The Golay Codes: NASA, Quantum, an Interactive Web Application, and More

Ethan Balcik, Sergio Hernandez-Fierro, and Rian Nihart

December 9, 2021

1 Introduction

In today's world, information is everything. Every day, human activity becomes more digitized and more distributed. Today, a social interaction, or perhaps a financial transation, is nothing more than an exchange of bits over the world wide web. With this in mind, imagine the chaos that would ensue if in every social interaction or financial transaction that takes place over today's modern internet, just one bit was flipped. If this was the case, sending the text message *have fun* could be received as, *have gun*, or a payment of 5 dollars could be received as a payment of 32773 dollars! Simply put, the more digitized the world becomes, the more the world had better equip itself with the necessary techniques to protect against such (potentially catastrophic!) errors.

As it turns out, techniques to protect information against noise have been extensively researched since as early as the 1940s. Claude E. Shannon, in his 1948 paper *A Mathematical Theory of Communication* and Richard W. Hamming, in his 1950 paper *Error Detecting and Error Correcting Codes*, together provided the theoretical framework atop which research into structures with error correction and error detection properties could take place. On one hand, Shannon quantified the notion of an information source as well as a communication channel, eventually proving in his *Fundamental Theorem for a Discrete Channel with Noise* that it is possible to send information at the channel capacity of a communication channel with as small a frequency of errors as desired provided the proper encoding scheme [1]. On the other, Hamming introduced the notions of error correction and error detection as they are studied in modern coding theory, introduced the family of binary Hamming codes, and even related the act of protecting information against noise to the problem of sphere packing in higher dimensions [2].

Ultimately, when discussing the development of coding theory as we know it today, the two aforementioned foundational papers are likely the first and only to come to mind for many. However, in the final remarks of Hamming in *Error Detecting and Error Correcting Codes*, Hamming referred to a work which seemed to be the only work in the field of error correction that preceded that of his own. This work to which Hamming alluded was the work which, 25 years later, the late coding theory great Elwyn R. Berlekamp deemed the best single published page in coding theory [3]. This work, entitled *Notes on Digital Coding* by Marcel J. E. Golay, introduced the perfect binary code which has since come to be known as the Golay code [4].

In this report, we detail the Golay code in all of its theoretical glory, as well as some of the instances in which the Golay code has been utilized practically. We begin in the following section by covering necessary background related to coding theory in general, and also related to the geometric structures to which the Golay code is closely related. We then discuss the constructions, properties, and decoding algorithm for the Golay code, after which we discuss its practical use. As an added bonus, we detail our own creation - an interactive web application using which users can interact with the Golay code and further understand its error correction properties, as well as its use practically.

2 Technical Background

2.1 Algebraic, Coding, and Geometric Background

The following definitions are indented to provide the reader with a sufficient understanding of algebraic and coding theoretic terminology before discussing the various constructions of the Golay code, and their respective properties. If the reader is equipped with the following background already, they are encouraged to advance to section 2.2. To understand the following background, the reader is expected to have an understanding of set theoretic concepts, notation, and perhaps some basic understanding of matrices and abstract algebra.

We begin by defining various abstract algebraic structures. Such objects are used to introduce algebraic structure atop sets, and they are useful in coding theory as they provide the elementary tools to generate and to analyze a class of codes known as linear codes. Each of the following algebraic structures are generated by introducing further structure on those defined previously.

Definition 2.1.1: A **group** $(\mathcal{G}, \circ)$ is a set $\mathcal{G}$ on which a binary operation is defined satisfying the following properties.

1. For all $a, b \in \mathcal{G}$, $a \circ b \in S$ (closure under $\circ$)
2. For all $a, b, c \in \mathcal{G}$, $(a \circ b) \circ c = a \circ (b \circ c)$ (associativity of $\circ$)

3. There exists some $e \in \mathcal{G}$ such that for all $a \in \mathcal{G}$, $a \circ e = e \circ a = a$ (existence of an identity element)

4. For all $a \in \mathcal{G}$, there exists some $a^{-1} \in \mathcal{G}$ such that $a \circ a^{-1} = a^{-1} \circ a = e$ (existence of an inverse)

Definition 2.1.2: A group $(\mathcal{G}, \circ)$ is called **abelian**, or **commutative**, if for all $a, b \in \mathcal{G}$, $a \circ b = b \circ a$.

Definition 2.1.3: A **ring** $(\mathcal{R}, +, \circ)$ is a set $\mathcal{R}$ on which two binary operations ($+$ and $\circ$) are defined satisfying the following properties.

1. $(\mathcal{R}, +)$ is an abelian group with the identity element denoted as 0

2. For all $a, b, c \in \mathcal{R}$, $(a \circ b) \circ c = a \circ (b \circ c)$ (associativity of $\circ$)

3. For all $a, b, c \in \mathcal{R}$, $c \circ (a + b) = c \circ a + c \circ b$ and $(a + b) \circ c = a \circ c + b \circ c$ (distributivity of $\circ$ over $+$)

Definition 2.1.4: A **field** $(\mathcal{F}, +, \circ)$ is a set $\mathcal{F}$ on which two binary operations ($+$ and $\circ$) are defined such that $(\mathcal{F}, +, \circ)$ is a ring, and $(\mathcal{F} \setminus \{0\}, \circ)$ is an abelian group with the identity element denoted as 1.

Definition 2.1.5: A **vector space** $(\mathcal{V}, +, \mathcal{F})$ is a 3-tuple such that $(\mathcal{V}, +)$ is an abelian group, $\mathcal{F}$ is a field, and scalar multiplication $\mathcal{F} \times \mathcal{V} \to \mathcal{V}$ is defined satisfying the following properties [5].

1. For all $a \in \mathcal{V}$, $1\vec{a} = \vec{a}$

2. For all $\alpha, \beta \in \mathcal{F}$ and for all $\vec{a} \in \mathcal{V}$, $\alpha(\beta\vec{a}) = (\alpha\beta)\vec{a}$

3. For all $\alpha, \beta \in \mathcal{F}$ and for all $\vec{a} \in \mathcal{V}$, $(\alpha + \beta)\vec{a} = \alpha\vec{a} + \beta\vec{a}$

4. For all $\alpha \in \mathcal{F}$ and for all $\vec{a}, \vec{b} \in \mathcal{V}$, $\alpha(\vec{a} + \vec{b}) = \alpha\vec{a} + \alpha\vec{b}$

After having defined the algebraic structure with which much of coding theory is concerned - the vector space - we drill down and investigate the algebra defined over vector spaces. Namely, the following definitions fall under the umbrella of what is known commonly as linear algebra.

Definition 2.1.6: A **linear subspace** $\mathcal{U} \subseteq \mathcal{V}$ is a subset of a vector space which, itself, forms a vector space.

Definition 2.1.7: Let $\mathcal{F}$ be a field. A **linear combination** of variables $x_1, \ldots, x_n \in \mathcal{F}$ may be expressed as $a_1x_1 + \cdots + a_nx_n$ where each $a_i \in \mathcal{F}$ is the coefficient of x_i. A **linear equation** in the variables $x_1, \ldots, x_n \in \mathcal{F}$ equates a linear combination

$a_1x_1+\cdots+a_nx_n$ to some constant $b \in \mathcal{F}$, and may be written as $a_1x_1+\cdots+a_nx_n = b$.

