Single-Track Gray Codes: An Efficiently Decodable Maximal Period General Construction

Georgie Botev

Massachusetts Academy of Math & Science at WPI

Abstract

Gray codes are sequences of binary codewords with numerous applications in telecommunications, robotics, error correction, combinatorics, and graph theory. Single-track Gray codes are a subclass of Gray codes that can be specified by a single binary vector. These codes can be constructed from a seed code, which becomes computationally difficult to generate for bit-lengths greater than 37. A general construction for single-track Gray codes is introduced, and the resulting codes yield a full-period code with 2^n distinct codewords. Unevenly spaced sensors and logical bitwise operators were utilized to make decoding efficient.

1 Introduction

Gray codes have numerous applications in telecommunications and robotics. When they were first introduced, they had a pivotal role in enhancing glitch-free operation of devices that performed analog to digital conversions. Today, they are widely used in optical and magnetic encoders. These codes also have interesting mathematical properties that make them prominent in the study of combinatorics, group theory, and puzzles, such as the Tower of Hanoi and Baguenaudier. The binary reflected Gray code (BRGC) is the most well-known multi-track code due to its efficient decoding algorithm and error-checking capabilities that are essential for many real-time applications. An elegant construction that transforms special multi-track (seed) codes to single-track Gray codes (STGC)s with evenly spaced sensors was shown to exist about forty years after the BRGC was introduced. However, STGCs have not yet been researched as thoroughly as the BRGC, and a general construction for their seed codes is still lacking in the literature. A discovery of the like would significantly reduce the physical size of encoders while increasing their resolution. We introduce an STGC construction that employs a novel application of unevenly spaced sensors that is easily decodable, has maximal period, and has minimal drawbacks.

2 Literature Review

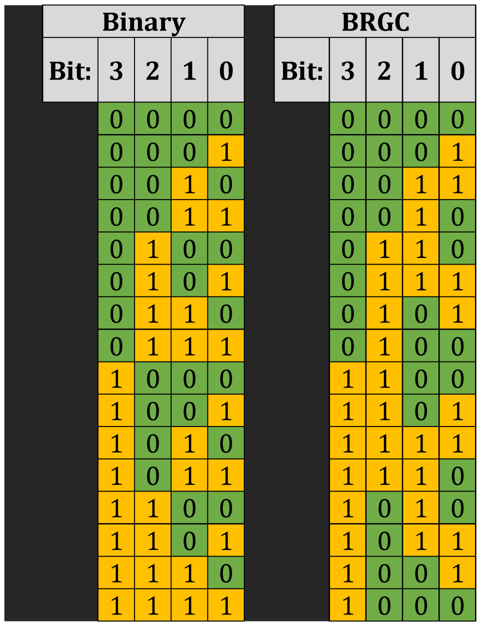
2.1 Brief History

On November 13th, 1947, Frank Gray introduced the concept of Gray codes with his patent application entitled "Pulse Code Communication", but it was not until March 17th, 1953 that his patent was finally published [1]. In fact, Donald Knuth suggests in [2] that the idea of such a code was known at least a decade earlier from [3]. Nevertheless, the term *Gray code*, referring to Gray himself, was popularized by other prominent scientists at the time. His proposed coding rule is known today as the binary reflected Gray code, named after its recursive construction. Gray was both a researcher and a physicist by trade who worked at Bell Laboratories. His contribution has been influential in the fields of mathematics and telecommunications, where these codes have had a fundamental role in enhancing error correction techniques.

