Decoding the San Jose Semaphore

Mark Snesrud
msnesrud@yahoo.com

Bob Mayo
bob-sjs@bobmayo.com

OVERVIEW

The San Jose Semaphore is a piece of animated public art located atop Adobe's building in San Jose, beginning operation on August 7, 2006. It consists of four orange illuminated discs that rotate every 7.2 seconds, along with a soundtrack available in the immediate proximity on AM 1680 radio. In addition to the esthetics of the art, the movements of the discs and the sounds convey an encoded message. This report describes the successful decoding of that message. More information on the San Jose Semaphore is available at www.sanjosesemaphore.org.

The Semaphore atop Adobe's headquarters in San Jose, California.

Introduction

It was a Thursday night in downtown San Jose, where we decided to meet for a beer and dinner along with a few other friends and co-workers. Not much was going on at the place where we ate dinner, so we decided to explore the surrounding areas to see if we could locate a fun place to hang out. As we crossed Almaden Boulevard, the orange discs of the San Jose Semaphore stood out against the night sky, presenting a display that engaged our curiosity. Not much progress was made on our exploration of San Jose that night, but two of us decided to try to solve the puzzle that the Semaphore posed.

Scaling the Wall Next to the Elevator

We noticed the online simulcast of the semaphore, but wasn't sure if contained the encoded message, too. The simulcast did not behave exactly like the version on the Adobe building, in that the actual art varied its appearance when airplanes flew over, while the online simulcast did not. Bob conjectured that we would need to videotape a few hours of the semaphore, and perhaps even set up a webcam for long term operation.

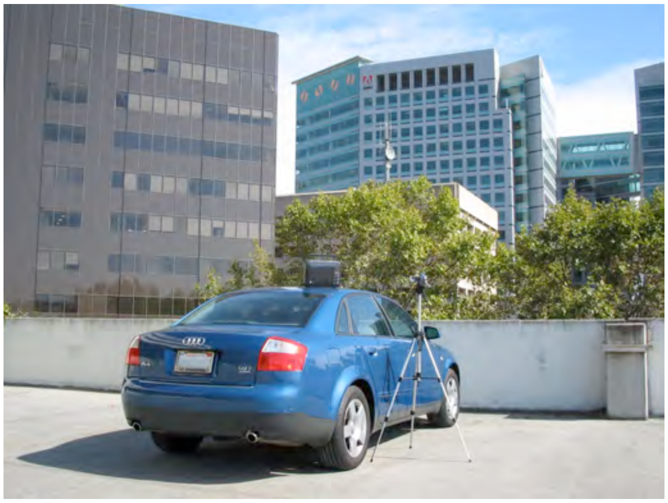


Fig. 1: Filming the Semaphore. The Semaphore was easily visible throughout the area, although clear reception of the associated AM radio signal was problematic. Since 24/7 monthly parking rental was available for this location, we thought we might leave a vehicle there with a webcam and uplink.

While it was easy to film the semaphore, capturing a clean audio track from AM 1680 radio was more difficult. Using a simple handheld radio we were not able to get a clean signal anywhere in the area. Switching to a larger boom-box type of radio helped a little bit, but was still not good enough if we were counting on it to capture all the data. So Bob purchased a Grundig G1000A radio, which reviews on the web said was an excellent radio for receiving weak AM radio stations.

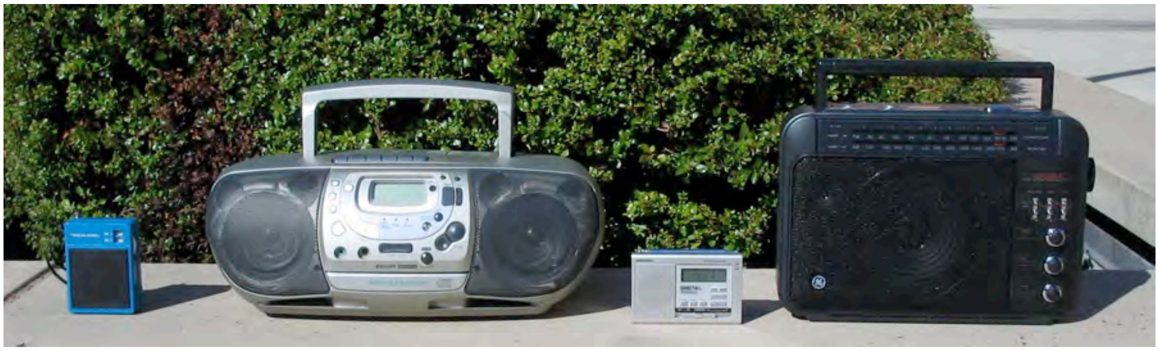


Fig 2: A Succession of Radios. Several radios were tried in an attempt to get a strong AM audio signal. The GE SuperRadio on the right is specially designed to receive weak signals.

Alas, the Grundig worked better in most locations, but still did not produce a signal we were satisfied with. Unexpectedly, one of the strongest signals was received far away from the Adobe building, near a railing that we hypothesize had something to do with enhancing reception. But the signal was still weak and faded in and out more than we liked. Next step: a GE SuperRadio, the king of long distance AM radio reception.

