Ma, Yishu
University of California, San Diego
October 31, 2016

Report for drone

The main idea of this project is sense augmentation with the help of drone, especially in those scenarios when senior citizens need help.

Hardware Platforms: Parrot Mini Spider (Bluetooth), Parrot Bebop2 (Wi-Fi), SamSung tablet, Android mobile phone

1. Mini drone: Not support video, a camera on vertical direction, no GPS

The Mini Spider has a very ideal size for indoor use. And also, it can work with application that need to connect to the drone and the internet simultaneously. However, it has very limited hardware that can support limited functions. Its Bluetooth connecting and low battery capacity make it also not capable of fast photo streaming.

2. Bebop2: FPV (First person view), a horizontal camera that has a very large scale of view, GPS

The Bebop2 drone is not very ideal for narrow indoor space use which can become unstable when approach like corners in a room. However, its capable camera and real-time video streaming can help achieve many other functions.

Part 1 Android Application

Things have been done

The main idea for this part is trying to achieve as much as we can based on the given sample code and Android functional API and libraries. Three ideas been tried for this part.
Software Platform: Android Studio
Programming Language: Java

Feature 1 Voice Control
Tool: Android RecognizerIntent with speech recognition API integrated
Firstly, I added the voice control function to the original android application. The goal of this part is to make the control of the drone to be as simple as possible. There do exists some simple voice control drone in market, like FX-21V. However, in FX-21V's case, it need to use a special controller with some processor integrated and a microphone and with very limited control instructions. On the other hand, our android voice control application takes advantage of existing speech recognition API and can enable a much larger scale of control instructions. Additionally, based on android, we can enable the voice control to be more fault-tolerant, like wrong recognition result for words with similar pronunciation.

(1) Open App File with Android Studio:
The code of this function is the folder named "Android_hash_voice_control_mobiledata". To develop Android Application, the software Android Studio is needed. The first time one run the application, please follow the following steps:
Open Android Studio->Open an existing Android Studio

Project->Android_hash_voice_control_mobiledata->SDKSample->buildWithPrecompiledSDK->build.gradle->OK
Then Android Studio will start to build the project and help add path of jdk, etc. The next time, just do Open Android Studio->Open an existing Android Studio Project->Android_hash_voice_control_mobiledata->SDKSample->buildWithPrecompiledSDK because the project has already been built as an Android Studio project.

(2) Code Details:
After the project opened by Android Studio, go to the folder java, there are two subfolders named com.parrot.sdksample and res. Under com.parrot.sdksample, there are four folders: activities, discovery, drone. Under res, there are three folders: layout, mipmap, values. When developing the application, **activity**, **drone and layout** are the three folders that are mainly word on.

(a) drone->BebopDrone.java
The BebopDrone. Java file in the folder drone is where properties of the Bebop drone are defined, like take off, land, take picture, media download and so on. Also for voice control, in this class, I added four hashtables which take distances expressed in text as keys and its corresponding flying durations as the value. In detail, *BebopFBTable* is the table for forward and backward fly; *BebopLRTable* is the table for left and right fly; *BebopUDTable* is the table for up and down fly; *BebopRTtable* is the table for rotation in the horizontal plane.

Additionally, I added two methods in this class named *setMaxVerticalSpeed()* and *setMaxRoatationSpeed()*. It is clear that the first method is used to set Bebop's max vertical speed(m/s) and the second method is for setting max rotation speed(degree/s). This two function is useful because when we clearly know Bebop's speed, then we can precisely know the time that is needed to fly to a certain distance or rotate to a certain degree. Therefore we can store precise values in *BebopUDTable* and *BebopRTtable*.

Note: up&down and rotate can be controlled precisely. However, because drone achieves left&right and forward&backward flying through leaning to some degree. Also these motions have varying acceleration that is influenced by motor power, leaning degree and air resistance. Therefore, speed is hard to determine for motions to this four directions if we don't have access to the data collected by accelerometer and Gyro. Therefore, the values stored in *BebopLRTable* and *BebopFBTable* are got by field experiments and measuring. They are not as precise as those in *BebopUDTable* and *BebopRTtable.*

(b) layout->activity_bebop_two.xml
Xml files under the folder layout are used to define the layout elements like bars, buttons and TextViews on screen of mobile phone. activity_bebop_two.xml is the layout file I implemented for voice control. Behaviors of all layouts are defined in activity classes.