Definition 2.1.8: An n by m **matrix** is a rectangular array of numbers $a_{ij} \in \mathcal{F}$ with n rows and m columns. A **row vector** $\vec{v} \in \mathcal{V}$ is an 1 by m matrix,

$$\vec{v} = \begin{bmatrix} a_1 & a_2 & \dots & a_m \end{bmatrix}$$

Definition 2.1.9: The m by m square matrix with the values along its diagonal a_{ii} being the multiplicative identity $1 \in \mathcal{F}$ for $1 \le i \le m$, and with the rest of its values being the additive identity $0 \in \mathcal{F}$, is called the **identity matrix** I.

Definition 2.1.10: Let A be an m by n matrix and B be an n by p matrix. Then the **matrix product** $C = AB$ is an m by p matrix such that the entry in the ith row and the jth column of C is given as $c_{ij} = \sum\limits_{k=1}^{n} a_{ik}b_{kj}$.

Definition 2.1.11: Let $\mathcal{V}$ and $\mathcal{W}$ be vector spaces defined on the field $\mathcal{F}$. A function $f : \mathcal{V} \to \mathcal{W}$ is said to be a **linear map** if for any $\vec{u}, \vec{v} \in \mathcal{V}$ and any scalar $c \in \mathcal{F}$ we have that $f(\vec{u}+\vec{v}) = f(\vec{u}) + f(\vec{v})$, and $f(c\vec{u}) = cf(\vec{u})$.

Note that we may define a linear map $f : \mathcal{V} \to \mathcal{W}$ by a matrix multiplication of the form $f = A\vec{x}$ (or $f = \vec{x}A$) where A is a matrix with entries belonging the field $\mathcal{F}$ and $\vec{x}$ is a vector belonging to the vector space $\mathcal{V}$. It is assumed that the product of such multiplication is a vector belonging to the vector space $\mathcal{W}$. A is called the **standard matrix** for the linear map f.

Definition 2.1.12: A set of vectors $\{\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n\}$ belonging to the vector space $\mathcal{V}$ defined over the field $\mathcal{F}$ is **linearly independent** if for all scalars $a_1, a_2, \dots, a_n \in \mathcal{F}$ such that there exists some scalar $a_i \neq 0$, we have that $a_1\vec{v}_1 + a_2\vec{v}_2 + \cdots + a_n\vec{v}_n \neq 0$.

Definition 2.1.13: The **span** of a set of vectors $\{\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n\}$ is the set of all vectors which can be written as a linear combination $a_1\vec{v}_1 + a_2\vec{v}_2 + \cdots + a_n\vec{v}_n$.

Definition 2.1.14: A **basis** for the vector space $\mathcal{V}$ is a linearly independent set of vectors $\{\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n\}$ in $\mathcal{V}$ for which the vectors $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n$ span $\mathcal{V}$.

An important result in linear algebra states that all bases of a vector space share the same cardinality. This result motivates the following definition.

Definition 2.1.15: Let the set $\mathcal{B}$ be a basis for the vector space $\mathcal{V}$. The **dimension** $dim(\mathcal{V}) = |\mathcal{B}|$ is the cardinality of any basis for $\mathcal{V}$.

This concludes the discussion of purely linear algebraic concepts [6], however these linear algebraic definitions provide an elementary understanding of the tools used throughout coding theory. As mentioned, much of coding theory involves the use of linear algebra. To elaborate on this, much of coding theory may be thought of quite simply as the practice of finding linear subspaces with good properties. The properties by which we gauge how "good" a linear code is are explored and built upon in the following definitions.

Definition 2.1.16: The **weight** $wt(\vec{v})$ of a vector $\vec{v} \in \mathcal{V}$ is the cardinality of the set of nonzero elements in $\vec{v}$, given formally as follows.

$$wt(\vec{v}) = |\{a_i \in \vec{v} : a_i \neq 0\}|$$

Definition 2.1.17: The **Hamming distance** $dist(\vec{u}, \vec{v})$ between two vectors $\vec{u}, \vec{v} \in \mathcal{V}$ is the cardinality of the set of indices over which the elements of $\vec{u}$ and $\vec{v}$ differ, given formally as follows.

$$dist(\vec{u}, \vec{v}) = |\{i : \vec{u}_i \neq \vec{v}_i, 0 \leq i < n\}|$$

Definition 2.1.18: Given a set of vectors $U = \{\vec{u}_1, \vec{u}_2, \ldots, \vec{u}_k\}$ such that $U \subseteq \mathcal{V}$, the **minimum distance** d of the set U is given as,

$$d = \min_{\vec{u}_i, \vec{u}_j \in U} dist(\vec{u}_i, \vec{u}_j) \text{ such that } \vec{u}_i \neq \vec{u}_j$$

Definition 2.1.19: An $[n, k, d]$ q**-ary linear code** is a linear subspace $\mathcal{C} \subseteq \mathcal{V}$ defined over the field $\mathbb{F}_q = \{0, 1, \ldots, q-1\}$ containing vectors of length n such that $dim(\mathcal{C}) = k$, and the minimum distance of $\mathcal{C}$ is exactly d.

Definition 2.1.20: A **generator matrix** for a q-ary linear code $\mathcal{C}$ is a matrix whose rows form a basis for $\mathcal{C}$.

Definition 2.1.21: A **parity-check matrix** for a q-ary linear code $\mathcal{C}$ is the transpose of the generator matrix of $\mathcal{C}$.

Definition 2.1.22: The **dual code** $\mathcal{C}^{\perp}$ of a linear code $\mathcal{C}$ is defined such that for any $u \in \mathcal{C}$ and $v \in \mathcal{C}^{\perp}$ we have $u \cdot v = 0$. The generator matrix for $\mathcal{C}^{\perp}$ is the parity-check matrix for $\mathcal{C}$, and vice versa. A code $\mathcal{C}$ is **self-dual** if $\mathcal{C} = \mathcal{C}^{\perp}$.

As is discussed in the following chapter which details the practical use of a particularly "good" code, codes are generally used for the purpose of **error correction**. We begin with k q-ary digits of information, to which we append $n-k$ redundant q-ary digits of information to protect the information from noise according to some linear map, yielding a linear subspace $\mathcal{C}$. The resulting **code words** are vectors in the subspace, each having some distance between one another, the minimum distance between any two code words being d. If we send a code word through a noisy channel, there is a probability that that code word will arrive with some of its q-ary digits different from those which were sent due to the noise in the channel causing errors.

To recover the sent code word from that which was received, we can reliably consider the closest code word (in terms of Hamming distance) to that which was received. This is due to the multiplicative nature of probability in conjunction with the minimum distance of the code d; the code word which was sent is most likely the closest code word to the vector which was received in most practical situations as a large number of errors occurring is exponentially less likely than, say, 1 or 2 occurring. Thus, even in the face of a certain number of errors, we may retrieve the sent code word with a certain degree of accuracy. This is known as **decoding**. For larger minimum distance d, we have more accurate, or error-tolerant, decoding

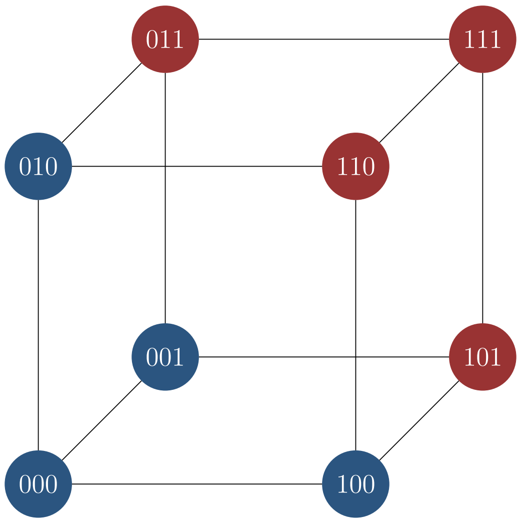
The dimensions along which codes are deemed "good" are the $[n, k, d]$ parameters described by definition 2.1.19. A code with a large k can encode much information, however this limits the capacity of the code to correct errors if we fix n. Furthermore, a code word with small n is typically faster to transmit, but small n limits the sizes of k and d. Such tradeoffs illustrate the complexity of the engineering problem that is finding "good" codes. We present the following definitions which establish some basic understanding of how coding theorists confront this problem.