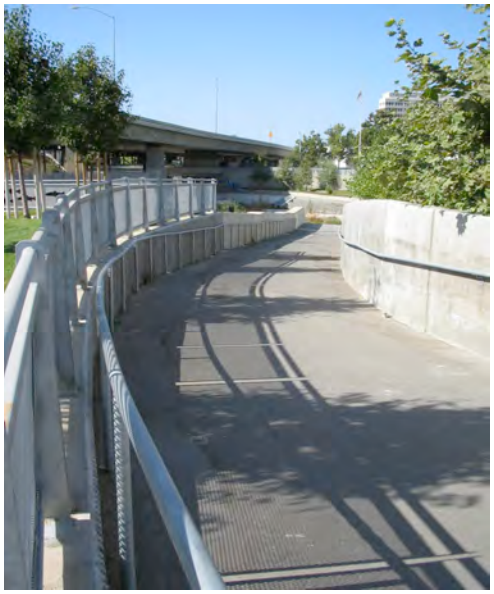


Fig 3: Way out here? Surprisingly, AM radio reception was good in this location far behind the Adobe building, perhaps due to the railing or other metal objects acting like antennas.

The GE radio lived up to its claim to fame. When a block away from the semaphore, it managed to pull in a radio station from Boulder, Colorado instead of the semaphore. But when close to the semaphore, it locked on and produced an excellent signal. Somehow, we needed a way to webcast this signal from near the Adobe building. The news rack in front caught our eye.

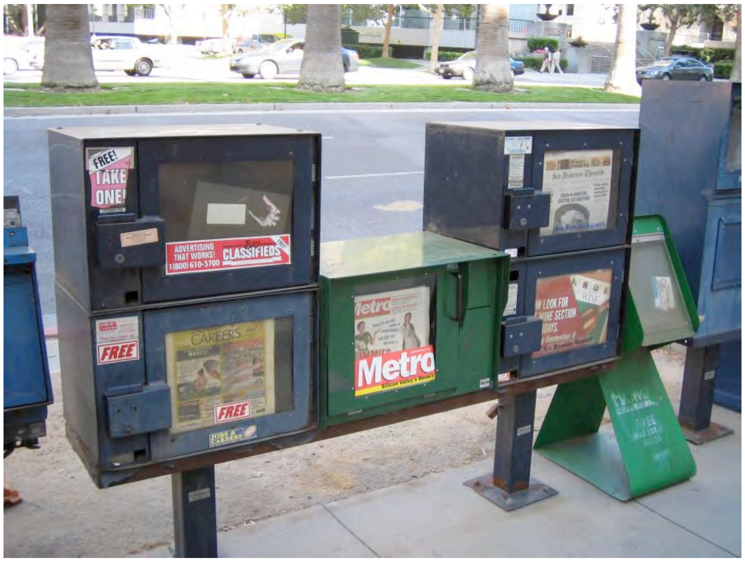


Fig 4: Future home of webcam and radio? This spot in front of the semaphore had excellent AM radio reception and also a clear view of the semaphore. And, conveniently, room for one more newspaper vending box to be added.

This news rack is located close to the semaphore and provides excellent AM radio reception with the GE SuperRadio, and also has a clear view of the Semaphore. We wondered: could we install our own newspaper box containing a webcam, the SuperRadio, and a wireless uplink? Wi-Fi access is available, so…..

But maybe all that was unnecessary. After all, we had the Simulcast to work from. And deriving data from the Simulcast seemed like a much easier proposition. We found software that could take snapshots of the simulcast at 10 frames per second, and other software that could process these into a form that clearly could be recognized by image processing software we intended to write.

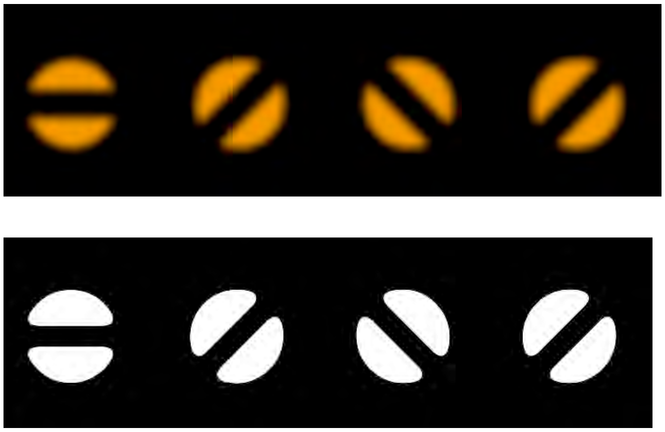


Fig 5: Recognizing the discs. It appeared to us that we could write image recognition software to determine the movement of the discs. The images to the left show a screen captured image along with the result of minimal image processing.

Furthermore, we noticed that the audio track from the simulcast was clean, and consisted of six distinct segments. While it would take a great deal of work, we believed we could automatically recognize these segments.

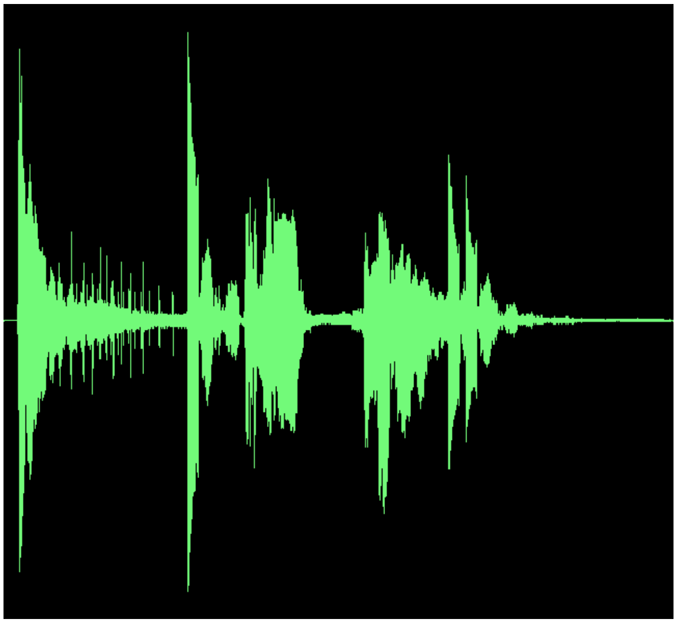
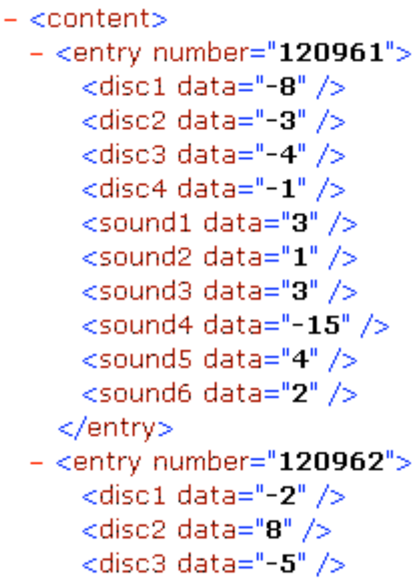