2.2 Gray Code Definition

There most common way to define the Gray code is as a sequence of n-tuples, or codewords, ordered such that successive codewords differ in exactly one bit. Furthermore, for k-ary Gray codes with $k > 2$, the difference in value of the differing bit between adjacent codewords cannot be greater than one. If the last and first codewords also differ by at most one in exactly one bit, then the code is cyclic; otherwise, it is acyclic. If we denote the k^{th} codeword of a binary Gray code as g_k and the numerical representation of a in radix b by $(a)_b$, then the Gray property implies that there exists a nonnegative $m \in \mathbb{Z}$ such that $(g_k)_2 \oplus (g_{k+1})_2 = (2^m)_2$, where $0 \leq m \leq n-1$ [4]. A comprehensive survey on Gray codes and their application to combinatorial algorithms is given in [2] and [5]. Moreover, Knuth concisely presents efficient algorithms to encode and decode the binary reflected Gray code in [2]. Gray codes can also be viewed as Hamiltonian paths on the hypercube graph Q_n, where n is the bit-length of the codewords [6]. In fact, cyclic Gray codes induce Hamiltonian cycles on Q_n.

In a multi-track Gray code, n sensors are needed to identify each of the codewords. In Figure 1, the difference between regular binary codewords and the multi-track BRGC is shown.



Binary						BRGC				
Bit:	3	2	1	0		Bit:	3	2	1	0
	0	0	0	0			0	0	0	0
	0	0	0	1			0	0	0	1
	0	0	1	0			0	0	1	1
	0	0	1	1			0	0	1	0
	0	1	0	0			0	1	1	0
	0	1	0	1			0	1	1	1
	0	1	1	0			0	1	0	1
	0	1	1	1			0	1	0	0
	1	0	0	0			1	1	0	0
	1	0	0	1			1	1	0	1
	1	0	1	0			1	1	1	1
	1	0	1	1			1	1	1	0
	1	1	0	0			1	0	1	0
	1	1	0	1			1	0	1	1
	1	1	1	0			1	0	0	1
	1	1	1	1			1	0	0	0

Figure 1: A side by side comparison of regular binary codewords and the binary reflected Gray code is shown above. This makes the Gray property between successive codewords evident.[1]

2.3 Multi-Track Gray Codes

Gray codes can be implemented to correct errors in absolute positioning sensors that would otherwise occur if the tracks were encoded using traditional binary codewords. Utilizing multi-track Gray codes in lieu of binary codewords was the initial approach to achieve glitch-free operation for these devices. In either case, n sensors can uniquely identify a total of 2^n sectors on the encoder.

First, we consider a traditional binary encoder as shown in Figure 2. There are $n+1$ concentric circles, including the boundary contours, that partition the encoder into 2^n sectors and n tracks. Furthermore, the n sensors are arranged radially, oriented in the same direction and distributed evenly among these tracks. The limitations of such encoders become apparent when the sensors are on the boundary of two sectors. In theory, transitions between sectors should be instantaneous, but in reality, sensors are not perfectly aligned. As a result, they may not detect the subsequent codeword in synchrony. Therefore, faulty data is likely to be read by the sensors during operational use. In fact, the noisy data is

[1]Adapted from https://computingrmurray2.wordpress.com/2015/01/26/binary-reflected-gray-code/

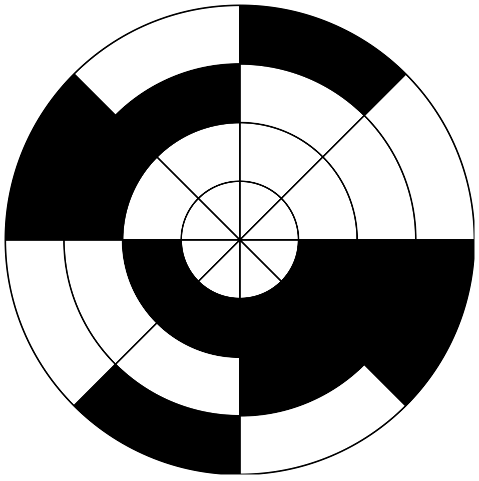


Figure 2: A 3-bit multi-track binary rotary encoder is shown above. The 3 sensors are positioned along the 3 tracks with the same orientation. The encoder shaft is represented by the white center.[2]

