Autonomous Vehicle Processor

Final Report

Dec1710 - Team VOLDEMORT
Rockwell Collins - Josh Bertram
Advisors - Philip Jones, Joseph Zambreno
Team Leader - Alex Orman
Communications Leader - Sean Jellison
Webmaster - Chris Kelley
Key Concept Holders - Lixing Lin, Evan Lambert
Scribe - Lucas Ince
dec1710@iastate.edu
http://dec1710.sd.ece.iastate.edu/

Revised: 12/05/2017, Ver. 3.0

Contents

1 Introduction

1.1 Project statement

The purpose of this project is to utilize an embedded system in realtime to provide information to flight systems regarding location of objects in a drone camera's view. A neural network is used to identify complex objects and then a series of other, simpler methods to extract additional information such as distance away from the drone.

1.2 Goals

Intended goals for the system

- Detect runways in an image
- Detect obstructions on the runway
- Detect multiple objects in an image and identify them
- Determine distance and position of the object relative to the camera
- Able to be installed and run on an embedded system
- Stream the results to an external source in a consistent format
- A user should be able to provide a video stream and provide an output with no knowledge of the internal system

1.3 Related Products

Similar works and research to help give ideas of what is possible and possible guidance if issues arose:

1) One similar project to ours, created by a university research group in Austria, highlighted the plausibility of this project working. In their project they built a drone to be controlled by a neural network that took in live data and transmitted it back to a ground station. The ground station, in this case a laptop, computed the neural network and then transmitted instructions back to the drone.
2) Matt Havey posts in his blog about how he went about using a raspberry pi and a camera to do on the fly analytics like we are. In his case, he wanted to know if football was on the TV or if was commercial. Many of the method he discusses in the post gave us a good starting point and more importantly validated that this project was possible.
3) Havey continues on in another post where he explains different ways we can utilize machine learning and machine vision, and the associated trade offs. He also talks about utilizing a different method on his Pi experiment in hopes of decreasing processing time, which we are interested in.
4) Pilot AI Labs, Inc. is a startup founded by Stanford students are working on a deep-learning based computer vision platform to solve real world problems on computer-constrained embedded devices. They currently have a drone that is capable of following someone around. This project is similar to ours in the sense that it uses deep-learning, an embedded system, and does some calculations based on what the neural network predicts.

2 Design

The following sections will discuss the final design of the project.

2.1 Requirements

2.1.1 Non-functional

- Embedded system must fit inside the drone
- Must be able to process an image at least once per second from onboard camera
- Ideally, target goal is 30-60 FPS

2.1.2 Functional

- Runable on the embedded system.
- Can detect basic objects.
- Must be able to take in images from a camera
- Be able to locate key features in an image
- Be able to identify known objects or features from 400ft - 600ft

2.2 System specifications

Hardware Used:

- Board - Nvidia Jetson TX1
- Camera - Logitech C270
- Tested On - Windows 10 Laptop using 2.8Ghz Quad Core CPU
- Trained On - Windows 10 using Nvidia GeForce GTX 1080 Ti gpu

The Jetson TX1 was selected for use at the recommendation of our client for future use as they already had a similar board built into their drones. Additionally, the TX1 offers an impressive amount of compute power for its form factor and cost. After considering these factors it was decided to move forward with originally the TK1, a sister board, and eventually move over to the more powerful TX1. The camera used here is used only for testing and proof of concept. The recommended features for the camera are a minimum of 1Mpx resolution, global shutter to prevent the jello effect, and color images for best results.

Software Used:

- Python 2.7
- Tensorflow - Inception Network
- Cuda 8.0
- openCV 3.1.0
- Tkinter
- Pillow (PIL)

The system runs on Python 2.7 to allow for fast development and better compatibility. Tensorflow is used to implement the neural network. The Inception network available from Google gives a pre-trained network that is trained on simple and common shapes to provide a more accurate analysis. The last layer is trained on runways. It uses openCV 3.1.0 to interact with the camera and perform image modification. Tkinter and PIL are Python libraries used only in the optional component of the system to provide simple

graphical capabilities.

