c4d Demo Documentation (v0.3)

The demo consists of a Kria board with PX4 equipped with a camera and a GPS module. The board is running an artichoke detection neural network and an AES encryptor.

### c4d-companion-demo

****

**Bin :** Folder containing the AES binary file and the script to execute the binary and manage the output.

**Docs :** Folder containing these very logs

**Models :** Folder containing the AITEK plant detector CNN model ONNX file.

**Modules :** Folder containing custom library files in python for the c4d-uc5-launcher.py script.

**Output** : Folder containing files with the encrypted output of the project as well as an encryption report.

### Output

On each loop a string of comma separated (no space) values in the order shown below:

| timestamp | IMEI | Latitude-Longitude-Altitude | SpeedX | SpeedY | SpeedZ | NumPlants |
| --- | --- | --- | --- | --- | --- | --- |
| Yyyymmddhhmmss  (14 chars) | XXXXXXXXXXXXXXX  (15 chars) | {sign}XX.XXXXXX{sign}XX.XXXXXX{sign}XX.XXXXXX  (30 chars) | {sign}XX.XXX  (7 chars) | {sign}XX.XXX  (7 chars) | {sign}XX.XXX  (7 chars) | XXX  (3 chars) |

All of the output values are captured at the beginning of each loop except for the NumPlants variable that is captured at the end of the CNN execution. Functions for the format are defined in “aesHelper.py”

## Command Line Arguments

| Argument | type | Description | Default |
| --- | --- | --- | --- |
| --vehicle\_id | string | Vehicle Identifier used for communication. (Output message IMEI) | "0000000000000000” |
| --px4\_device | string | Path of PX4 device | "/dev/ttyACM0" |
| --px4\_mavlink\_mode | integer | Mavlink mode | 4 (AUTO) |
| --px4\_skip\_wait\_position | boolean | Bypass wait for GPS lock loop | False |
| --px4\_use\_fake\_position | boolean | Use fake GPS position (for simulation purpose) | False |
| --input\_video\_path | string | Path to input video. NOTE: that if this argument is set, camera frames are skipped. | None |
| --input\_camera\_device\_id | integer | Input camera ID | 0 |
| --set\_max\_num\_of\_frames | integer | Set max number of frames processed (for simulation purpose) | None |
| --crop\_detector\_model | string | ONNX DL Model for crop detection | "models/c4d-aitek-v1.onnx" |
| --crop\_detector\_thresh | integer | Set ONNX NMS threshold | 0.2 |
| --show\_crop\_detection | boolean | Show on the frame the bboxes | False |
| --output\_video\_path | string | Write video output. NOTE: if --show\_crop\_detection option is active, the video will contain the detected bboxes | None |
| --skip\_encryption | boolean | Set to True for skipping the encryption step. | False |
| --skip\_communication | boolean | Set to True for skipping the communication step. | False |
| --transmitter\_device | string | Set the serial device for outbound communication | "/dev/ttyUSB0" |
| --transmitter\_boudrate | integer | Set the serial device baud rate | 9600 |
| --verbose | boolean | print messages for debugging | False |

| *Boolean options are set true just by –{command} while all other have to be assigned a value like –{command}={value}* |
| --- |

# Main Script

The main script is the “c4d-uc5-launcher.py”. In general the script 1. initializes the onnx runtime for the artichoke detector network model as well as 2. the mavlink communication and then 3. loops infinitely until the camera is closed.

In each instance of the loop, the GPS position is captured and then the network is executed. At the end of the post-processing of the network output, the GPS coordinates and the artichoke count are encrypted and sent to the transmitter for communication. The produced output can also be evaluated in the output folder.

The functionality of the containing code of the c4d-uc5-launcher.py is presented in the sections below.

## Aitek Artichoke Detector NN

### Summary

The network provided by aitek is running on [ONNX runtime](https://onnxruntime.ai) on 4 arm-cortex cores of the board.

Initialization, execution of the network and post processing is handled by python code.

Before the execution of the network the camera or video input is preprocessed with padding and downscaling. The [ONNX runtime](https://onnxruntime.ai) is initialized to utilize the 4 arm-cortex cores.

The raw output is then post-processed. Bounding boxes with confidence lower than the threshold (0.25) are dumped and Non-maximal suppression (NMS) is applied to clean overlapping boxes with threshold of overlap at 0.2.

Preprocessing function “ResizeAndPad()” and post-processing functions “bbox\_intersection\_over\_union() “ and “FilterBoxesNMS()” are defined in the c4d\_uc5\_onnx\_helper.py

### Parameterization

Parameterization of the network is done via the argparse python library with all the options defined in the “c4d\_uc5\_settings.py” file. The default configuration of the deliverable is set for the outdoor application using a camera for artichoke detection. The settings can be configured to use an input video instead of a camera and write the input video to AVI for testing.

For further notes on the parameters either run the script using the –help parameter (`python3 c4d-uc5-launcher.py –help`) or look at the previous table “Command Line Arguments”.

## Mavlink Communication

Code for communication between the board and the connected PX4 via mavlink is drawn from [DroneKit](https://dronekit.io) and defined in “c4d\_uc5\_mavlink\_helper.py ”.

The code receives GPS information from the GPS module of the PX4. For this, the gps module requires a “Good 3D-fix” -communication with multiple satellites to triangulate an accurate gps position-. This only works in outdoor areas.

To bypass the waiting loop for a “Good 3D-fix” execute the masterScript.py with the “--px4\_skip\_wait\_position” argument.

In outdoors application, the board requests an initial home position from the PX4 GPS module that is updated at the start of every loop. The GPS position coordinates of the PX4 are updated dynamically.

## AES Encryptor

### Summary

In the current version of the demo the AES encryptor is called from an executable binary file “aes-hw-accel-wp5-08-rot” written in C. The executable takes an input of 16 bytes per execution. Note that the application will always encrypt 16 bytes, filling the rest of the message with junk from memory should less bytes be inputted.

The executable was produced for testing and outputs a complete report of the execution (input message, execution time, encrypted message, decrypted message etc)

The executable is called at each loop from the “encrypt.sh” script that cleans the long output report, writing only the encrypted message (cipher) in the “encrypted-msg.out” file that is overwritten with the new message on every loop. The full encryption report is saved in “output/aes-hw-accel-wp5-08-rot.out”.

# Notes for the Developer

The project has been tested on a Xilinx Kria 260 board with a Pixhawk Holybro with a Holybro Pixhawk 4 GPS M8N module. The message is sent to a serial converter utilizing the ftdi driver for the transmitter.