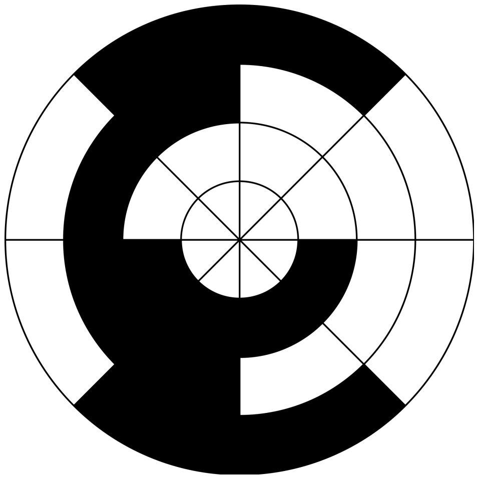


Figure 3: A 3-bit multi-track binary reflected Gray code rotary encoder is shown above. The 3 sensors are positioned along the 3 tracks with the same orientation. The encoder shaft is represented by the white center.[3]

haphazard in nature, making it difficult to correct.

On the other hand, the use of Gray codes as seen in Figure 3 is advantageous because it forces exactly one sensor to detect the change between successive codewords. This minimizes noisy data that would otherwise be misleading. In fact, inaccurate data from decoding errors is reduced to no more than $\frac{2\pi}{2^n} = \frac{\pi}{2^{n-1}}$ radians. Unfortunately, implementing a high resolution encoder for applications with size constraints is challenging. As the bit-length increases, the distance between successive features on the encoder decreases. In particular, because the circumference is directly proportionate to the radius of the encoder, the overall resolution is limited by the innermost track. The aforementioned physical limitations make it difficult to construct multi-track encoders with large bit-lengths [7, 8, 9].

2.4 Single-Track Gray Codes

Single-track Gray codes were designed to remedy the intrinsic drawbacks of multi-track encoders. In these encoders, n sensors read from the same track, and successive codewords continue to differ in exactly one bit. Mathematicians, such as Torsten Sillke, long thought it was impossible to construct single-track Gray codes, except for small bit-lenghts [10]. Surprisingly, the first single-track encoder was patented by Norman Spedding, who was

[2]Adapted from https://commons.wikimedia.org/wiki/File:Encoder_disc_(3-Bit_binary).svg by Colin M.L. Burnett under the Attribution-ShareAlike 3.0 Unported License (https://creativecommons.org/licenses/by-sa/3.0/deed.en)

[3]Adapted from https://commons.wikimedia.org/wiki/File:Encoder_Disc_(3-Bit).svg

unaware of the widespread skepticism [11]. It was shown in [9] that single-track Gray codes of bit-length n and overall period p exist for all even multiples $p \in \mathbb{N}$ of n such that $2n \leq p \leq 2^n$. It was also shown in [9] that single-track Gray codes of bit-length n and period nt exist for all even $t \in \mathbb{N}$ such that $2 \leq t \leq 2^{n-\lceil\sqrt{2(n-3)}\rceil-1}$. The standard definition of single-track Gray codes from [9] is as follows. Suppose that $[w_0, w_1, \ldots, w_{p-1}]$ is a Gray code of bit-length n and period p. If for each $0 \leq i \leq p-1$, codeword w_i has individual bits denoted by $[w_i^0, w_i^1, \ldots, w_i^{n-1}]$, then it is clear that the k^{th} column is given by $[w_0^k, w_1^k, \ldots, w_{p-1}^k]$ for each $0 \leq k \leq n-1$. The single-track Gray code satisfies

$$[w_0^k, w_1^k, \ldots, w_{p-1}^k] = [w_{c_k}^0, w_{c_k+1}^0, \ldots, w_{c_k+p-1}^0]$$

for all j columns and column-specific nonzero constants c_k. The subscripts are taken modulo p. Thus, each column is a cyclic shift of the first column. It readily follows that the i^{th} codeword is also given by

$$[w_i^0, w_{i+c_1}^0, \ldots, w_{c_{i+n-1}}^0]$$

Consequently, each of the n columns that the sensors would normally read in a multi-track fashion can also be obtained by placing all n sensors on the 0^{th} track at $0, c_1, c_2, \ldots, c_{n-1}$, respectively. If the cyclic shift for all columns is the same, as in the constructions presented in [7] and [8], then the entire code can be obtained from a smaller, seed code. As shown in Figure 4, the entire single-track Gray code can also be constructed by performing a cyclic shift equal to the length of the seed code to the vertical codewords as a group.