Definition 2.1.23: The **sphere packing bound** is an upper bound on the number of code words in a q-ary linear code with a given length and minimum distance. Any q-ary $[n, k, 2t + 1]$ code satisfies the bound,

$$q^n \geq q^k \sum_{i=0}^{t} \binom{n}{i} (q-1)^i$$

Definition 2.1.24: A **perfect code** is a q-ary linear code which achieves the sphere packing bound.

Clearly, a perfect code will be, in some sense, optimal with regard to its dimension k. Furthermore, although it may not be immediately obvious, perfect codes hold some geometric significance. In particular, we may rephrase the above definition geometrically in the following way. Let the **ball of radius** r about a code word $\vec{c} \in \mathcal{C} \subset \mathcal{V}$ be defined as the number of vectors $\vec{v} \in \mathcal{V}$ such that the Hamming distance $dist(\vec{c}, \vec{v}) \leq r$. The **volume** of this ball is given by the summation from definition 2.1.23. Then a code which corrects $2t + 1$ errors is composed of balls of radius t about the code words $\vec{c} \in \mathcal{C}$ packed into the vector space $\mathcal{V}$. A perfect code is then a code that does so in such a way that each ball is packed into $\mathcal{V}$ with no overlap *and* such that the entire space is saturated (i.e. no vector $\vec{v} \in \mathcal{V}$ lies outside the balls).



The repetition code in $\mathbb{F}_2^3$ yields a perfect sphere packing.

Such analogous approaches to defining "good" codes are certainly part of what make coding theory so interesting. Beyond this geometric picture, there are many more interesting connections that can be made between coding and various other concepts that give rise to interesting codes. In the following definitions, we explore the class of codes known as cyclic codes which utilizes the space of polynomials over $\mathbb{F}_q$ mod $x^n - 1$ to generate "good" codes.

Definition 2.1.25: A linear code $\mathcal{C}$ is called a **cyclic code** if for all $\vec{c} \in \mathcal{C}$ such that $\vec{c} = [c_0, c_1, \dots c_{n-1}]$, there exists some $\vec{c'} \in \mathcal{C}$ such that $\vec{c'} = [c_{n-1}, c_0, c_1, \dots, c_{n-2}]$.

Thus, if we have a code word $\vec{c} \in \mathcal{C}$, then we may obtain more code words in $\mathcal{C}$ for every unique cyclic shift of $\vec{c}$.

To further understand cyclic codes, we analyze the isomorphism between $\mathbb{F}_q^n$ considered as a group equipped with addition and the ring of polynomials over $\mathbb{F}_q$ mod $x^n - 1$, given as $\mathbb{F}_q[x]/(x^n - 1)$.

Definition 2.1.26: An **irreducible polynomial** over $\mathbb{F}_q$ is a polynomial which cannot be factorized into factors with coefficients belonging to $\mathbb{F}_q$.

An irreducible polynomial may be understood as the product of factors $(x - \omega^i)$ where ω is a primitive root of unity over a field extension of $\mathbb{F}_q$. The irreducible polynomial is obtained by considering the minimal polynomial of each factor as given previously until receiving a factorization with coefficients entirely belonging to $\mathbb{F}_q$.

Definition 2.1.27: The **generator polynomial** $g(x) \in \mathbb{F}_q[x]/(x^n - 1)$ for a cyclic code $\mathcal{C}$ is said to generate $\mathcal{C}$ as follows. $\vec{c} = [c_0, c_1, \ldots, c_{n-1}] \in \mathcal{C}$ if and only if there exists some $c(x) = c_0 + c_1x + c_2x^2 + \ldots c_{n-1}x^{n-1} \in \mathbb{F}_q[x]/(x^n - 1)$ such that $c(x) = a(x)g(x)$ for some $a(x) \in \mathbb{F}_q[x]/(x^n - 1)$.

The above definition is quite technical, but what it is essentially saying is that polynomial multiplication over $\mathbb{F}_q[x]/(x^n-1)$ is analogous to cyclicly shifting a code word. This gives us a very strong theoretical foundation upon which we may discuss and understand cyclic codes. A fundamental theorem related to generating cyclic codes states that a linear code $\mathcal{C}$ is cyclic if and only if its generator polynomial $g(x)$ is a divisor of $x^n - 1$ over $\mathbb{F}_q$. There are many beautiful properties of cyclic codes which we could analyze further with this foundation, but since we hope to understand the Golay code, we will opt to understand the notion of a BCH code next.

Definition 2.1.28: A cyclic code of length n over $\mathbb{F}_q$ is called a **BCH code of designed distance** δ if its generator polynomial $g(x)$ is the least common multiple of the minimal polynomials of β^l, β^{l+1}, $\ldots$, $\beta^{l+\delta-2}$ for some l where β is a primitive nth root of unity. The minimum distance of a BCH code with designed distance δ is at least δ.

We utilize these definitions [5] to cyclicly construct the binary $[23, 12, 7]$ Golay code in the following section, and to understand the use of a particular decoding algorithm for this Golay code.

2.2 Constructions of the Golay Code

Lemma 2.2.1: $g(x) = x^{11}+x^9+x^7+x^6+x^5+x+1$ is the cyclic generator polynomial for the $[23, 12, 7]$ binary Golay code.

> **Proof**: By noting that $x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1$ is an irreducible factor of $x^{23} - 1$ over $\mathbb{F}_2$, we have that there exists an $\alpha \in \mathbb{F}_{2^{11}}$ such that $g(\alpha) = 0$. Noting that α is, in fact, a primitive 23rd root of unity in $\alpha \in \mathbb{F}_{2^{11}}$, it follows that $g(x)$ generates a $[23, 12, 7]$ cyclic code; the only such code is the binary Golay code [7].

Thus, there is a cyclic construction for the $[23, 12, 7]$ binary Golay code; its code words may be written as $c(x) = a(x)g(x)$ with $g(x)$ as defined in the above theorem and $a(x) \in \mathbb{F}_q[x]/(x^{23} - 1)$. We may use this definition to construct a basis for the $[23, 12, 7]$ binary Golay code, and thus, a generator matrix.

Theorem 2.2.3: The following is a generator matrix for the $[23, 12, 7]$ binary Golay code.

$$\begin{vmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&0&0&0&1&1&1&0\\
0&1&0&0&0&0&0&0&0&0&0&0&1&1&1&0&1&1&1&0&0&0&1\\
0&0&1&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0&0&1&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&1&0&1&1&0&0&1&1&0&1&1\\
0&0&0&0&1&0&0&0&0&0&0&0&0&1&1&0&1&0&0&1&0&1&1\\
0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&1&1&1&1&1&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&1&0&1&1&1&0&0&1\\
0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&1&1&1&1&1&1&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&1&0&1&0&0&1&0&1&1&0&1\\
0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&1&1&1&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&0&1&0&1&0&1&1&0
\end{vmatrix}$$

Proof: Consider the matrix given by the first 12 cyclic shifts of the coefficients of $g(x)$ as defined by theorem 2.2.2 (pictured below).

