

## Part C Project Notes

Catherine Vlasov

October 31, 2018

# Contents

<b>1</b>	<b>Task Documentation</b>	<b>2</b>
<b>2</b>	<b>Meeting Notes</b>	<b>3</b>
2.1	24/10/18 . . . . .	3
2.2	17/10/18 . . . . .	6
2.3	03/10/18 . . . . .	6
<b>3</b>	<b>Notes to Self</b>	<b>8</b>
3.1	Useful Commands . . . . .	8
3.2	Script Timings . . . . .	8
3.3	Lessons Learned . . . . .	8

## Chapter 1

# Task Documentation

## Chapter 2

# Meeting Notes

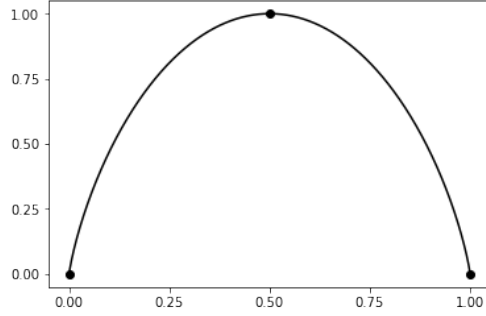
### 2.1 24/10/18

- What I did:
  - Read Chapter 3 of the Advanced Security notes on steganography
  - Wrote a script (`initial_curation.py`) to find all the largest images in the `actor00003` directory and then make them all grayscale and landscape
    - \* Wasn't quite working due to "Empty input file" error when performing multiple `jpegtran` operations
- Action plan:
  1. Calculate image sizes
    - Preserve the 4:3 aspect ratio, not because we have to but because we can and it means we can keep things as similar as possible
    - The largest image size we'll use is  $3072 \times 2304$  since that's the size of the largest `actor00003` images.
    - The smallest size will be  $320 \times 240$  since that's a relatively common image size (and it has a 4:3 aspect ratio)
    - The short-edge dimensions will be computed by hand by calculating  $240x$  (where  $x = \sqrt{1}, \sqrt{2}, \dots, \sqrt{10}$ ) and then rounding to the nearest multiple of 24. Then the long-edge dimensions are calculated such that the 4:3 ratio is maintained.
  2. Create the directory structure on the server in `/array/vlasov/`
    - Keep a copy of all the original images in `actor00003/`
    - Create one directory per image size, called `size3072` (for instance)
    - For each size, create two subdirectories:

- (a) One for the unaltered images, called **cover**
  - (b) One per number of payload bits, called **stego-1234bits**
- Each **cover** subdirectory will have three files per cover image:
  - (a) **image12345.jpg**: the unaltered image
  - (b) **image12345.costs**: the costs computed by J-UNIWARD
  - (c) **image12345.fea**: the features computed by JRM
- Each **stego-1234bits** subdirectory will have one file per stego image:
  - (a) **image12345.jpg**: the stego image, which is the cover image **sizeXXXX/cover/image12345.jpg** with a 1234-bit message embedded in it
- 3. Crop the  $3072 \times 2304$  cover images to the sizes calculated in task 1. Do this by cropping  $8 \times 8$  pixel blocks evenly from the top/bottom and right/left.
- 4. Generate the costs (using Dr. Ker's slightly modified J-UNIWARD code) and features (using JRM) for all the cover images of all the different sizes.
  - JRM produces 22510 real numbers (the features)
  - Up to me how to store them, but ASCII is probably the most portable
- 5. Use J-UNIWARD to embed 0.4 bits per non-zero AC coefficient in some of the covers
- 6. Write a function that takes a payload size (as the number of bits) as input and computes the probabilities with which each coefficient changes during (binary) embedding.
  - Goal: given the costs  $c_1, c_2, \dots, c_N$  (where N is the total number of coefficients) of changing each coefficient (by adding/subtracting one), compute the probabilities  $\pi_1, \pi_2, \dots, \pi_N$  of making each of these changes
  - Size of the payload:  $\sum_{i=1}^N H_2(\pi_i)$ 
    - \*  $H_2$  is the "entropy" and is defined as:

$$H_2(x) = -x \times \log_2 x - (1 - x) \times \log_2(1 - x)$$

- \* Graph of  $H_2$ :