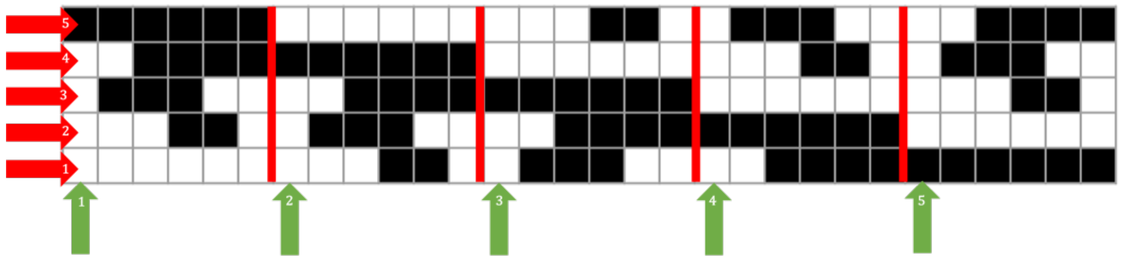


Figure 4: A 5-bit single-track Gray code and its multi-track array are shown above. Organizing the codewords in an array (red sensors) is crucial to the construction of the single-track Gray code (green sensors), which is represented by the bottom row.[4]

2.5 Lyndon Words

Combinatorial necklaces are k-ary strings of length n that form an equivalence class under cyclic permutations. Lyndon words are aperiodic combinatorial necklaces given in least

[4]Adapted from [4]

lexicographic order. They were first introduced in [12] as “standard lexicographic sequences”, and in [13], the first algorithmic way to generate Lyndon words was presented, known today as the FKM Algorithm. According to Section 2.4, it is clear that finding an arrangement of length n Lyndon words in Gray order, where the last codeword is Gray with a cyclic shift of the first codeword, satisfies the properties of a seed code. This approach was explored in [14], where graph search algorithms used brute force techniques to find valid arrangements. Using clever edge sorting techniques, such orderings of Lyndon words were found for primes less than or equal to 37. For greater bit-lengths, expected memory usage and total run-time exceed four gigabytes and one day, respectively. For prime bit-lengths greater than seven, valid arrangements relied on cyclic shifts of Lyndon words. Although this type of construction still satisfies the properties of the seed code, we believe that a general construction of a seed code composed of non-cyclically shifted Lyndon words exists for all prime bit-lengths. If such a construction were to be found, the results in [7] guarantee that an infinite family of high-period single-track Gray codes could be generated.

3 Motivation

The binary reflected Gray code can be transformed into a single-track Gray code by a fairly trivial construction. In order to accomplish this, we simply place each of the n columns side by side and readjust the corresponding sensor placements accordingly as shown in Figure 5.