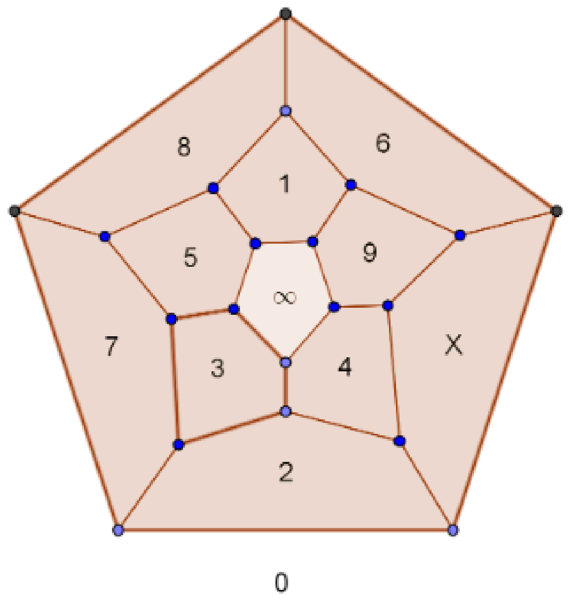
$$\begin{vmatrix}
0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&1&1&0&0&0&1&1\\
1&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&1&1&0&0&0&1\\
1&1&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&1&1&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&1&1&0&0\\
0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&1&1&0\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&1&1\\
1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&1\\
1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1\\
1&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&1&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1\\
1&0&1&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&1&0&1&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{vmatrix}$$

One may verify that by row reducing this matrix over $\mathbb{F}_2$, we receive the matrix given in the statement of the theorem.

As hinted by the leadup to this section, the $[23, 12, 7]$ binary Golay code is an incredibly symmetric and elegant object. Now that we have a generator matrix for this code, we may explore a geometric method for generating the $[23, 12, 7]$ binary Golay code.

Proposition 2.2.4: The generator matrices for the $[24, 12, 8]$ binary extended Golay code, and the $[23, 12, 7]$ binary Golay code, may be obtained by considering the regular dodecahedron.

The first step in obtaining the $[23, 12, 7]$ binary Golay code from the regular dodecahedron is to project the object onto a plane in such a way that causes the resulting projection to be a planar graph. This is given in the following figure courtesy of R.T. Curtis.



Suppose we then define an adjacency condition in the following way. Let a face in the above graph be defined by the cycle about a unique label. Two faces are adjacent if and only if they share a single edge in common. Using this condition, we may devise an adjacency matrix in the following way. Allow the ith row and column to signify the ith face of the above projection. Place a 0 at the ith column and the jth row if and only if the ith and jth faces are adjacent. The resulting matrix is as follows [8].

$$\left|\begin{array}{cccccccccccc}
1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1
\end{array}\right|$$

Suppose we remove the trailing column (corresponding to face X) in the above matrix. We observe that the resulting matrix is equivalent to the submatrix consisting of the trailing 11 columns of the generator matrix given in theorem 2.2.3. Take this punctured matrix and append the identity matrix I_{12} to the left side of it, forming a 12 by 23 matrix. This resulting matrix is exactly the generator matrix for the $[23, 12, 7]$ binary Golay code.

Furthermore, suppose we did not puncture the trailing row from the above adjacency matrix, and instead appended the identity matrix I_{12} to the above adjacency matrix.

It turns out that this is a generator matrix for the extended $[24, 12, 8]$ binary Golay code. This can be verified by applying an even parity check to the rows of the generator matrix for the $[23, 12, 7]$ binary Golay code.

2.3 Properties of the Golay Codes

Lemma 2.3.1: The $[23, 12, 7]$ Golay code is a perfect code.

Proof: Considering the discussion in section 2.2, the existence of the $[23, 12, 7]$ Golay code is trivial. We simply plug the parameters of this Golay code into the sphere packing bound to verify our result.

$$2^{23} = 2^{12} \sum_{i=0}^{3} \binom{23}{i}$$

$$\Rightarrow 2^{11} = \sum_{i=0}^{3} \binom{23}{i}$$

$$\Rightarrow 2048 = 1 + 23 + 253 + 1771$$

$$\Rightarrow 2048 = 2048$$

The $[23, 12, 7]$ Golay code achieves the sphere packing bound, and it is thus a perfect code.

In the paper *Notes on Digital Coding* in which Marcel Golay introduced the $[23, 12, 7]$ Golay code, Golay was essentially reporting on a computer search, the goal of which was to uncover perfect codes [3]. Little did Golay know, to attain such a goal would be to uncover the interesting results shown in the below theorem and its following remarks.

Theorem 2.3.2: Excluding the repetition code of length 7, the binary $[23, 12, 7]$ Golay code is the only perfect binary code with minimum distance 7.

Proof: Suppose C is a binary perfect code of length n with minimum distance 7. Then $7 = 2e + 1$, implying that $e = 3$. As a result, every $x \in \mathbb{F}_2^n$ has distance ≤ 3 to exactly one code word $y \in C$. Therefore, if we consider balls of radius 3 around each $y \in C$, the summation of the volume of these balls must equal 2^n since C is binary.

$$|C| \sum_{i=0}^{3} \binom{n}{i} = 2^n.$$

Note that $|C| = 2^k$ for some $k \in \mathbb{N}_{<n}$, and that $n \in \mathbb{N}$. Also note that $\binom{n}{0} = 1$, and

$\binom{n}{1} = n$. Furthermore note that $\binom{n}{2} = \frac{n(n-1)}{2}$. Finally note that $\binom{n}{3} = \frac{n(n-1)(n-2)}{6}$. Thus, the previous expression may be rewritten as,

$$\tfrac{6(n+1)+3n(n-1)+n(n-1)(n-2)}{6} = 2^{n-k}$$

$$\Rightarrow \tfrac{6(n+1)+n(n-1)(3+(n-2))}{6} = 2^{n-k}$$

$$\Rightarrow \tfrac{6(n+1)+n(n-1)(n+1)}{6} = 2^{n-k}$$

$$\Rightarrow \tfrac{(n+1)(6+n(n-1))}{6} = 2^{n-k}$$

$$\Rightarrow (n+1)(6+n(n-1)) = 3(2^{n-k+1})$$

Therefore, $n+1$ is a factor of the expression $3(2^{n-k+1})$ and can be written as $n+1 = 3^i(2^{n-k+1})$ with $i \in \{0,1\}$. Suppose $n-k+1 \geq 4$. Then $n+1 = 16m$ for some $m \in \mathbb{Z}_{\geq 0}$. Therefore, the factor $6+n(n-1)$ may be rewritten in terms of m as follows,

$$6+n(n-1) = 6+(16m-1)(16m-2)$$

$$\Rightarrow 6+n(n-1) = 6+(16m-1)(16m-2)$$

$$\Rightarrow 6+n(n-1) = 256m^2-48m+8$$

$$\Rightarrow 6+n(n-1) = 8(32m^2-6m+1)$$

Trivially, if $m = 0$, then $n = -1$, which is impossible. Thus, $m \in \mathbb{Z}_{\geq 1}$. However, this would imply that the term $6+n(n-1)$, which is a factor of the term $3(2^{n-k+1})$, has an odd factor greater than 3, which is a contradiction. Therefore, $0 \leq n-k+1 \leq 3$. We may check these cases.

