Design Documentation

By: Masa Hu, Rory Main
https://github.com/yhu9/TrashClassification_capstone

Implementation Overview

Overview of hexacopter(hereafter referred to as "drone") functionality:

1. Equipment
2. Navigation
3. Picture Capture

Additionally there will be a post capture image processing in order to determine percentage distributions of specified classifications.

1) Equipment

The drone should be equipped with the following gear:

- Pixhawk
- 3DR GPS and Compass Module
- Raspberry Pi
- Raspberry PI 5MP Camera Board Module
- Li-Po Battery
- GoPro Hero 3
- GoPro Camera Mount

2) Navigation:

For this project we seek to automate the navigation of a drone in a semi-enclosed area without the use of GPS. At the time of writing we do not have the gear necessary to enact this plan, but we will outline how it can be achieved. The plan to achieve this is as follows:

1. A Raspberry Pi compatible camera will point towards the areas ceiling while in flight.
2. The Raspberry Pi camera will take a picture every half a second, and send it to a python script running on the Raspberry Pi.
3. An image recognition program running on the Raspberry Pi will identify what percentage of the image is identified as a roof, this is to tell when the drone has left the roofed area.
4. MAVLINK commands will be stored in a script as a list of lists of strings.

5. These commands, if run in the right locations, will set the drone along a preset path.
6. To determine when to launch the MAVLINK commands, we will use the pictures taken of the roof.
7. If three successive images return as having greater than X percent non-roof material (X will be tuned as necessary, will initially be set to 70%), then we can assume the drone has left the roofed area, and the script will send the next list of MAVLINK commands to the Pixhawk.
8. The Pixhawk will execute the MAVLINK commands, controlling the drones movements.
9. This will continue until the last set of commands are reached, which will land the drone.

3) Picture Capture

During flight the drone is to capture pictures of the ground below it. These pictures will then be processed to be useable by the trash recognition program.

1. To capture pictures of the ground, the drone is equipped with a mounted GoPro Hero 3 camera.
2. Before the drone launches, an operator will set the GoPro to capture pictures every half a second.
3. As the drone flies, many pictures of the trash will be gathered.
4. When the drone lands, and is disarmed, an operator will retrieve and cease picture capture of the GoPro.
5. For the moment, an operator will have to plug the GoPro into a computer and remove pictures from the beginning and the end of the flight, which do not contain trash.
6. If the drone has to take multiple passes to cover the whole area, the operator will have to split the photos into folders based on each pass. These splits will be identifiable by clusters of photos where the drone has temporarily left the roofed area.
7. In the future we hope this photo management can be handled by a script, possibly using marked out areas on the ground and image recognition, but this is currently not in the scope of our project.
8. The split clusters of photos will be loaded into AutoStitch[6], a program which will stitch together each pass into large panoramas, removing overlap and reducing the number of photos to hand to the trash classification program.

Post Image Capture Processing:

This process uses the images captured by the drone in order to create percentage classifications of the image within the specified categories. These categories currently include:

- cardboard
- tree matter (leaves, plywood, branches, leaves, lawn grass)
- construction waste (concrete, pipes, bars)
- general household goods
- trashbags

Currently a Support Vector Machine (SVM) is being used for the classification of the images. Each instance vector is a concatenation of each channel for the HSV color distribution of the image (H + S + V). In order to apply SVM we use svm_light [5] an open source software.

The following two steps are taken in order to accomplish the creation of these classifications:

1. model creations
2. image testing using the models

1) model creations

svm_light requires the creation of a model for each classification category. Inner workings of the model creation goes through several image processing steps as well as feature extraction methods.