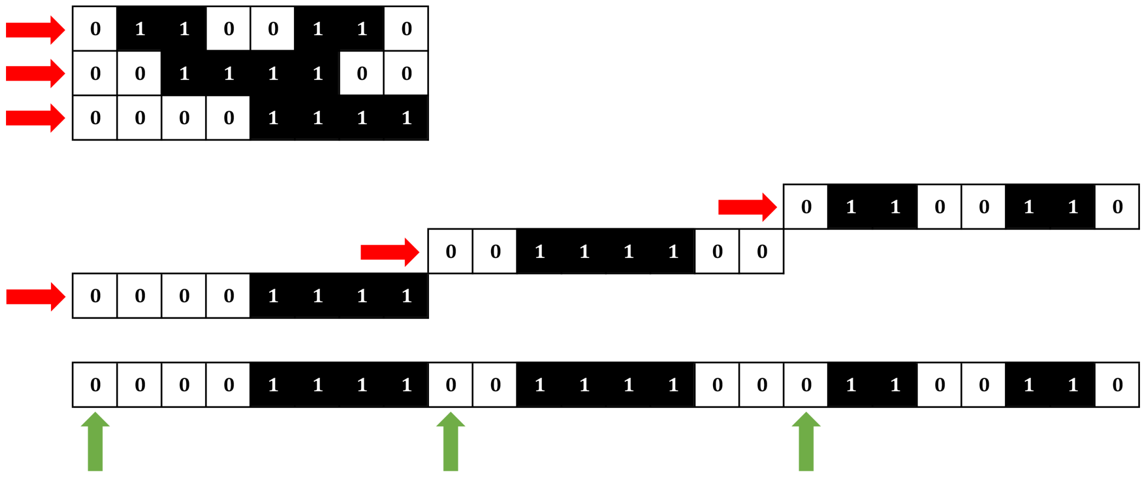


Figure 5: A transformation of the 3-bit binary reflected Gray code to the trivial construction of an acyclic single-track Gray code.

We immediately observe that this code is cyclic, but we also notice that certain codewords will be repeated. In particular, when the last sensor reaches the end of the single-track, codewords will begin repeating. Instead, if we interpret this as an acyclic code for use in a linear encoder, we notice that this particular configuration results in a full-period, single-track Gray code with the same decoding as the binary reflected Gray code. The only other drawback of this construction is its overall length. Thus, shortening the track length is preferable for real-world applications.

We consider ways to shorten the track length while trying to maintain the efficient decoding property. In Figure 6, an arrangement of the columns of the binary reflected Gray code that exploits some of their existing overlap is shown.

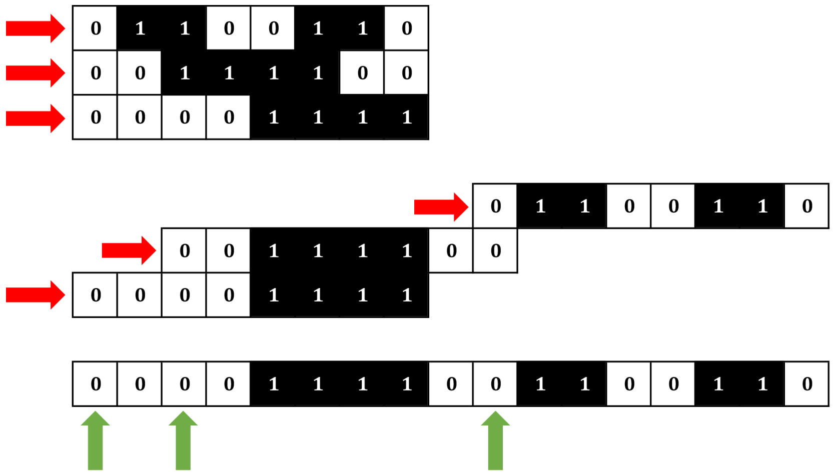


Figure 6: A transformation of the 3-bit binary reflected Gray code that achieves maximal overlap.

Upon further investigation, we notice that we can achieve even more overlap if certain columns are inverted. We prove in Theorem 4.1 that the binary reflected Gray code retains the Gray property even after a certain column is inverted. It is important to note that inverting the k^{th} column of the binary reflected Gray code is equivalent to performing the bitwise XOR operation between $(2^k)_2$ and every codeword, assuming that the columns use zero-based numbering. In Figure 7, we demonstrate how inverting the 0^{th} column allows us to shorten the track length even further.

Refer to Column C of Figure 8 in the Section 6 to see the percentage decrease of our construction with respect to the trivial construction. In general, every other column is inverted, but the starting position depends on the bit-length. This is generalized in Section 4, where we present the constants necessary for similar shortening in larger bit-lengths.