- Average total cost:  $\sum_{i=1}^N c_i \pi_i$
- Two (equivalent) optimization problems for computing the payload size:
  - (a) Distortion-limited sender (DLS)

$$\text{Maximize } \sum_{i=1}^N H_2(\pi_i) \text{ such that } \sum_{i=1}^N c_i \pi_i \leq C$$

- (b) Payload-limited sender (PLS)

$$\text{Minimize } \sum_{i=1}^N c_i \pi_i \text{ such that } \sum_{i=1}^N H_2(\pi_i) \geq M$$

- For some fixed  $\lambda$ , we can compute the probabilities:

$$\pi_i = \frac{1}{1 + e^{\lambda \cdot c_i}}$$

- We'll use PLS, where M is the payload size.

- \* The optimal solution is when  $\sum_{i=1}^N H_2(\pi_i) = M$
  - \*  $\sum_{i=1}^N H_2(\pi_i)$  is actually monotonically increasing, so we can find a value of  $\lambda$  such that  $\sum_{i=1}^N H_2(\pi_i) = M$  for any M we choose. Then, we can compute the probabilities  $\pi_1, \pi_2, \dots, \pi_N$  using this value of  $\lambda$ .
  - \* The end goal is to do the embedding ourselves by modifying each coefficient with these probabilities.
- *Is 80 a standard JPEG quality factor (QF)?* In the massive image database released by Flickr, the most common QFs were 100, the QF used by iPhones, and 80. So we're using 80 because that gives us a greater selection of images.

## 2.2 17/10/18

- What I did:
  - Read Chapters 1 and 2 of the Advanced Security notes on steganography
  - Read the 2008 paper “The Square Root Law of Steganographic Capacity”
- Discussed questions I had about Chapter 1 (Steganography) and Chapter 2 (Steganalysis) of the Advanced Security notes and about the 2008 paper.
  - *What is downsampling?* Shrinking
  - *When you take a pictures on your phone, what happens?* Captures raw image, immediately compresses it as a JPEG, and discards the raw image
  - *What determines a cover’s “source”?* Primarily the camera. The camera’s ISO setting, in particular, is very important. The subject of the photos don’t make much of a difference.
  - *In JPEG compression, don’t you lose some information when dividing the image into  $8 \times 8$  pixel blocks?* No, the DCT is linear (i.e. 1-to-1 mapping from  $8 \times 8$  blocks to coefficients)
  - *Is a JPEG decompressed every time you view it on a computer?* Yes
  - *When LSBR is used on RGB images, which bit(s) are changed?* Good question - it depends, but usually the LSBs of all three components (in sync)
- After embedding a payload, the original cover is destroyed. Otherwise, two nearly identical images would be floating around and Alice could easily be outed if someone got their hands on both versions.

## 2.3 03/10/18

- What I did: N/A
- Discussed software to be used for embedding (J-UNIWARD), feature extraction (JRM), and detection (ensemble of linear classifiers)
  - All the software is here
- Server’s IP: 163.1.88.150
- Amounts of payload to embed:  $O(1)$ ,  $O(\sqrt{n})$ ,  $O(\sqrt{n} \log n)$ ,  $O(n)$

- $m \sim \frac{\sqrt{DC}}{2} \log \frac{C}{D}$
- TIME EVERYTHING
- I will test new embedding and new detecting methods and I could also try old embedding and new detecting methods
- Total amount of space needed (assuming around 10,000 images are used):
  - Images:  $2MB \times 10000 \times 9 \approx 180GB$
  - Costs:  $8B \times 5M \times 10000 \approx 400GB$
  - Features:  $170KB \times 10000 \times 9 \approx 17GB$



## Chapter 3

# Notes to Self

### 3.1 Useful Commands

- Run a command in the background so that you can keep using the terminal or close it
  - `nohup python script.py &> script_output.out &`
- Check on processes that are running
  - `ps aux | grep vlasov`

### 3.2 Script Timings

- `initial_curation.py`
  - $1131.18478608s \approx 18m51s$  (30/10/18)

### 3.3 Lessons Learned

- The input and output file to `jpegtran` can't be the same, otherwise you get an “Empty input file” error.