(c) activity->BebopActivity_three.java
I have did many experiments for Bebop's activity and the final and right version is BebopActivity_three.java. This is the class where the properties of the bebop control part in this

android application are defined. And its corresponding layout file is activity_bebop_two.xml. Similarly, bebop_activity.xml is the corresponding layout file of BebopActivity.java which defines the original control screen with various physical control bars.

Speech recognition is achieved using the method *promptSpeechInput()*. To make the app work smoothly, we usually need to use multithreads that allow some process can run at the back ground and will not froze the main thread. Therefore, for speech recognition I used an Intent. And luckily, Android integrated its speech recognition API into **RecognizerIntent.** This is the intent that mainly supports different function for speech recognition. I used **ACTION_RECOGNIZE_SPEECH** to start an activity that prompt the user for speech and send it through a speech recognizer. An the method *onActivityResult()* works like a listener that show the recognized result on the screen and also store the corresponding text instruction once a result is sent back from server. And then, check the text instruction and convert it into corresponding motion of drone.

(d) Activity->getaddress_service&activity->GetAddressActivity
One problem I encountered with Bebop is about internet connection. Bebop uses Wi-Fi to connect with mobile devices and speech recognition also needs internet connection. Therefore, when using a mobile device that can only use Wi-Fi to access internet to control Bebop drone, the mobile device cannot connect to internet to do speech recognition because Wi-Fi channel will be occupied to communicate with the drone. Considering that under most cases, like outdoors, people will not have access to Wi-Fi, I chose to solve this problem by using mobile data. For a mobile device that have both access to Wi-Fi and mobile data, I developed the application to force data transmission with specific IP address to use mobile data even when Wi-Fi is connected.

Before forcing data's transmission, we need to know the real time IP address for the speech recognition API. I achieved this with an IntentService. As I have mentioned before, this kind of time consuming work should be done behind the scenes otherwise users can see the application become frozen for some time. This can badly worsen user experience. getaddress_service.java is a class for this IntentService that will run in another thread other than the main thread. In this class, I got the IP address by instantiating an instance of InetAddress and then get IP address through host name. And then, when IP address is got, just call the method *forceMobileConnectionForAddress()*

To let user know the address has been correctly got and they can start to run the application, I added another activity named GetAddressActivity. This activity will launched first when the application is started. And once this activity is launched, the getaddress_service will automatically be started.

Note: Voice control has also be implemented for Mini drone. The structure of code is the same. Mini drone. Properties of mini drone is the class in folder drone named MiniDrone; layout xml file is named activiti_minidrone_two; activity class that defines how does this application work for Mini drone is MiniDroneActivity_two.java. However, for Mini drone, we don't need to force data transmission through mobile data because Mini uses Bluetooth to connect with mobile device so that mobile device can use Wi-Fi to connect to internet and simultaneously use Bluetooth to control Mini drone.

Feature 2 Barcode reading
Tool: Google Play Services Barcode API (com.google.android.gms.vision.barcode)
I tried to do barcode reading with real time video streaming. However, due to image quality requirement, video streaming cannot successfully be recognized. Therefore, I choose to do barcode reading with image. The application will run in following steps:
user press button "BARCODEREADING"->a take picture instruction be sent to the drone->once picture is taken, drone transmitting the picture back to the mobile device-> read barcode->show the result on screen.

It is the same as voice control part that **activity**, **drone and layout** are the three folders that are mainly word on. And their functions are also the same. xml in layout defines how do screens of the application look like and Classes in activity defines how layout components work. Classes in drone defines properties of each kind of drone.

(a) layout->activity_bebop_two.xml
To achieve barcode reading function, I added a button named "BARCODEREADING" at the right bottom corner.

(b) activity->BebopActivity_three.java
To use Google's barcode API, google play services must first be included in the application. And then import the Barcode and BarcodeDetector parkages. In this activity, I implemented a function *readbarcode()* that take the path of a local image and do barcode reading and then output the result as a string on the screen. Also, I added a click listener for the "BARCODEREDING" button to send take picture instruction to Bebop drone.

(c) Modifications to listeners
Because we need the picture of barcode be transmitted back to the mobile device automatically once it is successfully taken, we need to use listeners to track the process and react when some events happens. Therefore the original method on *onPictureTaken()* and *onDownloadComplete()* of Bebop Listener should all be modified.

