data sequence numbers. Packets that have the same origin as the **databaseHandler** are propagated.

The whole system is divided into the following scripts for readability and scalability, that each executes a different function.

databaseHandler
Entry point for the system. Contains the callbacks for saving packets and handling requests for incomplete data, as well as topic subscriptions.
DatabaseUtils
Handles interfacing with the database, contains insert and select queries and ROS message serialization/deserialization
databaseLogging
Handles preparing and saving data into the database
databaseSynchronization
As the name suggests, this script contains many functions related to the synchronization component of the databases. Has functions for propagating data, checking data completion, creating data requests and so on.
DatabasePacketImports
A helper script where ros message imports can be imported from, ensuring all the above scripts import and have access to the same base of ROS messages.

flightStateController
Mission/Task planner able to interrupt and resume tasks, without losing its previous progress.

orthocamera
Script responsible for controlling the camera, capturing images/video

componenents / componentHandler
Script responsible for handling components

crps
relative positioning system

raft
Raft is a consensus algorithm that is designed to be easy to understand. An implementation of the algorithm exists in this projects that can be used to elect a leader and provide Consensus. Consensus involves multiple servers agreeing on values. This can be deciding which drone in a squadron should hold the forward position or which of two similar packets is more “valid”. If there is an issue where the nodes of a distributed system might provide different results or come to different decisions, the raft consensus algorithm can provide a protocol of solving such inconsistencies.

joy/controllers
joystick controllers.
Implementing all the features DJI Smart controller provides with any control scheme system.

rangefinder
Since rangefinder values are not available by default to OSDK applications, communication with an MSDK device allows the system to obtain rangefinder distance, etc. in a roundabout way. This script is responsible with enabling and disabling the rangefinder via the MSDK and publishing the values retrieved to a ROS topic.

DJI Codebase
Implemented features
Explaining DJI’s approach

Future/Planned Work
...