First consider $n-k+1 = 0$. This is impossible since $n+1 = 3^i \Rightarrow n = 3^i - 1$ with $i \in \{0,1\}$. C cannot have minimum distance 7 when $n \in \{0,2\}$ is necessarily less than 7. The same is true for $n-k+1 = 1$ as $n = 3^i(2^1)-1$ with $i \in \{0,1\}$. Again, C cannot have minimum distance 7 when $n \in \{1,5\}$ is necessarily less than 7. Considering $n-k+1 = 2$, we have $n = 3^i(2^2)-1$ with $i \in \{0,1\}$. Therefore $n \in \{3,11\}$ may be achievable for the case $n = 11$ (but not for $n = 3$ by the previous reasoning). To verify this, we may check the other factor in a similar manner as before.

$$6+n(n-1) = 6+11(10)$$

$$\Rightarrow 6+n(n-1) = 116$$

$$\Rightarrow 6 + n(n-1) = 2^2(29)$$

Since $3 \nmid 29$, we have a contradiction since the other factor cannot be a factor of the expression $3(2^{n-k+1})$ when $n = 11$. Thus, we may finally check the case $n - k + 1 = 3$. In this case, we have that $n = 3^i(2^3) - 1$ with $i \in \{0, 1\}$, meaning that $n \in \{7, 23\}$. We may we may ignore the case $n = 7$ as it corresponds to the binary repetition code of length 7. Instead, we check $n = 23$.

$$6 + n(n-1) = 6 + 23(22)$$

$$\Rightarrow 6 + n(n-1) = 6 + 23(22)$$

$$\Rightarrow 6 + n(n-1) = 6 + 506$$

$$\Rightarrow 6 + n(n-1) = 512$$

$$\Rightarrow 6 + n(n-1) = 3^0(2^9)$$

Therefore, we see that for $n = 23$, the factor $6 + n(n-1)$ is a factor for the expression $3(2^{n-k+1})$. Noting that this corresponds to the golay code, we may verify that this case is achievable.

A more general result in coding theory as shown by Van Lint and Tietavainen states that any non-trivial perfect q-ary code where q is a prime power must have the same parameters as one of the Hamming or Golay codes. Beyond this, there is a conjecture which hypothesizes that there are no non-trivial perfect codes over non-prime-power alphabets [9].

Theorem 2.3.3: The binary $[24, 12, 8]$ extended Golay code is self-dual.

Proof: We verify this by computer-aided brute force calculation. See [10].

Theorem 2.3.4: The binary $[23, 12, 7]$ Golay code is a BCH code.

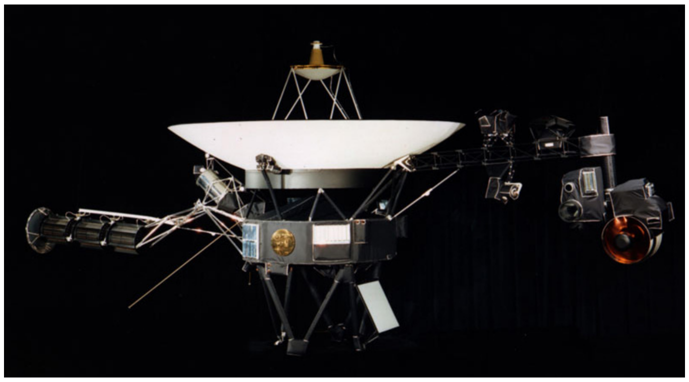
The above theorem follows from lemma 2.2.1; to elaborate on the above result, the Golay code is a BCH code of designed distance 5. As a result, we have that the binary $[23, 12, 7]$ Golay code may decode up to 2 errors using the BCH decoding algorithm as specified by Truong, Holmes, Reed, and Yin [7].

3 Practical Instances of Golay Codes

3.1 NASA Voyager Missions

There have been many great advancements made throughout human history; however, few are known so well as the ability of humans to be able to send spacecraft millions upon millions of miles to distance bodies in space and photograph them in such crisp detail as a professional photographer photographs his subjects. Though, it takes a relatively large of effort for the images from the spacecraft to be sent back to Earth. In addition, because of the great distance and the consequential noise that will persist the vast channel between the spacecraft and Earth, consideration had to be made to guarantee much of the image data would be received accurately on Earth. Initially, this was done using the Reed-Solomon code; the 7-error correcting Reed-Solomon code was what enabled some of the first clear black and white photos of the planets to be seen during the first several Mariner missions. Each mission allowed NASA mathematicians and engineers to increase the coding capabilities of the spacecraft in order to enable an ever-increasing amount of data to be accurately transmitted back. In later missions, starting with the voyager 1 and 2 missions, the extended binary Golay code was used due to necessity.

The Voyager 1 and 2 spacecrafts needed to transmit hundreds of color images of Jupiter and Saturn in their 1979, 1980, and 1981 planetary orbits around these two planets within a constrained telecommunications bandwidth, (i.e., the channel that had to be considered). Color image transmission required three times the amount of data as black and white images due to the red, green, blue channels, so the Hadamard code that was used to transmit the black and white images during the later Mariner missions was switched to the Golay $[24, 12, 8]$ code. This Golay code is only triple-error correcting, but it could be transmitted at a much higher data rate than the Hadamard code that was used during the Mariner mission [11].

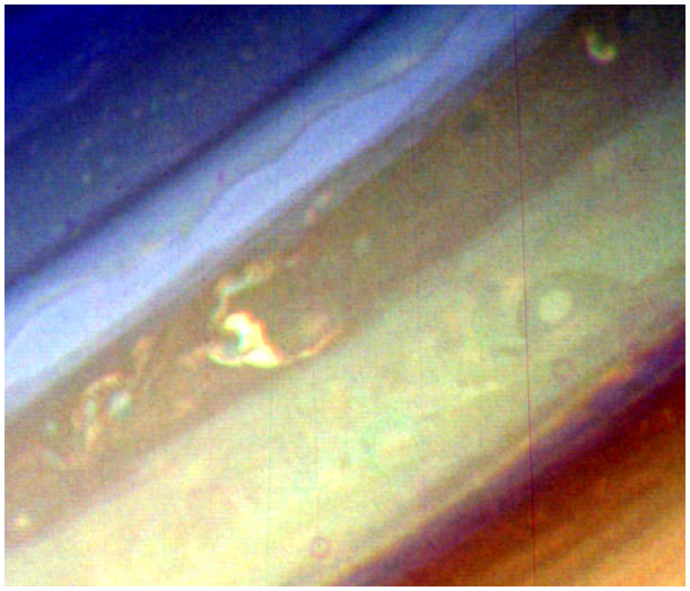


The NASA voyager spacecraft. Voyager 1 and 2 were identical in their design and physical implementation [12].

In addition, due to the vast distance that codewords would travel, the voyager 1 and 2 spacecraft were equipped with convolutional coding capabilities. Convolutional coding is a newer type of coding that gives better bit error rate reduction than block coding for a given symbol-to-bit ratio. In addition, the encoder is simpler to implement in hardware than a block encoder [13].

When even greater error correction is needed it is possible to combine coding schemes, called concatenation. The Golay encoding algorithm used at Jupiter and Saturn required the transmission of one overhead bit for every information bit transmitted (100 percent overhead). Voyager carried an experimental Reed-Solomon data encoder, expressly for the greater communication range of the Uranus and Neptune phase of the mission; the Reed-Solomon code being the outer layer and convolutional code being the inner layer. The data bits were first convolutionally coded, after which the output symbol stream from the convolutional coder was Reed-Solomon coded. This scheme provides significant increase in error protection and a significant increase in symbol rate [14].