Fig. 6: Recognizing the sounds. Each 7.2 second period of the soundtrack began with a 1.8 second beep with sounds of discs spinning, another 0.6 second beep, a 1.2 second recording of a spoken letter, and a 0.9 second spoken number followed by two beeps. The first of these was 0.2 seconds long and was followed by a 2.5 second period consisting of a beep followed by silence.

Clearly, we had our work cut out for us. While we were confident we could use software to automatically recognize the beeps and words of this audio, it was not something we would be able to do in an evening. Fortunately Mark had a better solution.



```
- <content>
  - <entry number="120961">
      <disc1 data="-8" />
      <disc2 data="-3" />
      <disc3 data="-4" />
      <disc4 data="-1" />
      <sound1 data="3" />
      <sound2 data="1" />
      <sound3 data="3" />
      <sound4 data="-15" />
      <sound5 data="4" />
      <sound6 data="2" />
    </entry>
  - <entry number="120962">
      <disc1 data="-2" />
      <disc2 data="8" />
      <disc3 data="-5" />
```

Fig 7: XML File. After discovering that an XML file like this was available from the Simulcast web site, we abandoned efforts to decode the actual semaphore and the image and sounds of the simulcast. Unless the data did not match the actual semaphore, the file gave us all we needed.

“I found an XML file” was the utterance that changed all of this. As it turns out, the simulcast downloaded an XML file containing the current transmission, and we could reliably fetch this file directly from the Simulcast web site.

Now we were in business. We would not have to plant a fake newspaper box in front of Adobe after all, nor would we have to face up to the resulting interrogation from Homeland Security after they blew up our unexplainable black box with a water cannon. The first of many wrong turns had come to an end.

The Gang of 41

After acquiring a few hours of data, our next task was to find any sort of pattern. We noticed that disc position is expressed as the number of eight’s of a circle rotated, with positive numbers indicating clockwise rotation and negative numbers counterclockwise. The normal range was -8 to +8. But when we plotted these values for one of the discs, we saw a chart shown in Figure 8.

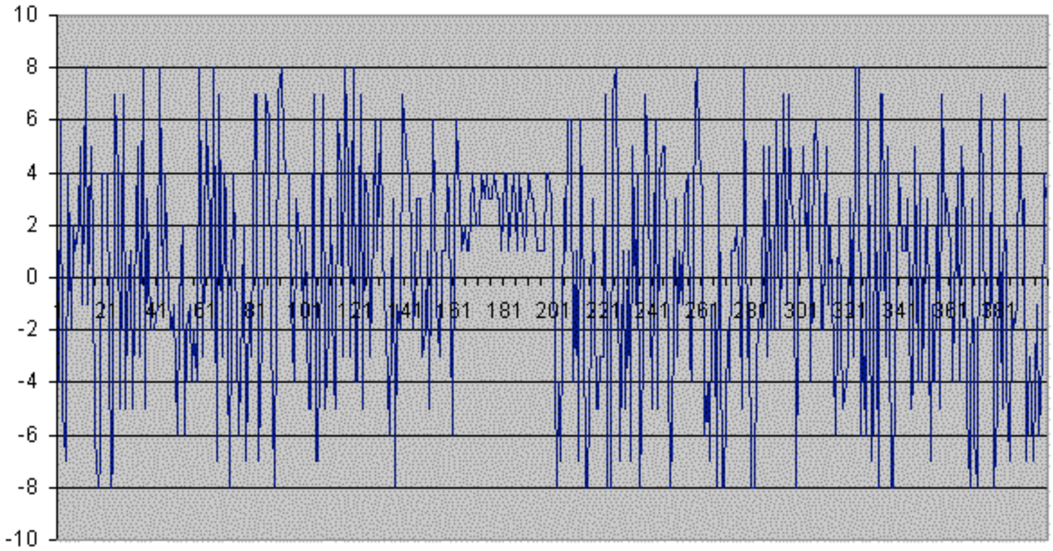
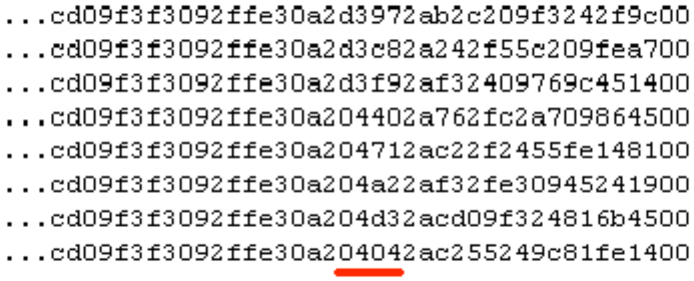


Fig 8: Finding the header. Every so often the discs have a period of restricted movement, always in the range +1 to +4 rather than the usual range -8 to +8. This turned out to indicate an important structure we called a "header".