(d) Drone->SDCardModule
SDCardModule is a class where media The original SDK's has the feature that download multiple pictures for last flight all if last flight ID is available download all that day's pictures. This is not suitable for barcode reading because we only need one picture for the barcode. Therefore, in SDCardModule, I added two methods named *getLastPictureWithId()* and *getLastPictureWithoutId()* that enables Bebop drone to just transmit back the latest one picture for barcode reading.

Things can be done
1. Voice Control Optimization
For the voice control function, we still can optimize it. The speech recognition now has a low accuracy when people using it in a noisy environment or recognizing words with similar

pronunciations. I make a research and found that Google's newly released cloud speech recognition API may have a better performance (https://cloud.google.com/speech/). Or can generate an algorithm that can calculate the pronunciation distance between two text words.

2. Try Automatic following

We wanted to achieve automatic following for both the Mini drone and Bebop drone.

(1) Tried by Debjit with keeping tracking with signal strength. The problem of this method is one can only get the signal strength every time when a new connection set up. That is you will need to keep disconnecting and connecting from the drone. However, the drone will just hovering in the air after the first disconnection and will never be connected again. Therefore, automatic following for the mini drone has not been achieved.

(2) For Bebop2 drone. I also tried to achieve following through its GPS. I wanted to set Bebop's home location as the controller's location and then send return home instruction to the drone. There is a function in Parrot's library that can enable the drone to receive GPS coordinates of controller (in this case mobile device). However, the mobile device's GPS location can never be got correctly using Android's LocationManager and I also tested the controller's location stored inside Bebop2 after and it seems that function provided by Parrot was not able to send controller's coordinates correctly to the drone. Considering the precision of GPS, I did not spend too much time to solve this problem.

A structure diagram that shows how Parrot SDK works is included in appendix. When developing android application, I mainly worked with CONTROLLOR, DISCOVER and SDCARMODULE

Part 2 Object Detection Algorithm

Things have been done: Object Detection Model Training

Software Platform: Caffe Deep Learning Framework; Linux OS

Reference: Lu, Yongxi, Tara Javidi, and Svetlana Lazebnik. "Adaptive Object Detection Using Adjacency and Zoom Prediction." *arXiv preprint arXiv:1512.07711*(2015).

The main idea for this part is to achieve automatic focusing the drone to interesting objects. In last part, I have mentioned that we have achieved voice control which can make it much easier for user to communicate with the drone. Now we want to further simplify the controlling. It is not easy to use voice control to make the drone do fine tuning to focus on interesting objects. Therefore, we want the drone to do fine tuning by itself. Consider the scenario where we want the drone to automatically focus to barcodes and read them by itself so that senior citizens will not need to worry about the control.

The workflow of this part is:

- Setup a server that can do heavy computing so that it can do object detection.
- User will first sent the drone to a place around a product and then press read barcode button.
- The drone will take a picture of its current view and transmit it back to the mobile controller (mobile devices).
- The picture will then be sent to the server to do object detection and based on detection result the server will send back moving instructions to the mobile controller.
- Mobile controller will send those moving instructions to the drone to make it fix its position
- Keep running step2-step5 until the barcode is at the center of drone’s view and is clear enough to read the barcode.

Note: As I mentioned in last part, I implemented barcode reading for mobile device. It is also can be implemented to be done with the server.

Yongxi’s algorithm can achieve efficient object detection. Therefore I just trained a model based on his algorithm for detecting barcode in images.

1. Environment Set-up

To download and install required components, refer to the write up included in the following Github link:https://github.com/luyongxi/az-net.

2. Code Details

For this part, because the whole system is so complex, it is not possible to include every detail here. Therefore, I will mainly focus on how to add new datasets to the model and how to run the algorithm. For more details and theoretic background of deep leaning, please refer to Yongxi’s paper as mentioned before.

Open the folder named “az-net”, one can see several main folders. “caffe-fast-rcnn” is the folder where Caffe deep leaning framework are installed. When customizing this algorithm to work for specific objects, **“data”**, **“experiments”**, **“output”** and **“lib”** are the three folders that will be mainly worked on.