The new Reed-Solomon encoding scheme reduced the overhead to about one bit in five (20 percent overhead) and reduced the bit-error rate in the output information from 510^{-3} to 10^{-6}. In the case of Voyager 1 and 2, this wouldn't be enough, so they went for a 24-bit long code word. From the total of 16 million code words, they only defined 4096 as valid, in other words, 12 bits carry actual information and another 12 are used for error correction. This resulted in a 3-error correcting, 7-error detecting code [15].

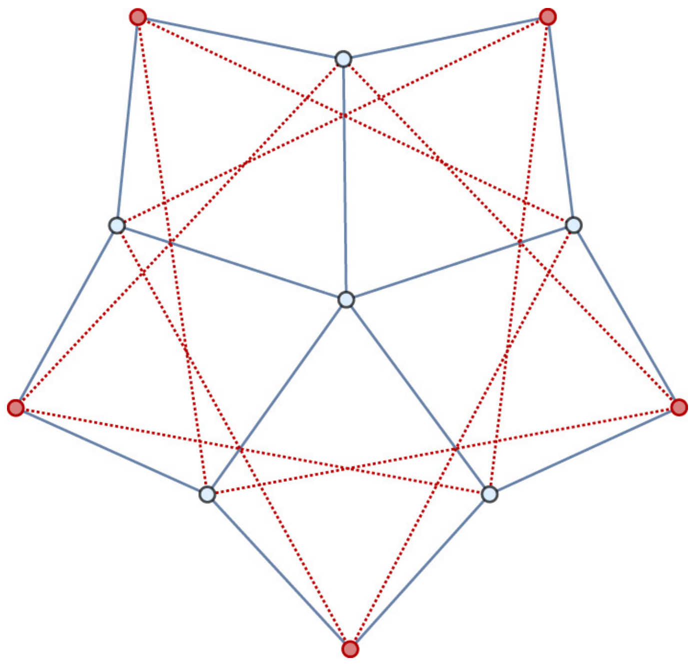

An image of Saturn's clouds captured by the NASA Voyager spacecraft [12].

To this day, the extended binary Golay code is still supported by the Defense Switched Network (DSN), the largest and most robust communications network in use by the Department of Defense, showcasing the continued vital importance the extended Golay code has for furthering human achievements. The code is still attractive for use on the DSN uplink where the spacecraft uses software decoding with the on-board computer [7].

3.2 Other Applications

In addition to its use by NASA, the Golay code is has been shown to be useful in the medical field for generating patient health history as well as analyzing patterns to help determine risk of diseases. This is achieved through a process known as clustering. Clusters are defined as a data container. Inside these containers are homogeneous data points. The Golay code clustering technique makes use of the perfect $[23, 12, 7]$ binary Golay code. For the purposes of the clustering technique, vectors are used to store any type of data. This includes but is not limited to patient info, RNA and DNA sequencing, drugs used by the patient, and medical history. When encoded according to the $[23, 12, 7]$ binary Golay code, these vectors are 23-bits in length. Each bit tells whether a patient is afflicted with some condition. Typically, a 0 represents the absence of or testing negatively for a disease or condition. Conversely, a 1 is used if patient is diagnosed with an illness. By checking the hamming distance of these vectors, it is possible to find commonality and increased risk for certain illnesses. The paper calls this the 23-bit meta template. Each template is a series of questions with each question answer being stored as a bit in a vector. So if, say, the hamming distance of two vectors is four, then this shows that their data in these vectors have commonality and thus they are clustered together. A machine that analyzes the data after clustering has taken place will be able to provide doctors with a more comprehensive overview of a patient's health [16].

Another application of the golay code is its use in quantum computation. The specific version of the golay code used is the $[11, 6, 5]_3$ ternary golay code (over $\mathbb{F}_3$); it is used to introduce fault-tolerance into quantum trits, which is achieved through magic state distillation. Fault-tolerance allows quantum computation (or any form of computation in general) to operate even in the event of some error. This usually leads to a negative tradeoff of system performance, with the added advantage of avoiding system shutdowns in the event of an error. The more fault-tolerance built into a system, the more errors the system is equipped to deal with at once before failing. As for magic state distillation, it is the process of turning quantum states into something more reliable to work with, in this case in hopes of presumably achieving more effective fault-tolerance. To elaborate, in quantum computing, there is a concept known as quantum decoherence, the process by which there is a loss of information stored on the qubits over time. This loss of information is unavoidable as operations are performed. The more time spent on an operation, the greater the likelihood that errors due to decoherence will occur. However, by applying a ternary Golay encoding scheme to 11 qutrits and treating the larger system as one qutrit (with more protection against error), the quantum computer would be able to handle this natural decoherence much more than if we were to consider an uncoded qutrit. Provided a larger number of qutrits, this allows for much more time-consuming calculations to take place even in the face of decoherence [17].



A graph state representation of the 11-qutrit Golay code [17].

4 Interact with Golay Codes: An Interactive Web Application

4.1 Overview

In addition to our report, we intended as a group to produce a publicly-available, open-source web application which would allow those interested in the binary extended Golay code to interact with it. First, such interaction would both allow users to interact with individual code words in order to understand the theoretical error-correction properties of the binary extended Golay code. In addition, it would allow users to encode and "send" image data through a simulated noisy channel to display the error-correction properties of the binary extended Golay code in a practical environment. In the following sections, we discuss the design of the web application at various levels of abstraction, after which we discuss the various areas for further work implied by the current iteration of the application.

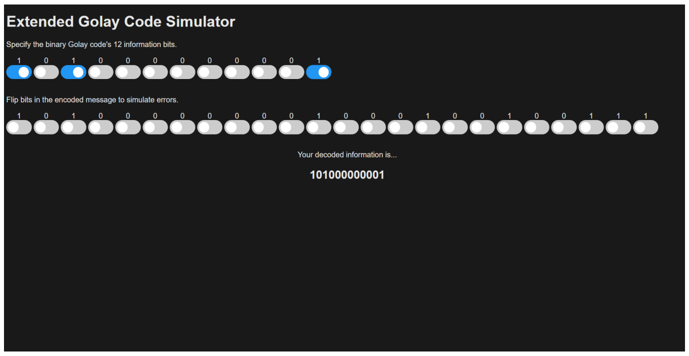
4.2 Architecture

At a high level, we planned for the application to take the form of a full-stack, loosely coupled client-server web application. In order to achieve this, we broke our application into two main components, namely the front-end and the back-end. The front-end of our application took the form of an Express server built using NodeJS which would serve the various static HTML, CSS, and client-side JavaScript assets to be loaded in the user's browser. These front-end assets would then communicate with the back-end of our application using AJAX. Meanwhile, the back-end of our application took the form of a Flask server built using Python, which would handle all of the bitwise operations implied by encoding the information, "transmitting" the code words of the Golay code, and decoding the Golay code's code words back into the original information. Finally, the front-end and back-end components are containerized using Docker, and they are deployed on an Amazon Web Services EC2 instance.

4.3 Front-End Design

All in all, there are five total views that are served by the Express server on the Front-end. Three of which lack any dynamic JavaScript component and thus are rather insignificant in terms of discussion, albeit one of which hosts this report, and another covers some overview surrounding the Golay code which is also covered more in-depth by this report. The remaining view is simply an "about" page covering some information about the creators of the web application. Thus, we focus the majority of our discussions on the views which contain a dynamic JavaScript component, those being the extended Golay code simulator, and the NASA Voyager simulator.

For the first of the two dynamic, interactive web applications, we have the extended Golay code simulator. This application is far simpler than its counterpart, as its intention is for users to work with individual code words at a time. To do so, there are two rows of 12 and 24 on/off slide switches respectively. The switches allow the user to control the values of each of the information bits, and each of the code word's bits respectively by row. By flipping a switch in the first row which controls the information bits, the front-end proceeds to send the new string of information bits to the back-end which encodes the information and promptly sends it back to the front-end for display. The front end then displays the 24 encoded bits in the second row alongside their respective switches.