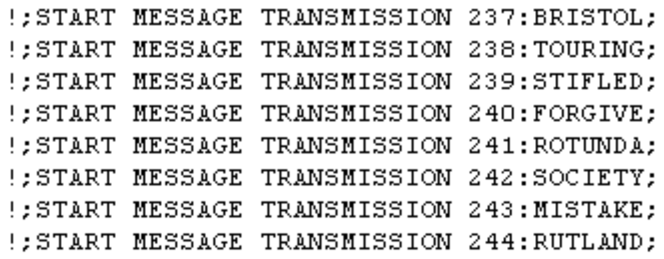
Every so often the discs restricted their movement for 41 time periods. This looked significant to us, so we referred to these sections as headers. A number of attempts were made to find a pattern in the disc movements, but we could not decode a message. After other false starts, we looked at sound3 and sound4 in each header (Figure 9).



```
...cd09f3f3092ffe30a2d3972ab2c209f3242f9c00
...cd09f3f3092ffe30a2d3c82a242f55c209fea700
...cd09f3f3092ffe30a2d3f92af32409769c451400
...cd09f3f3092ffe30a204402a762fc2a709864500
...cd09f3f3092ffe30a204712ac22f2455fe148100
...cd09f3f3092ffe30a204a22af32fe30945241900
...cd09f3f3092ffe30a204d32acd09f324816b4500
...cd09f3f3092ffe30a204042ac255249c81fe1400
```

Fig 9: An incrementing number? Most of a header appeared constant, and other parts of it appeared to change in a way we did not understand. But some places, indicated by the red bar, appeared to behave like an incrementing counter.

We noticed that most of the sound3 and sound4 data was always the same in each header, while other data changed in a way we did not understand. But a small part of the header had the behavior of an incrementing number: the digits changed in a pattern that repeated every 10 headers, and after 10 headers the next position to the left changed to its next pattern. So the natural hypothesis was that this was an incrementing number, and that the 10 patterns saw represented the digits 0 through 9. A character-substitution formula was developed that mapped the data we saw into these digits and, after some work, also mapped the other parts of the header into readable ASCII characters. The result is shown in Figure 10.



```
!;START MESSAGE TRANSMISSION 237:BRISTOL;
!;START MESSAGE TRANSMISSION 238:TOURING;
!;START MESSAGE TRANSMISSION 239:STIFLED;
!;START MESSAGE TRANSMISSION 240:FORGIVE;
!;START MESSAGE TRANSMISSION 241:ROTUNDA;
!;START MESSAGE TRANSMISSION 242:SOCIETY;
!;START MESSAGE TRANSMISSION 243:MISTAKE;
!;START MESSAGE TRANSMISSION 244:RUTLAND;
```

Fig 10: Decoded Headers. After noticing the counter and decoding the digits, we were able to apply the decoding to the entire header. Each header contained a seven-letter keyword, the purpose of which we did not understand.

Clearly we have successfully decoded the header, showing it contained a counter and a seven-letter keyword. After some work, we were also able to demonstrate that the discs and other sounds in the header could be automatically generated from just the keyword, telling us that there was no more information in the header for us to decode.

So, at this point we had headers which contained seven-letter keywords, each of which was followed by a block of data. But what did the keywords mean? Were they random?

Some of the keywords were unusual and not ones that would appear often in normal text. These allowed us to Google for the keywords and discover they were all from the book "Ulysses", by James Joyce. They appeared in the Semaphore's transmission in the same order as the book, but some words in the book were skipped over. We wondered why.

It turns out that certain words in the book are eliminated from the words used as keywords by a set of rules. Words followed by apostrophe "s" are excluded. Some other words were too, but we didn't know why. For a while we entertained the possibility that the message was encoded in the pattern of exclusions, but we could not decode anything from this. Eventually we discovered that words containing the letters H, P, and X are not included. These happen to be the 8th, 16th and 24th letters of the alphabet. Hmmm….what does it all mean?

There's Something in Here

From our experience in generating the entire header from just the keyword, we learned that one step in the process was to use a function of sound3 and sound4 to produce the lower two bits of the discs, and then to set the upper bits to zero. On a hunch, we tried the same function to generate the lower bits of the discs in the data block, too, and found it worked. We didn't know how to compute the upper bits, though.

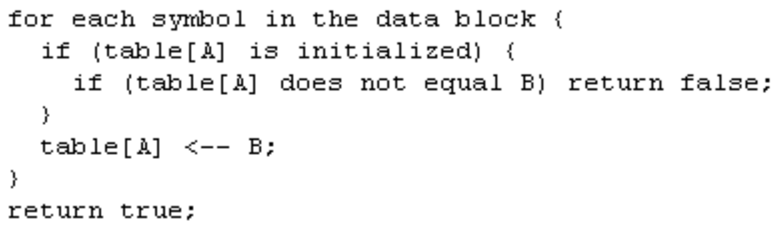
We shifted our attention again to sound3 and sound4. Was there any pattern there? We decided to compute the percentage of (sound3, sound4) pairs that were used versus the number of combinations that were possible. We found that a large portion was used, but not all. However, too many were used to give us much hope that each pair represented a character as in the header, since in that case we would expect to see 26 combinations plus a few for punctuation, or perhaps 52 plus punctuation if upper and lower case were used. Our analyses showed many more than that.