2.3 Final Design

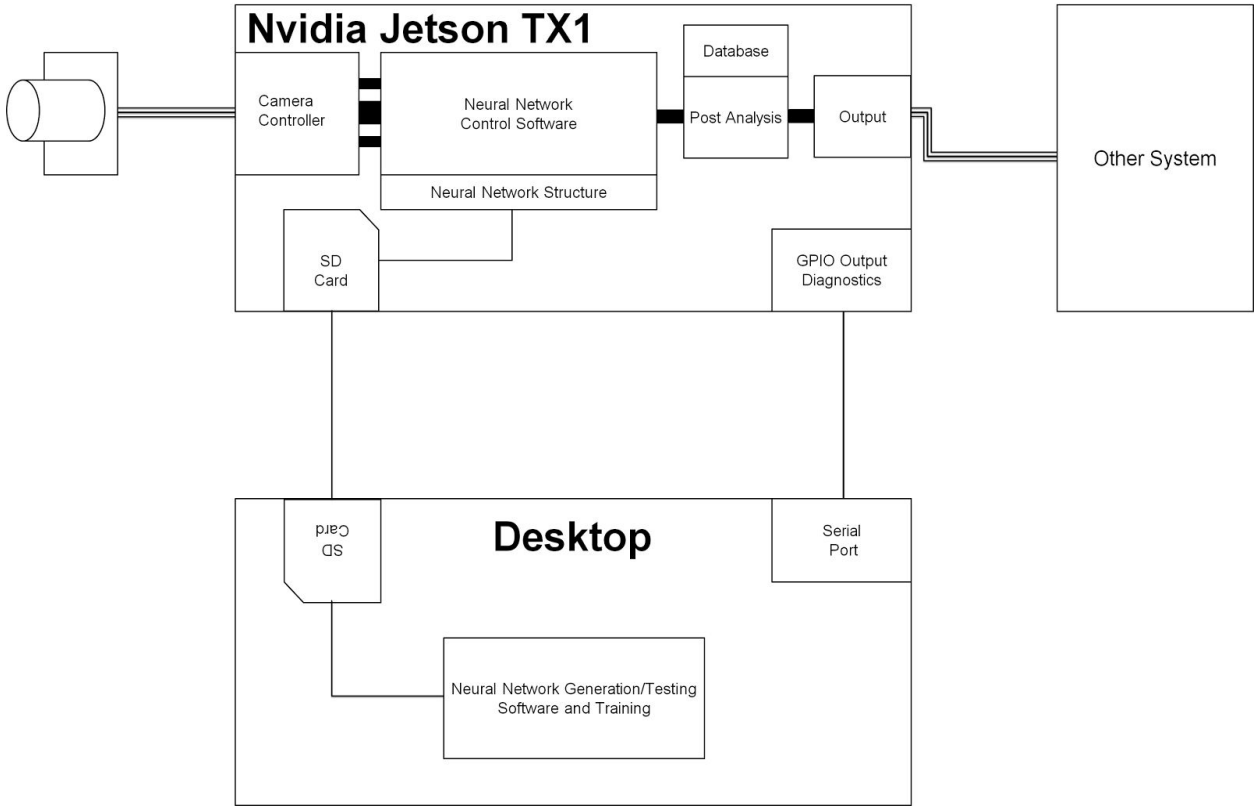


Figure 1 : System Diagram

The final design utilizes module based architecture to better separate the responsibilities of each component. Each module has a well defined role as described below:

1) **Camera Controller Module** - The Camera Controller interacts with the attached camera to pull data from the camera and modify the data as needed. The module runs in its own thread to allow the camera to constantly be taking in new data without being restricted by the other modules. An image is stored in memory and is replaced as quickly as the camera module can update it.

2) **Neural Network Module** - The Neural Network module implements the neural network and all requirements for it. This module runs in its own thread as well and will block until a new image is available from the camera module. This module is expected to run slower than the camera module and therefore will not be able to analyze every image the camera pulls. By keeping this and the camera module in their own threads, the image processing rate is determined only by the slower module and not a combination of all parts.