(1) Work with “data” folder

Database Reference: Robust Angle Invariant 1D Barcode Detection

Alessandro Zamberletti, Ignazio Gallo and Simone Albertini Proceedings of the 2nd Asian Conference on Pattern Recognition (ACPR), Okinawa, Japan, 2013

To train the model for detecting barcode object inside an image. I downloaded a dataset on line and named the folder as “DICMVOC2010”. To use the algorithm, data folders should follow a certain name pattern so that the dataset can be configured later.

(a) DICMVOC2010->VOC2010->Annotation

This folder stores annotation xml files of all images in this dataset and there files contain ground truth locations of barcode objects in images which are used to check the correctness of detection results.

(b) VOC2010->ImageSets->Main

This folder contains three text files that stores image numbers that should be used as training dataset, training-validating dataset and testing dataset. One can just modify these files to customize dataset segmentation.

(c) VOC2010->ImageSets->JPEGImages

This folder stores all of the images in this dataset.

(2) Work with “lib” folder

This folder contains code to configure datasets. It is very important.

(a) Lib->datasets

To add a new dataset and apply it to the algorithm, we need to first configure it. For each dataset, first a ***.py file should be generated. Therefore, for my barcode dataset, I generated the file “dicom_voc.py” that defines the dataset to be applied into this deep learning model. And in the “factory.py”, all dataset should be added into to it to be setup for the model. And for successful importing, all datasets should be imported at the header of “__init__.py” that initializes all these dataset classes.

(3) Work with “experiments”

This folder contains scripts to run the algorithm (training and testing) and also Log files for each run and network configuration files.

(a) Experiments->scripts

This folder contains scripts to train and test on different datasets. For my barcode dataset, the script is “exp_barcode_shared.sh”, where “shared” means the algorithm shares convolutional features between fast R-CNN and AZ-Net. There is no need to go to deep for this idea. To my mind, according to Yongxi’s paper, model with shared feature can be faster and more precise. In the “exp_barcode_shared.sh” file, one can modify the iteration times to run the training and testing phases. This number is not larger is better. The right number should be with which the loss stops decreasing and starts to increase again. I observed the loss of training and the current 10000 and 5000 works fine.

After proper training, we got the right model, then we can just run the last block of code in the “exp_barcode_shared.sh” for testing.

(b) Experiments->logs

This folder contains log file for each run of the algorithm. Because this algorithm will always run for a long time. For this small barcode dataset with only 155 images, it takes 2-3 hours to run both training and testing, therefore logs are essential for checking whether the algorithms works fine and whether parameters like iteration times are properly set.

(c) Experiments->cfgs

This folder contains files that configure the network for different datasets. For my barcode dataset, the file is named “dicomvoc.yml”. The only parameter that is always needed to be reset is the SNAPSHOT_ITERS and it should be the same with that used in “exp_barcode_shared.sh”. This number should be big enough to at least train the model for a few epochs (a scan through the entire training dataset).

(4) Work with “output”