After searching the web for words like "Ulysses cipher", we discovered that Ulysses S. Grant (no relation to the book Ulysses, of course) was well known for the cryptographic work of his team in decoding the confederate cipher. While reading about this, we learned about poly-alphabetic ciphers and the Vigenere cipher. In that cipher, each letter in the keyword is used in rotation to encode the next character in the message. In our case, we knew the keyword for each block and, more importantly, that it was seven letters long.

We decided to take a large block and sort the pairs of sounds into seven bins in rotation, with the first pair of the block going into bin 0, the second pair into bin 1, etc. According to the rotation, the eighth pair went into bin 0, and so on. In this way, each bin would correspond to one letter of the keyword of a Vigenere cipher. If there was a function that encoded each character according to the associated keyword letter, in rotation ala the Vigenere cipher, each bin should contain a small number of pairs, corresponding to the alphabet (26 or 52 plus punctuation). This was not the case, however. Dead end.

Around this time, it was discovered that the last pair of sounds in the block was special, in that a simple function would always turn it into an alphabetic character, and furthermore that character was contained in the block's keyword. That could not be just random, and gave us more reason to believe that somehow each pair of (sound3, sound4) pairs in the block was somehow created by the use of the corresponding keyword character.

Perhaps the discs and the minor sounds (sound1, sound2, sound5, and sound6) somehow came into play? In the header they were redundant, but we didn't know if they were or not for the data blocks. We decided to do a large number of experiments to see if a particular set of data (such as sound1) could be used to predict another set of data (such as bit 2 of disc 3). To do this, we wrote a function that checked for consistency. For instance, figure 11 shows code that tries to construct a table that takes a value A and predicts the value of B. However, if it finds a value A that maps into two different values of B, then we know that A cannot be used to predict B.



```
for each symbol in the data block {
  if (table[A] is initialized) {
    if (table[A] does not equal B) return false;
  }
  table[A] <-- B;
}
return true;
```

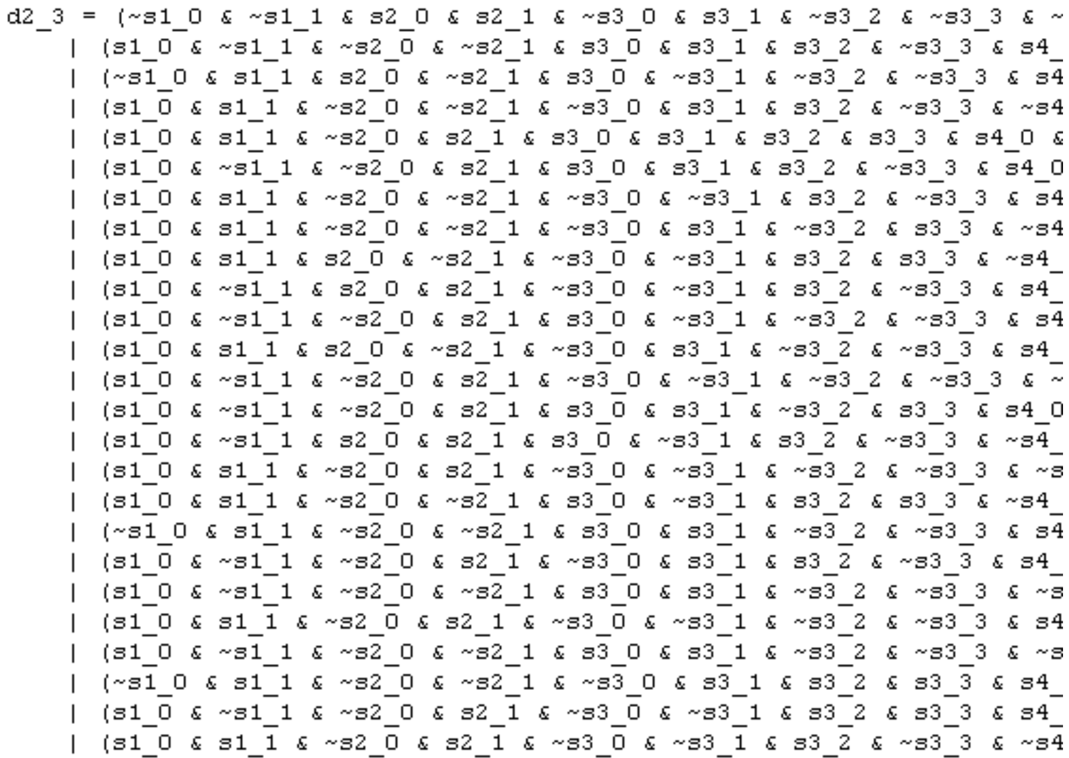
Fig 11: Predicting Data. We weren't sure if the data in the discs or minor sounds was important, so we used a function like this to see if we could compute them from other data, showing they were redundant.

After many false results, we discovered that the sign of disc 2 (which indicates either clockwise or counterclockwise rotation) was predicted by a combination of sound1 and sound2. By manual inspection, we discovered the function and found it used the upper bit of sound1 and both bits of sound2: "`sign = a_1 ^ ((a_1 ^ ~b_1) & (a_1 ^ ~b_0))`". A similar function predicted the sign of disc3 from sound5 and sound6.

Further work on decoding the sound3-sound4 pairs had not lead anywhere, so we decided to continue work finding bits in the discs that might be significant. We focused on bit 2 of disc3, and found that the previous values of the discs (from the previous time period) along with the current sounds could be used to predict it. After much experimentation, however, could not find the function. We knew that this problem was similar to the logic synthesis problem found in designing microprocessors, so we decided to apply some of those techniques.