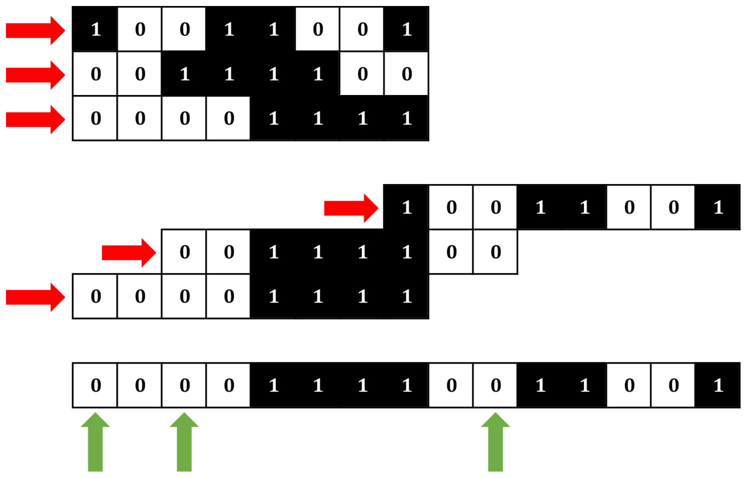


Figure 7: A transformation of the 3-bit binary reflected Gray code with inversion that achieves maximal overlap.

4 Constants

For odd bit-lengths n, the constant is of the form $[0, 0, 1, 0, 1, \ldots, 0, 1, 0, 1]$. Note that this codeword has a base ten value given by

$$\underbrace{2^0 + 2^2 + 2^4 + \cdots + 2^{n-3}}_{\frac{n-1}{2}} = \sum_{i=0}^{\frac{n-3}{2}} 2^{2i}$$

For programming and algorithmic convenience, we present a closed form of its base ten value in Lemma 4.1.

Lemma 4.1 (Constant for Odd n). *For all odd $n \in \mathbb{N}$ such that $n \geq 3$, we have*

$$\sum_{i=0}^{\frac{n-3}{2}} 2^{2i} = \frac{2^{n-1} - 1}{3} \tag{1}$$

Proof. We proceed by induction on n.

Base Case: Let $n = 3$. Substitute and simplify

$$\sum_{i=0}^{\frac{(3)-3}{2}} 2^{2i} = \frac{2^{(3)-1} - 1}{3}$$

$$1 = 1$$

Inductive Step: Suppose that there exists an odd $k \in \mathbb{N}$ such that (1) is true. Then

$$\sum_{i=0}^{\frac{(k+2)-3}{2}} 2^{2i} = \sum_{i=0}^{\frac{k-3}{2}} 2^{2i} + 2^{2(\frac{k-1}{2})}$$

Then, substitute the inductive hypothesis for $\sum_{i=0}^{\frac{k-3}{2}} 2^{2i}$ to obtain

$$= (\frac{2^{k-1}-1}{3}) + 2^{k-1}$$

Simplify and factor to obtain the desired result

$$= \frac{2^{(k+2)-1}-1}{3}$$

□

Similarly, for <u>even</u> bit-lengths n, the constant is of the form $[0,0,1,0,1,\ldots,0,1,0,1,0]$. Note that this codeword has a base ten value given by

$$\underbrace{2^1 + 2^3 + 2^5 + \cdots + 2^{n-3}}_{\frac{n-2}{2}} = \sum_{i=0}^{\frac{n-4}{2}} 2^{2i+1}$$

For programming and algorithmic convenience, we present a closed form of its base ten value in Lemma 4.2.

Lemma 4.2 (Constant for Even n)**.** *For all even $n \in \mathbb{N}$ such that $n \geq 4$, we have*

$$\sum_{i=0}^{\frac{n-4}{2}} 2^{2i+1} = \frac{2^{n-1}-2}{3} \tag{2}$$

Proof. We proceed by induction on n.