(a) output->default->dicomvoc_2010_test->vdd16_az_net_shared_iter_10000

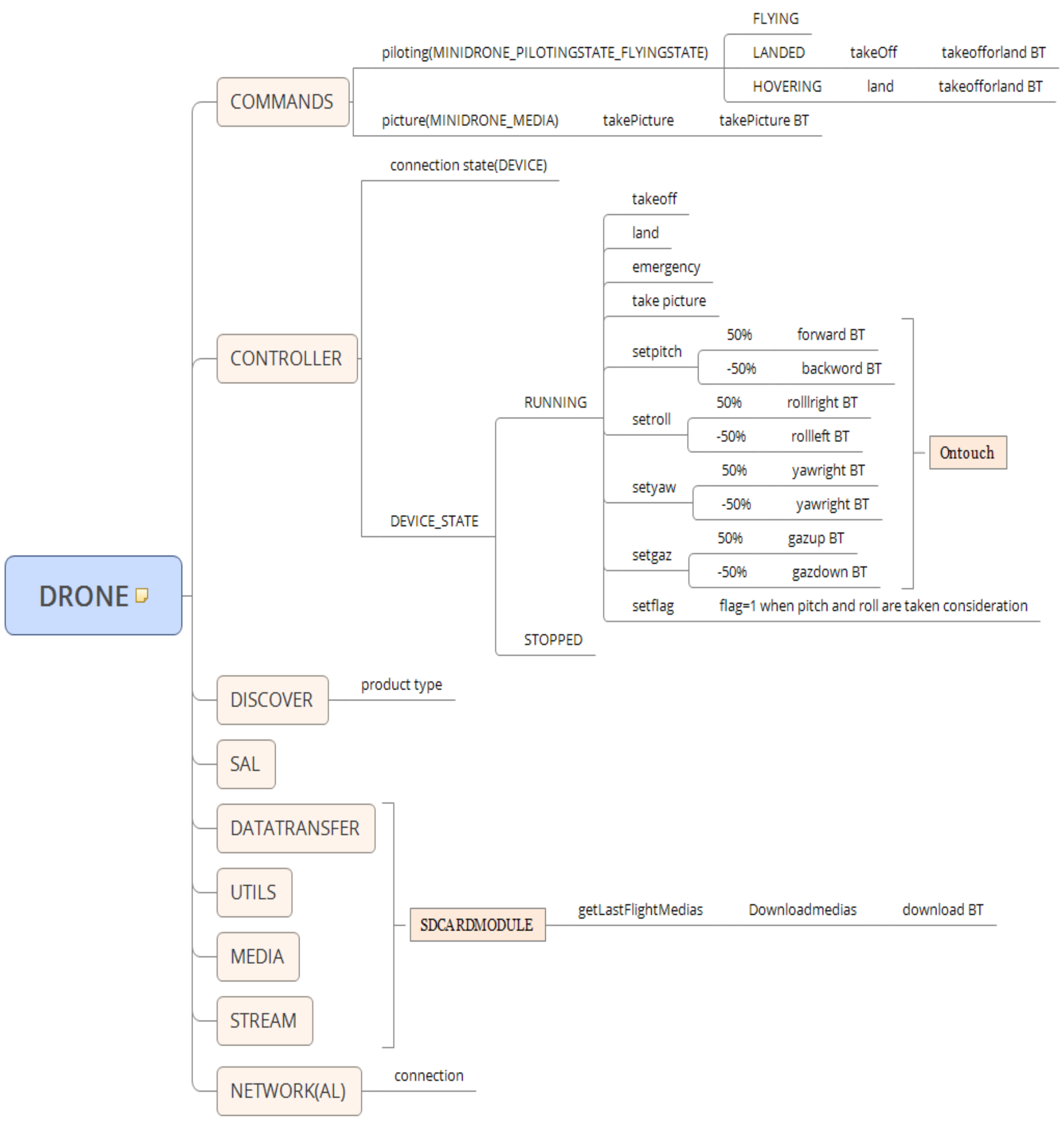
This folder contains the detection result of the training dataset named “detections.pkl”. This is a python package file and to read it, just convert it to some other file format like .mat. using the cPickle python module. The result file will contain several bounding boxes for the object on each testing image and also with a score. This score means how confident is the model to decide the object is at the certain location. For the drone controlling purpose, the most confident decision is enough because we have a loop to keep moving the drone toward the right location.

(b) output->default->dicomvoc_2010_trainval

This folder contains “proposal.pkl” that stores the locations of proposals generated during training phase. This result is not very important for the drone controlling purpose. Additionally, this is also the folder where trained Caffe models are stored.

Things to be done: Setup the Server

Appendix: Logic Diagram of Parrot SDK


DRONE
COMMANDS
piloting(MINIDRONE_PILOTINGSTATE_FLYINGSTATE)
FLYING
LANDED
takeOff
takeofforland BT
HOVERING
land
takeofforland BT
picture(MINIDRONE_MEDIA)
takePicture
takePicture BT
CONTROLLER
connection state(DEVICE)
DEVICE_STATE
RUNNING
takeoff
land
emergency
take picture
setpitch
50%
forward BT
-50%
backword BT
setroll
50%
rolllright BT
-50%
rollleft BT
setyaw
50%
yawright BT
-50%
yawright BT
setgaz
50%
gazup BT
-50%
gazdown BT
setflag
flag=1 when pitch and roll are taken consideration
Ontouch
STOPPED
DISCOVER
product type
SAL
DATATRANSFER
UTILS
MEDIA
STREAM
SDCARDMODULE
getLastFlightMedias
Downloadmedias
download BT
NETWORK(AL)
connection