We downloaded a classic logic optimization tool, called Espresso, to see if it could do better. To use as input to the tool, our software produced a table showing the value of bit 2 of disc3 for each combination of previous discs and current sounds. We hoped that Espresso would be able to grind away on this data and present us with a simple formula, similar perhaps to the one we found for the sign. With only an underpowered laptop available for this purpose, we had our result in 14 hours.

There was indeed such an equation. Part of it is shown in Figure 12. It was 4,142 pages long. This is not the sort of equation we would expect the semaphore designer to choose. Furthermore, its length indicated it probably was more like a restatement of the data seen so far, rather than a succinct way to compute bit 2 of disc3. To test this hypothesis, we substituted a random number for bit 2 of disc3 in our data, and re-ran our process. The result took 36 hours to compute this time, but produced an equation of similar size. We don't know what to make of this, but our hypothesis was that bit 2 of disc3 is either random, produced by a good-quality hash function, or produced by a number of exclusive-ors (which are not handled well by Espresso). Or else it contained real data. We couldn't tell, and this was another dead end of sorts.



```
d2_3 = (~s1_0 & ~s1_1 & s2_0 & s2_1 & ~s3_0 & s3_1 & ~s3_2 & ~s3_3 & ~
     | (s1_0 & ~s1_1 & ~s2_0 & ~s2_1 & s3_0 & s3_1 & s3_2 & ~s3_3 & s4_
     | (~s1_0 & s1_1 & s2_0 & ~s2_1 & s3_0 & ~s3_1 & ~s3_2 & ~s3_3 & s4
     | (s1_0 & s1_1 & ~s2_0 & ~s2_1 & ~s3_0 & s3_1 & s3_2 & ~s3_3 & ~s4
     | (s1_0 & s1_1 & ~s2_0 & s2_1 & s3_0 & s3_1 & s3_2 & s3_3 & s4_0 &
     | (s1_0 & ~s1_1 & ~s2_0 & s2_1 & s3_0 & s3_1 & s3_2 & ~s3_3 & s4_0
     | (s1_0 & s1_1 & ~s2_0 & ~s2_1 & ~s3_0 & ~s3_1 & s3_2 & ~s3_3 & s4
     | (s1_0 & s1_1 & ~s2_0 & ~s2_1 & ~s3_0 & s3_1 & ~s3_2 & s3_3 & ~s4
     | (s1_0 & s1_1 & s2_0 & ~s2_1 & ~s3_0 & ~s3_1 & s3_2 & s3_3 & ~s4_
     | (s1_0 & ~s1_1 & s2_0 & s2_1 & ~s3_0 & ~s3_1 & s3_2 & ~s3_3 & s4_
     | (s1_0 & ~s1_1 & ~s2_0 & s2_1 & s3_0 & ~s3_1 & ~s3_2 & ~s3_3 & s4
     | (s1_0 & s1_1 & s2_0 & ~s2_1 & ~s3_0 & s3_1 & ~s3_2 & ~s3_3 & s4_
     | (s1_0 & ~s1_1 & ~s2_0 & s2_1 & ~s3_0 & ~s3_1 & ~s3_2 & ~s3_3 & ~
     | (s1_0 & ~s1_1 & ~s2_0 & s2_1 & s3_0 & s3_1 & ~s3_2 & s3_3 & s4_0
     | (s1_0 & ~s1_1 & s2_0 & s2_1 & s3_0 & ~s3_1 & s3_2 & ~s3_3 & ~s4_
     | (s1_0 & s1_1 & ~s2_0 & s2_1 & ~s3_0 & ~s3_1 & ~s3_2 & ~s3_3 & ~s
     | (s1_0 & s1_1 & ~s2_0 & ~s2_1 & s3_0 & ~s3_1 & s3_2 & s3_3 & ~s4_
     | (~s1_0 & s1_1 & ~s2_0 & ~s2_1 & s3_0 & s3_1 & ~s3_2 & ~s3_3 & s4
     | (s1_0 & ~s1_1 & ~s2_0 & s2_1 & ~s3_0 & s3_1 & s3_2 & ~s3_3 & s4_
     | (s1_0 & ~s1_1 & ~s2_0 & ~s2_1 & s3_0 & s3_1 & ~s3_2 & ~s3_3 & ~s
     | (s1_0 & s1_1 & ~s2_0 & s2_1 & ~s3_0 & ~s3_1 & ~s3_2 & ~s3_3 & s4
     | (s1_0 & ~s1_1 & ~s2_0 & ~s2_1 & s3_0 & s3_1 & ~s3_2 & ~s3_3 & ~s
     | (~s1_0 & s1_1 & ~s2_0 & ~s2_1 & ~s3_0 & s3_1 & s3_2 & s3_3 & s4_
     | (s1_0 & ~s1_1 & ~s2_0 & s2_1 & ~s3_0 & ~s3_1 & s3_2 & s3_3 & s4_
     | (s1_0 & s1_1 & ~s2_0 & s2_1 & ~s3_0 & ~s3_1 & s3_2 & ~s3_3 & ~s4
```

Fig 12: Predicting bit 2 of disc3. This bit is indeed predicted by the previous discs and the sounds, but only with an absurdly large equation. To show this equation in its entirety it would require 4,142 pages. We doubt Ben Rubin, the designer of the semaphore, would have written an equation that long.

Numerous other ideas were investigated which led nowhere. No new patterns could be found in the discs or in sounds 1, 2, 5 or 6. Eventually we turned our focus back to the (sound3, sound4) pairs.

The Final Leg

Once we had collected over two weeks of the transmission data, it became possible to perform frequency analysis on the (sound3, sound4) pairs. We made a table of the number of occurrences of each pair in the data block portion of the transmission. A couple of things were immediately obvious.