1. pass in an image
2. segmentation
    a. open the image as a colored image
    b. convert to greyscale
    c. binarize using otsu's algorithm
    d. find the contours
    e. draw the contours
    f. find the connected components from drawn contours
    g. apply watershed algorithm using connected components as markers
    h. return markers
3. HSVFeatureExtraction
    a. convert image to HSV
    b. take a segment from step 2
    c. extract H color distribution
    d. extract S color distribution
    e. extract V color distribution
    f. concatenate H distribution, S distribution, and V distribution
    g. store concatenation as a feature vector instance
    h. repeat steps a - g for the rest of the segments
    i. return feature vector instances
4. Write feature vector instances to a file
5. Parse file to ready it for svm_light using (+/-) for the group
6. repeat steps 1 - 5 for all images that are to be included in the group and concatenate all files into a single file
7. repeat steps 1 - 5 for all images that are to be excluded from the group and concatenate all files into a single file
8. shuffle the files from step 6 and 7 so the instances are randomized

9. Take the shortest file between step 6 and 7 and concatenate the two files using the shorter of the two as the number of lines from each file that should be taken. This is done in order to make the number of instances equal for both groups
10. create the model using the concatenated file from step 9 and svm_learn [5]
11. repeat steps 1-10 for the other classification groups.

All of this is taken care of using the createModel.sh script. The only thing that one needs to do is create the folder of the category that is to be classified. Then create the ingroup and outgroup within it. The ingroup directory should hold all images which are inside the category, and the outgroup directory should hold all images which are outside the category. Further details can be found in file structure.

2) image testing using the models

The models created from step 1 are used in accordance with the image to see the pixel percentage as well as the segment percentage of the image that was classified as part of the group using the model. This step takes advantage of the fact that the segmentation process produces the same result every time.

1. pass in an image
2. segment the image
3. HSV feature extraction
4. write features to a file
5. parse file to be included into the group for svm
6. pass in a model
7. classify the parsed file from step 5 using the model
8. save the classification file and the accuracy
9. attempt to stitch the image back together
    a. segment the image
    b. find unique markers
    c. for each classification and each unique marker
    d. if classification for marker is in the group color it black
    e. if classification for marker is out of the group color it white
    f. if classification for marker is unknown, color it grey
    g. percent accuracy = (black + grey) / white
    h. print percent accuracy to console
10. show the image

Required Software

1. AutoStitch Demo[6]

    a. If this project continues into commercial application, we should either pursue a license, or purchase one of the licensed commercial products implementing AutoStitch.
2. Ubuntu Operating System 14.04 or higher
    a. bash 4.3.11 or higher
    b. python-opencv 2.4.8 or higher
        i. sudo apt-get install python-opencv
    c. numpy
        i. sudo apt-get install python-numpy
    d. matplotlib
        i. apt-get install python-matplotlib
        ii. http://matplotlib.org/users/installing.html
    e. Tkinter 8.6
        i. apt-get install python-tk

Program Descriptions

Table of contents:

Any and all pseudo code used in the documentation of this project is not the literal code but is an overview of the actual code.

Descriptions:

1. segmentModule.py

This module exports a single function called getSegments(imageIn, SHOW) to create a Mat image of the image file passed in and the markers for the segments after segmentation. Creation of the mat image is simply reading the image file in with color. However, segmentation of the image takes several steps to accomplish.

getSegments()

Inputs:	STRING	file directory of the image to be segmented
	BOOL	SHOW flag if you want to see the process
Outputs:	MAT	mat image of the file passed in
	NUMPY ARRAY	markers for the segments

Process:

1. conversion to grey_scale [4]
2. binarization of the grey_scaled image using Otsu's algorithm [2]
3. find the contours of the image [1]
4. draw the contours [3]
5. find connected components [1]
6. apply watershed algorithm with connected components as markers [4]
7. return markers

The exported function also allows for the process to be viewed using matplotlib's user interface if the SHOW flag has a value of True.