Extended Golay Code Simulator

Specify the binary Golay code's 12 information bits.

1 0 1 0 0 0 0 0 0 0 0 1

Flip bits in the encoded message to simulate errors.

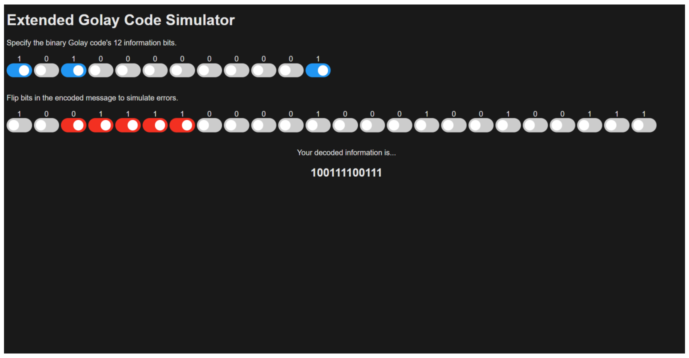
1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 1 1

Your decoded information is...

101000000001

Information specified by the user which is encoded, displayed, and decoded correctly as there are no errors present in the code word.

In the second row, the switches instead correspond to errors in the encoded bits; each time a switch is toggled, its respective bit is flipped, and the front-end sends a request to the back-end to decode the encoded bits with the errors included. Below both rows of switches, there is a display showing the final decoded information bits. Thus, the user can verify that for any code word, up to three errors can safely occur without loss of information.



Extended Golay Code Simulator

Specify the binary Golay code's 12 information bits.

1 0 1 0 0 0 0 0 0 0 0 1

Flip bits in the encoded message to simulate errors.

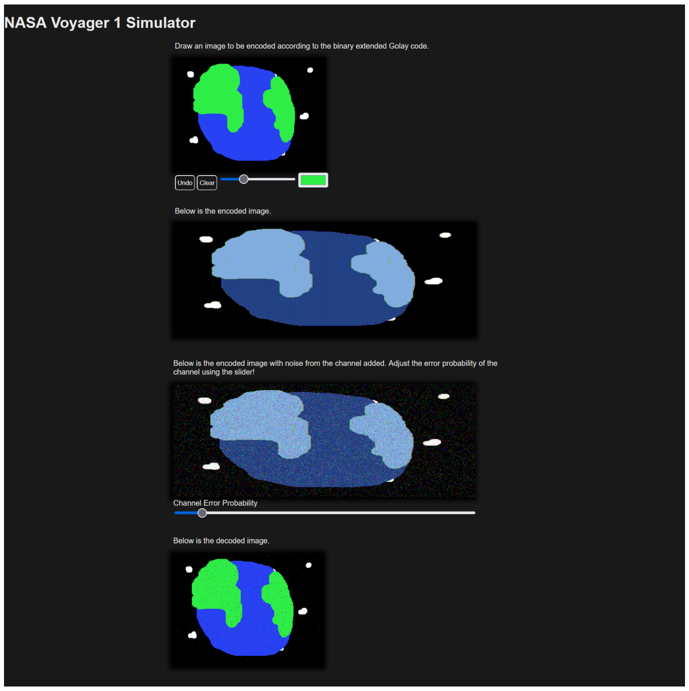
1 0 0 1 1 1 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 1 1

Your decoded information is...

100111100111

An instance in which errors in the Golay code's bits yield incorrectly decoded information.

The other dynamic, interactive web application is the NASA Voyager simulator. This application was meant for users to visualize an instance of practical use of the Golay code. To do so, we attempted to emulate how the Golay code was used to protect image data against the radiation in space during the NASA Voyager missions. Throughout the application's display, there are four HTML canvas elements which are used to display image data at different stages of the "source-encoder-channel-decoder-message" pipeline. The first 320 by 240 HTML canvas allows the user to draw a simple image to be encoded. Following this, the image data is sent row-by-row (of pixels) to the back-end which encodes it according to the Golay code's generator matrix. The image is then displayed on the following 640 by 240 HTML canvas; the canvas is twice the width since for every 12 information bits in the binary extended Golay code we have 12 additional redundant bits. Upon the loading of the final row of image data in the encoded image, the front-end then begins to send row-by-row requests to the back-end to apply noise to the encoded image. The noise is applied according to a stochastic probability specified by the user using a slider control located below the third HTML canvas where the noisy image is displayed. Upon completion of noise application, the front-end proceeds to send row-by-row requests for the noisy image to be decoded. Finally, upon completion of decoding, the decoded image will be displayed on the fourth HTML canvas element, and the user will have a view of the entire lifecycle of their original image through the ”source-encoder-channel-decoder-message” pipeline.



NASA Voyager 1 Simulator

Draw an image to be encoded according to the binary extended Golay code.

Below is the encoded image.

Below is the encoded image with noise from the channel added. Adjust the error probability of the channel using the slider!

Channel Error Probability

Below is the decoded image.

A drawing of Earth is encoded, sent through a noisy channel with an error probability of roughly 0.08, and is decoded.

4.4 Back-End Design

Implied by the front-end’s design are various back-end requirements for which we accounted in the design of the back-end. There are five main API endpoints for the two aforementioned dynamic applications which themselves motivate a slew of additional functions specified on the back-end. The five functions mapped to each of the API endpoints will guide the discussions of this section, and when needed we drill down and examin their dependent functions.

The first function we examine is that which is mapped to the "/extended-binary/encode" endpoint. This function is designed with the extended binary Golay code simulator in mind, and it handles the encoding of the information bits specified in that view. It takes an array of 12 integers as its input, those being the information bits of the message, and it outputs an array of 24 integers, those being the code word corresponding to the input information bits. It does so by first allocating memory for a code word (an unsigned integer length 32), then looping through information bits in the provided list, and any time a 1 is detected at a particular position, which we'll call i, it performs a bitwise XOR operation on the code word and the ith row of the extended binary Golay code's generator matrix. The resulting unsigned integer of length 32 is then converted back into a list of integers by breaking it into bytes and using Numpy's *unpackbits* function to loop through the integer's first 24 bits and append them to a list as integers. This is the list of integers which is sent in the JSON response by the API function, its corresponding key being "codeWord".

The other function related to the extended binary Golay code simulator is that which is mapped to the "/extended-binary/decode" endpoint. This function handles the decoding of a provided list of length 24 containing integer values 0 and 1, which itself is defined by a code word plus some error pattern as specified presumably by the user on the front-end. This function operates by first converting the provided list to an unsigned integer of length 32 similarly as was discussed previously. Then, the function locates the minimum distance code word to the provided code word with errors by applying a brute-force search, returning a code word of the Golay code. This is rather slow for most purposes, but for the purpose of the extended binary Golay code simulator in which the user is operating on a single code word, the delay is mostly unnoticeable. Finally, the function converts the code word back to a list of integers containing only the first 12 bit values of the code word, which are presumably the information bits specified by the user provided few enough errors. This is the list of integers which is sent in the JSON response by the API function, its corresponding key being "informationBits".