3) **Post Analysis Module** - This module is used to add additional analysis of the image beyond what the neural network is trained for. The module uses traditional image processing techniques and the results of the neural network plus a database that stores information related to objects it knows about to provide additional data about the image. An example would be identifying a truck on the road and using its actual size to determine how far from the camera if from the truck. This module will also format the data into an organized format for readability and makes it available to other modules.

4) **Output Module** - The Output module is responsible for passing along the data to other sources. This includes reformatting the data from the Post Analysis module as needed (such as serializing the data) to be able to communicate it over a desired medium. It is also responsible for establishing a connection to any external source or defining how and where the data will be stored in the long term as needed.

During runtime all modules utilize the same address space. Images are passed from the Camera Controller module by storing them as a JPG file to the board and the Neural Network reads the image to perform analysis. A thread lock is used to prevent the modules from trying to read and write simultaneously. When the network finishes analyzing an image, it stores the result as a Python list. The main thread locks the list and feeds the data to the Post Analysis module. The Post Analysis then passes its results to the Output module to finish formatting and pass it along.

Because of the use of threading and shared components, deadlock is a possibility. One lock is used to prevent reading and writing the stored image simultaneously, and one is used to prevent the results from the network from being written and read at the same time. The Camera Controller module does not block while trying to save an image, meaning that it is constantly grabbing new images without saving them all. The Neural Network does block while waiting to be able to read the image. This causes the network to analyze the latest image pulled by the camera whenever it finishes its current analysis. The Post Analysis and Output modules run in the main thread in sequence and blocks until the network provides new data. Whenever the Post Analysis finishes processing, it passes its result to the Output.

An additional module is included as well, the Debug module. This module is entirely optional and is used to provide a visual representation of the results for debugging and creating a demo.

2.4 DESIGN ANALYSIS

The collection of data at the moment, roughly 10,000 images, are used to train our network with. Most of these images were acquired from the X Plane 11 simulator through an autonomous process. Since these images of runways are using game models and not actual images of objects, it was decided to mix the dataset with real images of runways. These images were acquired from ImageNet and Google.

For the training process, an Nvidia 1080 Ti was used to train faster than using a cpu. Thousands of iterations through the dataset was used to increase the accuracy of the classification. To improve training results, multiple iterations through each training image narrowed down testing images that were misclassified and were then placed in the appropriate dataset.

Methods used to generate training images:

- Use a simulator and take screenshots of relevant objects
- Find online images and databases such as the famous ImageNet

3 Testing

3.1 The Process

Testing for this project was done in two ways: using a pre-made video, and using a live-stream camera.

Using a pre-made video gives consistency and control over the input. The debug module is used to provide a graphical representation of the analyzed image and results from both the neural network and the post analysis. The first concern is whether every component is functioning. In order to get the results to display, an image must have been pulled by the camera module, a result must have been gained by the neural network, and the post analysis must have finished organizing its results. The output module is tested by getting a result to a source that matches what the post analysis communicated. The procedure is run on both the TX1 and on a desktop system to compare the results and runtime of each system.

Figure <x> shows an example test result.

The Jetson TX1 analyzes 1 image in approximately 0.3 seconds while the laptop using its cpu takes approximately 1 second. The GeForce 1080 Ti took 0.02 seconds. The desktop graphics gpu gets the best performance as expected, but the TX1 being faster than the laptop is an interesting result.

Testing the system using a camera does not provide consistent comparable results, but does give an idea of application performance. Image processing takes longer in all cases, but the time the neural network takes to process an image is nearly identical.

Accuracy for all tests were comparable and ranged wildly. Similar images (1 or 2 frames apart) could have 0.65 to 0.71 confidence. This implies that the problem is not with the hardware.

4 Results

4.1 Expected results

- 85%+ confidence
- 1 frame per second processed

4.2 Final results

Based on testing results the rate of images processed are well within the functional requirements. The accuracy of the identified images are not in desired range. Desired accuracy would be within 95% and the goal of 85%, instead of 50%.

