c4d Demo Documentation

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.

Folder



**C4d-aitek-v1.onnx :** Onnx model of the network

**encryptDemos.sh :** Script for handling the AES encryption

**Aes\_sw :** SoftwareAES executable

**Samples :** folder containing the sample video of artichoke fields for testing

**Dronekit-python :** The default installation folder of DroneKit for mavlink comms

**Onnx\_mnist :** The default installation of the onnx runtime

Python Files

**masterScript.py** : The main script

**c4d\_sw/c4dSettings.py** : Command line arguements

**c4d\_sw/aesHelper.py** : Output Format Functions

**c4d\_sw/c4dCnnPkg.py** : CNN Post/Pre-processing and testing functions

**c4d\_sw/pymavlinkPkg.py** : PX4 initialization functions

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

**–simulation:** Bypass wait for GPS lock loop for simulation (False)

**--connectPX4:** Path of PX4 device (default /dev/ttyACM0 nar

'/dev/ttyACM0')

**--MAV\_MODE\_AUTO:** Mavlink mode (default 4)

**--model\_dir:** directory of .onnx model

**--model:** onnx model filename (c4d-aitek-v1.onnx)

**--vid\_path:** path to sample video for simulation (/samples/2022-01-28\_15m.mp4)

**--set\_encrypt\_script:** set the path to encryption script (encryptDemos\_16B.sh)

**--boxOnFrame:** print bboxes on frame (False)

**--write\_frames:** write frames to png (False)

**--write\_video:** write frames to avi (False)

**--full\_video:** write whole video (False)

**--nocamera:** enable camera input (False)

**--artichokeCount:** print artichoke count (False)

**--cameraID:** Choose camera (0)

**--thresh:** set bbox confidence threshold (0.2)

**--num\_of\_frames:** set number of frames from input video (100)

**--print\_msg:** print the output message to console (False)

**--verbose:** 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 “masterScript.py”.

In general the script initializes the onnx runtime for the artichoke detector network model as well as the mavlink communication and then 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 written in the cipher\_0.txt file.

The functionality of the containing code of the main script is presented in the sections below.

Aitek Artichoke Detector NN

Summary

The network provided by aitek is running on onnx runtime 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 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 c4dCnnPkg.py

Parameterization

Parameterization of the network is done via the argparse python library with all the option defined in the “c4dSettings.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, write frames to PNG images or video to AVI for testing.

Comments in the code provide details of each parameter’s functionality.

Mavlink Communication

Code for communication between the board and the connected PX4 via mavlink is drawn from DroneKit ([https://dronekit.io](https://dronekit.io/)) and defined in “pymavlinkPkg.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 “--simulation” 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 automatically.

AES Encryptor

Summary

In the current software version of the demo the AES encryptor is run from an executable binary file “aes\_sw” 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. If more bytes are inputted (96 in the current message format) the program will loop over the message producing encrypted blocks of 16 bytes.

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 “encryptDemo.sh” script that cleans the long output report, writing only the encrypted message (cipher) in the “cipher\_16B.txt” file that is overwritten with the new message on every loop.

Testing

For testing the AES software implementation one should either run the executable directly “./aes\_sw {message}” or edit the “encryptDemo.sh”.

Notes for the Developer

* If calls to the onnx produce an error try ”python3 -m venv onnx\_mnist”
* Sending the contents of the cipher text files (cipher\_0.txt) to the rover/workstation should be handled inside the main\_script.py for synchronization.