First, the sign of sound4 did not affect the frequency. This led us to ignore the sign in subsequent frequency analysis. Second, certain pairs were much more frequent than other pairs. We spent a lot of time trying to discover a pattern which would explain why some pairs were very frequent and some pairs never occurred. Unfortunately we were unsuccessful.

A breakthrough happened when we decided to do frequency analysis on each data block independently. Most of the data blocks were small and therefore hard to obtain any meaningful information from. There were however a few larger blocks with 2000 or more lines of data. These blocks provided important clues.

Each data block seemed to have its own 'signature'. Certain pairs were very common in one block, but were completely absent in another block. We spent time looking for patterns in the individual blocks, but again that was unsuccessful.

	sound3															
sound4	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	55	0	0	5	0	1	0	15	0	0	0	0	20	0	1	0
2	2	0	28	39	0	0	0	39	0	16	5	3	0	5	0	0
3	53	1	2	20	0	0	0	0	0	3	10	0	4	0	0	0
4	5	98	29	29	13	0	0	1	17	0	0	0	0	23	51	7
5	0	0	11	21	0	0	0	13	0	9	2	0	0	0	0	0
6	66	0	38	5	0	38	23	27	0	1	0	7	0	0	26	14
7	0	0	11	3	0	0	14	0	9	0	15	0	0	0	0	0
8	1	1	5	29	23	5	7	8	2	5	25	0	0	23	0	48
9	55	0	14	2	4	0	3	0	0	17	0	7	0	0	0	0
10	3	21	22	41	1	3	0	17	5	0	0	0	19	0	8	0
11	0	23	9	4	0	0	0	14	0	0	0	0	19	0	0	0
12	23	8	5	0	21	3	7	0	1	17	17	4	16	21	0	38
13	11	0	10	0	13	0	0	0	21	0	0	0	0	22	0	19
14	65	1	5	38	3	0	0	16	0	5	9	0	0	10	0	20
15	58	0	16	0	0	7	11	0	0	22	0	0	0	4	2	0
16	21	32	26	0	5	4	1	0	23	22	22	21	30	0	7	0

Fig 13: Signature for keyword FACTION. This table shows the number of occurrences of each (sound3, sound4) combination in data block following the header with keyword FACTION. Different keywords have tables that may look very different. For example, the table for keyword BUILDER has 56 occurrences of (2, 2) compared with 0 for FACTION, and 0 occurrences of (1, 3) compared with 53 for FACTION.

We had an idea that the 7-letter keywords in the header were related to the data, and eventually we were able to confirm it. By comparing two data blocks whose keywords were similar, we noticed a high correlation of the frequency pairs in each one. For example, the keywords LEONARD and SNARLED or RUTLAND and ROTUNDA share 6 out of 7 letters. The frequency pattern of the pairs for these blocks are very similar.

However, ROTUNDA and WIMBLES share 0 letters, and their frequency patterns are very different. So we now knew that the letters in the keyword affected the (sound3, sound4) values. By looking at blocks whose keywords differed by only a single letter, we tried to identify which pairs seemed to be affected by each letter of the alphabet. Unfortunately this was not possible since we had not captured enough large data blocks and the smaller data blocks didn't contain enough pairs to be meaningful.

We finally made a breakthrough when we developed a new way to present the frequency analysis data. We threw out the smallest data blocks, those with less than 750 entries. For the medium and large blocks, we normalized the frequency data to make the frequency of each pair independent of the total number of entries in the data block.

For each (sound3, sound4) pair, we listed the letters in the keyword along with the normalized frequency of that pair. When we sorted each list based on frequency, some interesting patterns appeared. All the keywords which contained a certain letter would be at the top of the list. Then the words which contained another letter would be next on the list. For example, all the keywords at the top of the list for pair (1, 4) had the letter D in them. The next keywords in that list all had the letter E. And the keywords which contained both D and E were listed at the very top.

Letters in the keyword	Keyword	Normalized frequency of pattern (1,4)
DE G NO TU	TONGUED	88
A DE I N R T	ARDENTI	87
A CDE O R T	REDCOAT	85
A DE N TUV	VAUNTED	84
DE G N R TU	GRUNTED	79
AB DE G NO	BONDAGE	75
CDE I N RS	DISCERN	74
A DE L N RS	SNARLED	71
B DE I L R U	BUILDER	70
CDE I O R V	DIVORCE	69
A CDE L O R	ORACLED	69
A D I L RS Z	LIZARDS	60
A D I LMN U	MAUDLIN	59
D I MNO S U	DOMINUS	58
AB D G I N W	WINDBAG	53
AB E I L ST	BASTILE	25
A E I N Q TU	ANTIQUE	24
E I LM O RS	LISMORE	20
A E M RST Y	STREAMY	20
A E J L N UV	JUVENAL	19
E I K N RS W	WINKERS	18
B E G L NO S	BELONGS	18
C E I NO ST	NOTICES	17
E IJKL R Y	JERKILY	17
B E I K RST	BRISKET	17
A C E LM TU	CALUMET	15
B EF I L R Y	BRIEFLY	15
A E J MNO S	NOSEJAM	15
B E I L O Q U	OBLIQUE	15
A E G O S V Y	VOYAGES	14
C E I KL N R	CLINKER	14
B E I KL R U	BULKIER	10
ABC G O ST	COSTBAG	6
A C F I NO T	FACTION	5
A L O QRS U	SQUALOR	2
G I NO RS U	ROUSING	0

Fig 14: Frequency Table for pattern (1, 4). The table for pattern (1, 4) shows that keywords with a D in them have the most occurrences of this pattern, followed by keywords with an E. Keywords with both D and E have even more.