2. extractionModule.py

This module exports the functions extractFeatures(), writeFeatures(). The purpose of this module is to extract features from a processed image which has gone through the segmentation process. Currently extractFeatures only implements the extraction of the a frequency distribution of HSV colors from the image.

extractFeatures()

Inputs:	MAT	image for extraction
	NP ARRAY	markers of segments
	BOOL	show flag
Outputs:	LIST of FLOAT	features extracted

Process:

```
1. Find how many unique markers there are besides the marker -1. (-1 is the marker for
   unknown regions as specified by the watershed algorithm by opencv [4])
2. Convert colored image to HSV image
3. features = empty list
4. for x in uniquemarkers
    a. region = getRegion(x) from image
    b. histrH = createHistogram(H) from image with domain [1,170]
    c. histrS = createHistogram(S) from image with domain [1,255]
    d. histrV = createHistogram(V) from image with domain [1,255]
    e. nH = normalize histrH with max(histrH)
    f. nS = normalize histrS with max(histrS)
    g. nV = normalize histrV with max(histrV)
    h. features <- (nH + nS + nV)
5. return features
```

If the show flag is True, you can watch the extraction of the histogram distributions for each region found.

writeFeatures()

Input:	LIST	features to be written to a file
	STRING	file name to be written to
Output:	BOOL	upon successful completion returns True else False

Process:

```
● open file for writing
● for f in features
    ○ for histr in f
    ○ for val in histr
        ■ file <- val
        ■ file <- “,”
    ○ file <- “\n”
● return True
```

Makes a file with the features using a “,” as the delimiter for tokenization.

3. execute.py

The main execution point for usage of the segmentationModule and the extractionModule. The proper steps used for this is:

1. getSegments
2. extractFeatures
3. writeFeatures

The getSegments step also handles several preprocessing steps as specified by the segmentationModule. This python program takes command line arguments and outputs a file in the working directory.

Inputs:	STRING	path to image file
	STRING	path to output file
	STRING	(show/noshow)
Outputs:	FILE	file named with 2nd input argument

If the image file doesn't exist the program will crash. The third argument show/noshow determines whether one wants to see the process as the execute program runs, but the argument must be literally "show" for this to happen.

4. parseForSVM.py

This python program takes a file created by execute.py and parses it to create a version of the file that can be used for svm_light [5]. Also, since svm_light is for two group classification, when the parsed file is created all instances are specified as one of the two groups passed in as an argument. Arguments must be passed as command line arguments.

Input:	STRING	path to file created by execute.py
	STRING	name of file for the parsed version
	STRING	(+/-) the group to be classified as
Output:	FILE	a file named as the 2nd argument

5. stichit.py

This takes an image and the classifications found for that image using svm, and stitches the image back together according to the classifications. The output gives a pixel percentage of the image that was classified as the “+” group. Unknown regions that are marked -1 are also classified as the “+” group. This program takes command line arguments. Make sure the classification file is for the image specified or this program will throw an error.

Finally, it is important to note that this program is best to not be used by itself and should be called using classifyImageWithModel.sh. The reasoning is that execute.py, parseForSVM.py, and svm_classify must be called first in order to acquire a classification file that can be used with the image.

Input:	STRING	path of image to be stitched
	STRING	path of svm classifications of the image
Output:	Terminal Output	accuracy of the image

Process:
```
1. blank = blank image
2. blank[markers == -1] = grey
3. for c, um in classifications, uniquemarkers
    a. if c > 0
        i. blank[markers == um] = white
    b. if c <= 0
        i. blank[markers == um] = black
4. percentWhite = (white + grey) / total * 100
```

This gives percentage value over total number of pixels on the image. SVM classifies anything in the ingroup as a positive number and anything in the outgroup as a negative number. Distance from the separating plane is also given in the classification file but it is not used here.

6. createmodel.sh

A bash script that creates a svm model given a groupname, directory of images in the “+” group, directory of images in the “-” group, and a show flag. This script takes care of implementing execute.py on multiple images and executing parseForSVM.py on each output from execute.py. Once all files are parsed for SVM, the script concatenates each file from both groups into a single file for each group. Those files are then shuffled and recombined into a single file with equal number of instances. The training uses svm_light [5] to create the model. All files are moved to their proper location or deleted accordingly.