To improve the accuracy, more data and better data is necessary. There is limited accuracy using simulated images which may be causing problems. Alternative data collection include parsing videos into images, using video streams instead of individual images, or getting more real world images. The projects mentioned above have obtained similar but more accurate and consistent data. To improve the results in the future a different neural network, such as YOLO or building a custom neural network, could have produced more accurate results.

5 Appendices

Appendix I: Operation Manual

Linux Ubuntu 16.04 Setup Instructions

1) If your system is using a graphics card make sure that the Nvidia graphics driver is up to date
2) Install Python 2(CentOS, Redhat Enterprise, and Ubuntu come with a version of python out of the box)
    a) To check the version of python use this command:
        python2 --version
    b) If none is specified follow this process:
        (Process to follow)
3) Install Nvidia CUDA 8.0 Toolkit(Required only if using a gpu)
    a) Download the appropriate installer from the Nvidia Website
    b) Change directory to where the installer was downloaded
    c) $ chmod +x cuda8_package
    d) $ ./cuda8_package --extract="where you want to extract the components"
    e) Change directory to where the installer components are extracted
    f) $ sudo ./cuda-linux-package
    g) Follow installer
    h) $ sudo ./cuda-samples-package
    i) Follow installer
    j) Both should be installed correctly
4) Install Nvidia cuDNN(Required only if using a gpu)
    a) Register an nvidia developer account and download cuDNN from the offical Nvidia Website
    b) Determine where CUDA is installed using the following commands:
        $ which nvcc
        $ ldconfig -p | grep cuda
    c) Extract contents of cuDNN download
    d) Copy files to the appropriate folders(copy these to where CUDA is installed found from the above step):
        $ cd folder/with/extracted/contents
        $ sudo cp -P include/cudnn.h /usr/include
        $ sudo cp -P lib64/libcudnn* /usr/lib/x86_64-linux-gnu/
        $ sudo chmod a+r /usr/lib/x86_64-linux-gnu/libcudnn*
5) Install openCV (and all its requirements, version 3.1.0.)
        Please follow the instructions at
        https://docs.opencv.org/trunk/d6/d15/tutorial_building_tegra_cuda.html
6) Install Tensorflow(intended use with the gpu but cpu version should also work)
        Please follow the pip installation for ubuntu:
        https://www.tensorflow.org/install/install_linux#determine_how_to_install_tensorflow

7) Once pip is installed from step 6, install the PIL and Tkinter libraries and prerequisites:
   $ sudo apt-get install python-dev libjpeg-dev libfreetype6-dev zlib1g-dev'
   $ sudo pip install PIL
8) Download a package of our source code
9) Extract the contents to desired location

L4T(Linux for Tegra) Setup Instructions on a Jetson TX1:

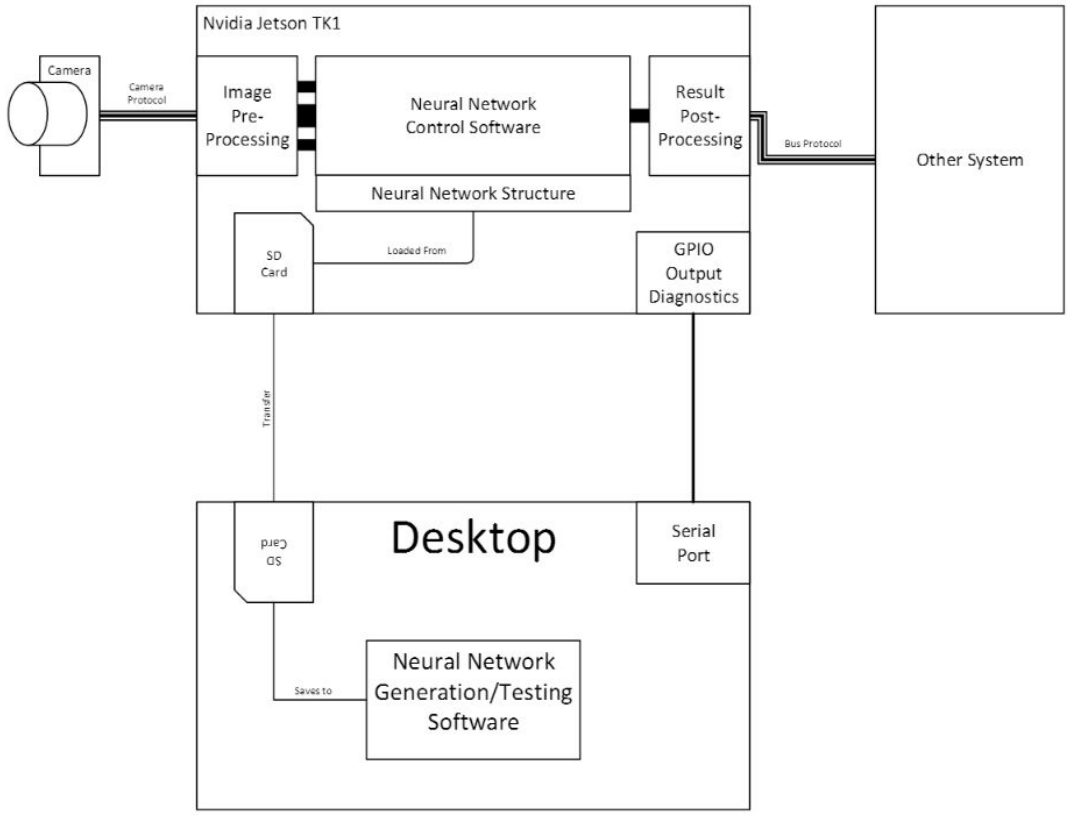
1) Install Jetpack on the system by following the Nvidia Doc:
   http://docs.nvidia.com/jetpack-tk1/1_1/content/developertools/mobile/jetpack_install.htm
   Here CUDA, cuDNN, and OpenCv for Tegra is installed
2) Install openCV (and all its requirements, version 3.1.0.)
   Please follow the instructions at:
   https://docs.opencv.org/trunk/d6/d15/tutorial_building_tegra_cuda.html
3) Install Tensorflow(intended use with the gpu but cpu version should also work)
   Please follow the pip installation for ubuntu:
   https://www.tensorflow.org/install/install_linux#determine_how_to_install_tensorflow
4) Once pip is installed from step 6, install the PIL and Tkinter libraries and prerequisites:
   $ sudo apt-get install python-dev libjpeg-dev libfreetype6-dev zlib1g-dev'
   $ sudo pip install PIL
5) Download a package of our source code
6) Extract the contents to desired location

Demo Instructions:

1) Determine whether a camera or video will be used for demoing the system.
2) If a camera is being used some code will need to be changed in the Camera.py file. By default a video file is being used.
3) $ cd folder/with/extracted/source/code
4) $ python main.py
5) After several seconds results should appear for the accuracy of the determined object

Appendix II: Previous Designs

Design 2.1



Our original design generalized the software components too much and we had some difficulty separating concerns. The image pre-processor was given the responsibility of modifying an image before it was fed into the neural network, the neural network would somehow process an image and provide some form of result, the post-processor handled editing the results of the neural network and communicating them, and it was not clear how to get the images or pass along the data between components. It was clear all the things that needed to happen, just not how to do it.

The original design used openCV 2.4.9 and Python 3 installed locally on our laptop. Initially functionality worked, but when moving the design to the TK1 it failed. Installations of openCV 2.4.9 would not interact with the camera. All pieces individually worked as expected, but they would not interact correctly with each other.

Design 2.2

This design changes the USB camera tested from a grayscale camera to a color camera. Based on the errors provided by openCV, the pixel format given by the camera could not be interpreted.

Design 2.3

The architecture evolved from a component based structure to a module based structure. This allowed for

better separation of concerns and for parts of the project to be easily exchanged.

Design 2.4

For this iteration, the whole project was moved to the TX1. This was needed to expand memory space for the Inception network.