We learned which pairs seemed to be a function of which keyword letters, but still didn't know what it meant. Also, for many of the pairs it wasn't clear which letters really were at the top of the list. We knew we had to come up with some sort of mapping function to explain the data. Based on the progress we had made so far, we expected that mapping function to be fairly straight-forward and elegant.

Eventually we noticed that aside from (1, 1), all of the pairs which seemed to be a function of the letter A were in two columns, (3, X) or (4, X). After looking at it for a while we figured that each pair in these columns represented the encoding of a different letter of the alphabet with the letter A.

We discovered an encoding which was simple and fit the data well, based on our knowledge of the frequency of certain letters of the alphabet in most English writing. For instance, the letters E, T, and A occur most often. The mapping we came up with for the letter A was:

```
(3, 2) = A
(3, 3) = B
(3, 4) = C
 ...
(3, 16) = O
(4, 1) = P
(4, 2) = Q
...
(4, 10) = Z
```

We then tried to come up with a mapping for the letter B, but it was difficult since few of the keywords had the letter B. But eventually we concluded that this was the correct mapping:

```
(5, 4) = A
(5, 6) = B
(5, 8) = C
...
(5, 16) = G
(6, 2) = H
(6, 4) = I
...
(8, 6) = Z
```

From there we were able to come up with a simple mapping of the letters of the alphabet for each of the possible keyword letters. We found that if we assumed that the letters were ASCII encoded our mapping was also able to handle punctuation and spaces.

```
L = keyword letter (ASCII code)
C = character to encode (ASCII code)
x = (L-64) * (C-32+1) - 1
sound3 = ((x >> 4) & 0xF) + 1
abs(sound4) = (x & 0xF) + 1
```

Using this mapping, we listed the possible ASCII characters for the entries in each data block based on the 7 letters of the keyword for that block. Even though we still didn't know which keyword letter to use on each entry, by examining the ASCII characters we could easily see sentences form in the data blocks.

Keyword	(Sound3, sound4)	Possible decoded chars	Possible encoding letters	decoded char
FACTION	4,-14:	"T]H"	"FAN"	T
FACTION	3, 9:	"H@"	"AI"	H
FACTION	8, 2:	"2E^"	"FCN"	E
FACTION	2, 4:	"M3 E"	"FATN"	
FACTION	7, -9:	"B&"	"CO"	B
FACTION	16, 8:	"EWC"	"TIN"	E
FACTION	8, 8:	"3G%'"	"FCTO"	G
FACTION	5, -8:	"+7I'W["	"FCTION"	I
FACTION	11, 7:	"N"	"I"	N
FACTION	13, -1:	"N"	"O"	N
FACTION	5,-12:	"VI"	"TN"	I
FACTION	2,-10:	"N9"	"FA"	N
FACTION	3, 8:	"G!"	"AT"	G
FACTION	1, 3:	"" "	"AC"	
FACTION	12, 16:	"?_O_?"	"FCTON"	O
FACTION	6,-15:	"F"	"I"	F
FACTION	1, 15:	".$ "	"ACO"	
FACTION	14, 13:	"T"	"I"	T
FACTION	4, -4:	"SH5"	"ATN"	H
FACTION	7, -6:	"0AY"	"FCO"	A
FACTION	4, -5:	"T"	"A"	T
FACTION	1, 6:	" %!D"	"FACN"	
FACTION	4, 1:	"P"	"A"	P
FACTION	15, -4:	"ED"	"FT"	E
FACTION	13,-10:	"R"	"N"	R
FACTION	5, 9:	"F"	"O"	F
FACTION	11, 16:	"[O"	"TI"	O
FACTION	16,-13:	"R"	"O"	R
FACTION	9, -4:	"5KL[M"	"FCTON"	M
FACTION	13,-12:	"A6"	"FT"	A
FACTION	3,-15:	"N"	"A"	N
FACTION	7, 12:	"1C>+9"	"FCTIN"	C
FACTION	3, 6:	"PE4"	"FAN"	E

Fig 15: Manual decoding. This figure shows how the manual decoding of the first decoded sentence was performed. The (sound3, sound4) pairs from the transmission were decoded with each of the 7 letters in the keyword. Some of the letters did not decode to an ASCII character and were discarded. By looking at the possible decoded characters column, it is fairly easy to identify the decoded message as “THE BEGINNING OF THAT PERFORMANCE”.

The first sentence we decoded was "THE BEGINNING OF THAT PERFORMANCE WAS CLEAR ENOUGH." An internet search of that brought up the text of Thomas Pynchon's "The Crying of Lot 49". We decoded other sentences to confirm that the decoded message was the text of that book. We had finally decoded the San Jose Semaphore!

Even though we had figured out the message, there were still a few loose ends to clean up. First, we still didn't know which keyword letter to use for encoding each character of the text. After some investigation we found out that the keyword letters go sequentially through the keyword, but the direction changed often.

For instance, if the keyword is FACTION, the order of the letters used to encode each character of the message may be:
FACTCTCTIONFACTIOITCAFAFNOI

Further investigation showed that the direction changes whenever sound5 > sound 6. We were now able to automatically decode the entire message, and it was indeed the entire text of "The Crying of Lot 49". We also noticed that each data block contained one paragraph of the book, which explained why some blocks were very short and others were quite long.

Conclusions

Decoding the Semaphore was an interesting challenge, and fun, too. The aesthetics of the art itself made it much more interesting than a standard puzzle, and we hope you get a chance to see the Semaphore in action in its San Jose location or in the Simulcast at www.sanjosesemaphore.org.