IMPORTANT:
Please specify the correct directory names at the top in order to properly organize everything. The script will still run but the working directory will become cluttered and unintended bugs may arise.

Input:	STRING	groupname
	STRING	path to ingroup “+”
	STRING	path to outgroup “-”
	STRING	(show/noshow)
Output:	MODEL FILE	model that can be used for classification with svm_light
	TRAINING FILE	text file containing all instances used to create the model

Process:

- for image in ingroup_directory
  - python execute.py image tmp1.txt showflag
  - python parseForSVM.py tmp1.txt tmp2.txt +
  - cat tmp2.txt >> ingroup.txt
  - rm tmp1.txt
  - rm tmp2.txt
- for image in outgroup_directory
  - python execute.py image tmp1.txt showflag
  - python parseForSVM.py tmp1.txt tmp2.txt -
  - cat tmp2.txt >> outgroup.txt
  - rm tmp1.txt
  - rm tmp2.txt
- shuffle ingroup
- shuffle outgroup
- length = findShortest(ingroup.txt,outgroup.txt)
- take -n length ingroup.txt >> training.txt
- take -n length outgroup.txt >> training.txt
- rm ingroup.txt
- rm outgroup.txt
- mv training.txt training_directory

Recreating the model is as simple as re-running the script with new images in the file directory.

7. classifyImageWithModel.sh

Given a model and an image, this script tests how much of the image is classified within the ingroup using the model. This will also output a nice picture with black as the ingroup and white as the outgroup, and grey as unknown. Accuracy values for pixel percentage inside the ingroup is printed to the console.

IMPORTANT: the script assumes the existence of the folder classification/ in the working directory

Input:	STRING	path of image file to be used
	STRING	path of model file to be used
Output:	FILE	classification.txt
	FILE	classification_accuracy.txt
	CONSOLE OUTPUT	pixel percentage of image

8. svm_learn [5]

This is a binary for svm_light. Implementation as well as usage guides for this can be found on the following website:

svmlight.joachims.org

9. svm_classify

This is a binary for svm_light. Implementation as well as usage guides for this can be found on the following website:

svmlight.joachims.org

10. Drone Navigation Pseudo Code

At the moment of writing we do not have the gear to implement this feature, so this is pseudo code to plan out future implementation.

A) RoofRecognizer.py

Input:	Image	Image from roof camera.
	FLOAT	Percent of photo that is not roof before returning true.
Output:	BOOL	True if X% of the image is not classified as a roof. Otherwise false.

This function will take in photos from the Raspberry Pi camera, and a percentage as a float. This function should utilize our trash classification algorithm, except that it is only trained to identify the roof of the transfer facility. If the algorithm returns that greater than X% of the image is not the roof, then it should return true. Otherwise it will return false.

B) CommandLauncher.py

Output:	STRING	MAVLINK command to be run in terminal.

This function stores a preset list of lists of strings. These strings are MAVLINK commands that can be run in console to send them to the Pixhawk and control the drone. These commands were found on the Ardupilot[7], and Pixhawk[8] websites. The commands we are interested in are:

- MAV_CMD_DO_CHANGE_SPEED
  - "Change the target horizontal speed and/or throttle of the vehicle. The changes will be used until they are explicitly changed again or the device is rebooted"
- MAV_CMD_CONDITION_YAW
  - "Point (yaw) the nose of the vehicle towards a specified heading."
- MAV_CMD_NAV_TAKEOFF
  - "Takeoff (either from ground or by hand-launch). It should be the first command of nearly all Plane and Copter missions."
- MAV_CMD_NAV_LAND
  - "Land the vehicle at the current or a specified location."