The next function we examine is that which is mapped to the "/extended-binary/encode-image" endpoint. This function is designed with the NASA Voyager simulator in mind, and as the endpoint implies, it encodes a chunk of image data according to the binary extended Golay code's generator matrix. This function relies on the same encoding methodology as was discussed in the first encoding endpoint, however it also has some additional nuance related to the context of the data being passed in which we discuss here. First and foremost, the data being passed in is representative of the nm pixels of an n by m image according to the ImageData object built into JavaScript. Each pixel contains four values (unsigned integers of length 8) which correspond to its RGBA values (red, green, blue, and alpha). We chose not to take transparency into account, opting to hold all alpha values constant at 255. Thus, we considered the

RGB values, 24 bits in total, as the information contained in the pixel. As a result, we have that any single pixel maps to exactly two pixels in the following way. First, take the 8 bits from the red channel, and the leading 4 bits from the green channel and this is the information for the first code word. Next, take the trailing 4 bits from the green channel and the 8 bits from the blue channel and this is the information for the second code word. The two strings of information are encoded, and the resulting code words are broken into RGB values based on their position in the code word (i.e. the leading 8 bits are the red value, etc.). As mentioned previously, the alpha value is then appended since we consider only alpha values of 255, and the two pixels are placed next to one another in the resulting array containing the encoded image data. This encoded image data is then returned to the front-end with the width and height of the image specified according to the ImageData object built into JavaScript.

Another function related to the NASA Voyager simulator is that which is mapped to the "/extended-binary/decode-image" endpoint. Similarly to the previously discussed function, this function is designed with the ImageData object that is built into JavaScript in mind. In fact, it operates almost perfectly in reverse to the aforementioned function, and it follows the same decoding methodology as was described in the previous decoding endpoint. This function simply receives the image data array of encoded pixels, and runs various bitwise operations on the pixels to eventually retrieve two 24-bit messages which are each code words with added noise. The 24-bit messages are decoded by either a brute force search or by use of a lookup table for optimization purposes depending on the message. Upon decoding, the 12 information bits from each of the two decoded code words are appended together, forming a single string of 24 bits which corresponds to exactly one pixel. Using bitwise operations, the pixel's RGB values are extracted, and the alpha value of 255 is appended, forming an image of half the width of that which was passed in, and the same height. The decoded image data is returned to the front-end with the width and height of the image specified according to the ImageData object built into JavaScript.

The final function composing the outward-facing API of the back-end is that which is mapped to the "/binary-channel/transmit-image" endpoint. This function is slightly different from the previous function as it does not relate to the Golay code in any way. Rather, its purpose is to automatically apply noise to the encoded image data according to some stochastic probability to simulate the encoded image being sent through a binary symmetric channel with the provided probability being the error probability for any given bit. It does so by unpacking the bits of each encoded pixel, looping through the bits, and generating a random number according to a binomial distribution with the given error probability as a parameter. If the random number is 1, then the bit at that particular position is flipped. This is done for each pixel, after which the image data is translated back into its previous format according to the ImageData object built into JavaScript and sent as the JSON response.

4.5 Areas for Further Work

Throughout the development of *Interact with Golay Codes*, there were many hurdles encountered along the way. Nonetheless, no such hurdle posed more of a challenge than that of optimizing the decoding of noisy code words, especially those in the image data of the NASA Voyager simulator. As mentioned throughout the previous section, we currently utilize a brute-force search decoding algorithm, which is by far the slowest among the reasonable set of options. The main next step in improving the NASA Voyager simulator portion of *Interact with Golay Codes* will be to optimize this decoding algorithm in terms of its speed. One option may be to utilize a syndrome decoding algorithm alongside an additional lookup table; after all, the current method used to optimize the decoding is the use of a lookup table for code word-to-code word mappings and for common errors, and thus we are no strangers to the use of lookup tables. Furthermore, the BCH decoding algorithm discussed previously would likely prove useful in terms of optimization as well, albeit it would come at the expense of potentially not being able to decode 3 errors.

As for more miscellaneous or distant progress that could be made on *Interact with Golay Codes*, we have some added features that we think may be interesting to implement. First, as discussed in previous comments, the binary Golay code (not the extended Golay code) is an extremely elegant and symmetric structure with ties to groups and lattices of significance throughout mathematics. Thus, we think there may be some way to let our imagination run wild and somehow attempt to visualize these higher-dimensional symmetries in some sort of interactive web application. An equally ambitious feature that could be added to our application may be to attept to visualize the use of the Golay code in quantum computation in some interactive way. Perhaps such an application would utilize the graph state of the 11-qutrit quantum Golay code. Lastly, it is worth mention that we had initially considered implementing a voice encoding application which would operate similarly to the NASA Voyager simulator, but with voice recordings. This still seems like a reasonable endeavor considering the existing back-end code that we have at our disposal. Were we to continue working on this application, this seems like the reasonable next addition to our backlog (that is, after optimizing our decoding algorithm) [18].

5 Final Thoughts

It is impossible to know what Marcel Golay expected would be the result of his computer-aided search for perfect codes. One thing that is for sure, though, is that what Golay found is a mathematically beautiful object that has since played a role in some of the great human achievements of the past decade. We hope that we have adequately summarized such significance of Golay's findings, and we hope that *Interact with Golay Codes* detailing the Golay code helps to portray its significance.

6 Bibliography

[1] C.E. Shannon, ”A Mathematical Theory of Communication”. Nokia Bell Labs, 1948. The Bell system technical journal, p. 379-423.

[2] R.W. Hamming. ”Error detecting and error correcting codes”. Nokia Bell Labs, 1950. The Bell system technical journal, p. 147-160.

[3] E.R. Berlekamp. ”Key Papers in the Development of Coding Theory”. IEEE Press, 1974. p. 4.

[4] M.J.E. Golay. ”Notes on Digital Coding”. Proc. IRE, 1940. p. 37.

[5] J.H. Van Lint. ”Introduction to Coding Theory”. Springer, 1999.

[6] D.C. Lay, S.R. Lay, and J.J. McDonald. ”Linear Algebra and Its Applications”. Pearson, 2021.

[7] T.K. Truong, J.K. Holmes, I.S. Reed, and X. Yin. ”A Simplified Procedure for Decoding the [23, 12] and [24,12] Golay Codes”. NASA, 1988.

[8] R.T. Curtis. ”Error-Correction and the Binary Golay Code”. London Mathematical Society Impact150 Stories, 2016. p. 51-58.

[9] R. Hill. ”A First Course in Coding Theory”. Oxford University Press, 1986.

[10] E.M. Balcik. ”Math 468 Homework 5”. GitHub, 2021. [Online]. Link.

[11] B. Cherowitzo. ”Combinatorics in Space - The Mariner 9 Telemetry System”. University of Colorodo, Denver. [Online]. Link.

[12] ”Voyager”. NASA Jet Propulsion Laboratory, Californial Institute of Technology. [Online]. Link.

[13] C.D. Brown. ”Elements of Spacecraft Design”. American Institute of Aeronautics and Astronautics, Inc., 2002.

[14] R. Ludwig and J. Taylor. ”Descanso Design and Performance Summary Series - NASA”. NASA, 2002.

[15] Uhoh, et. al. ”How is Stacking Oranges in 24 Dimensions Related to Receiving and Decoding SIgnals from the Voyagers?”. Space Exploration Stack Exchange, 2021.

[Online]. Link.

[16] F. Alsaby, K. Alnowaiser, and S. Berkovich, “Golay Code Transformations for Ensemble Clustering in Application to Medical Diagnostics,” thesai, 2015. [Online]. Link. [Accessed: 2021].

[17] S. Prakash, “Magic state distillation with the ternary Golay code,” US National Library of Medicine National Institutes of Health, 2020. [Online]. Link. [Accessed: 2021].

[18] E.M. Balcik. ”Math 468 Final Project”. GitHub, 2021. [Online]. Link.