From our understanding, the CHANGE_SPEED and CONDITION_YAW commands should be able to be used without GPS. However I do believe that the TAKEOFF and LAND might require GPS. This is unclear without testing, as the documentation assumes GPS will be available.

Here is some Pseudo code for how this solution might work:

```
listOfCommands =
[[MAV_CMD_NAV_TAKEOFF,MAV_CMD_DO_CHANGE_SPEED(0,0.5,.3,0)] ,
[MAV_CMD_DO_CHANGE_SPEED(0,0,.3,0), MAV_CMD_CONDITION_YAW(180,40,1,1),
MAV_CMD_DO_CHANGE_SPEED(0,0.5,.3,0)],
 [[MAV_CMD_DO_CHANGE_SPEED(0,0,.3,0), MAV_CMD_NAV_LAND]]

os.system(first list of commands)

if(RoofRecognizer returns true three times sequentially):
        For each command in next list:
                os.system(command)
```

These commands don't include all the parameters, and the exact parameters used will need to be tested, but in general what it says is this:

- Takeoff, set speed to 0.5 m/s, with 30% throttle
- Wait until three consecutive pictures of the roof return as X% not roof.
- Change speed to 0 m/s with 30% throttle, rotate yaw 180 degrees relative to our position, at 40 degrees a second, change speed to 0.5 m/s with 30% throttle.
- Wait until three consecutive pictures of the roof return as X% not roof.
- Change speed to 0m/s with 30% throttle, Land in current position.

Please note, this potential solution is as of yet untested, and any attempts to navigate with MAVLINK commands should be handled with care. Until a method is proven to be successful at avoiding collisions, a skilled operator should be prepared to take control of the drone at any moment.

File Structure

The project can be found in the github repository. However, there are certain things in the directory that are peculiar to the usage of the program. Therefore, some clarification is needed for the proper usage of this project.

1. Products - this folder holds any and all documentation created for this project
2. Source - this folder contains all the source code information as well as the images and categories used for the classifications
    a. categories - holds the directory with all the images for the classification groups
        i. holds the category groups
            1. ingroup - holds all images that are to be classified in the group

        2. outgroup - holds all images that are to be classified outside the group
    b. classification - holds the classification accuracy for the most recent classification and the classifications for each instance that was passed in
    c. future_material - holds possible machine learning programs that could be used for the future of the project
    d. models - contains all the models
    e. execute.py - main program implements extractionModule and the segmentModule
    f. extractionModule.py - exports extraction functions
    g. extractionModule.pyc - compiled version of the extraction module
    h. parseForSVM.py - parses a file from execute.py to be used for SVM
    i. segmentModule.py - exports the segmentation functions
    j. segmentModule.pyc - compiled version of segmentModule.py
    k. svm_classify - svm_light binary [5]
    l. svm_learn - svm_light binary [5]
    m. SVMLICENSE.txt - svm_light license
    n. createmodel.sh - creates the model given the proper inputs
    o. classifyImageWithModel.sh - classifies image according to the model passed
    p. readme.txt - contains information of the required software for usage of the project
    q. stitchit.py - implements putting the image together into a classified picture with percentages for how much of the image is in the classification
3. README.md - some information pertinant to the project

References

1. http://docs.opencv.org/3.1.0/d3/dc0/group__imgproc__shape.html#gac2718a64ade63475425558aa669a943a
2. http://docs.opencv.org/3.1.0/d7/d4d/tutorial_py_thresholding.html
3. http://docs.opencv.org/2.4.2/modules/core/doc/drawing_functions.html
4. http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html
5. T. Joachims, Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning, B. Schölkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999.
6. http://matthewalunbrown.com/autostitch/autostitch.html
7. http://ardupilot.org/planner/docs/common-mavlink-mission-command-messages-mav_cmd.html#common-mavlink-mission-command-messages-mav-cmd
8. https://pixhawk.ethz.ch/mavlink/