Base Case: Let $n = 4$. Substitute and simplify

$$\sum_{i=0}^{\frac{(4)-4}{2}} 2^{2i+1} = \frac{2^{(4)-1}-2}{3}$$

$$2 = 2$$

Inductive Step: Suppose that there exists an even $k \in \mathbb{N}$ such that (2) is true. Then

$$\sum_{i=0}^{\frac{(k+2)-4}{2}} 2^{2i+1} = \sum_{i=0}^{\frac{k-4}{2}} 2^{2i+1} + 2^{2(\frac{k-2}{2})+1}$$

Then, substitute the inductive hypothesis for $\sum_{i=0}^{\frac{k-4}{2}} 2^{2i+1}$ to obtain

$$= (\frac{2^{k-1}-2}{3}) + 2^{k-1}$$

Simplify and factor to obtain the desired result

$$= \frac{2^{(k+2)-1}-2}{3}$$

□

In [1], Frank Gray showed that if g_k is the k^{th} codeword of the binary reflected Gray code in base ten, then for all nonnegative $k \in \mathbb{Z}$ there exists a nonnegative $m \in \mathbb{Z}$ such that

$$g_k \oplus g_{k+1} = 2^m$$

This implies that there is one and only one set bit in the binary expansion of the result.

Theorem 4.1. *If the bitwise XOR operation is performed between a nonnegative constant c and every codeword of the n-bit binary reflected Gray code, then the resulting sequence of codewords continue to differ in one and only one bit.*

Proof. Let c be a nonnegative constant. Suppose that $g_k \oplus g_{k+1} = 2^m$. Consider performing the bitwise XOR operation between c and every codeword in the binary reflected Gray code. Then, the k and $k+1$ codewords will be given by $g_k \oplus c$ and $g_{k+1} \oplus c$. We perform the bitwise XOR between these two codewords to check the number of bits that differ. We associativity and commutativity of the bitwise XOR operator to simplify

$$(g_k \oplus c) \oplus (g_{k+1} \oplus c) = (g_k \oplus g_{k+1}) \oplus (c \oplus c)$$

For all nonnegative $x \in \mathbb{Z}$, we have $x \oplus x = 0$. Thus

$$= (g_k \oplus g_{k+1}) \oplus 0$$

But, we also have $x \oplus 0 = x$, for all nonnegative $x \in \mathbb{Z}$. Then

$$\begin{aligned} &= g_k \oplus g_{k+1} \\ &= 2^m \end{aligned}$$

as required.

$\square$

Remark. It is important to note that performing the bitwise XOR operation between a constant and a code inverts every column of the code that corresponds to a set bit in the constant.

5 Construction

We introduce a novel STGC construction and prove its validity for all bit-lengths n.

Definition 5.1 (Binary Vector)**.** We say that a bit array $[a_0, a_1, \ldots, a_{n-1}]$ is a <u>binary vector</u> if for all $0 \leq i \leq n-1$ we have $a_i \in \{0, 1\}$.

Remark. All binary vectors in this paper are finite.

Definition 5.2 (Consecutive Bits)**.** For all nonnegative $n, m \in \mathbb{Z}$, we say that 0_n denotes a sequence of n zeros and that 1_m denotes a sequence of m ones.

Definition 5.3 (Concatenation)**.** We denote the concatenation of two finite binary vectors B_1 and B_2 by

$$B_1 \bigstar B_2$$

Remark. Note that $\bigstar$ is a binary operator on the set of binary vectors.

We introduce a unique way to represent binary vectors. This representation is inspired by the run length encoding introduced in [15]. Although it was intended for data compression, we find that it helps describe the binary vectors that arise from the BRGC, and consequently, our STGC construction. In particular, we define a sequence that captures the respective run lengths of zeros and ones in a binary vector.

Definition 5.4 (Run Sequence)**.** For all binary vectors

$$B = \mathop{\bigstar}_{i=0}^{n-1} (i \pmod 2)_{a_i} = 0_{a_0} \bigstar 1_{a_1} \bigstar 0_{a_2} \bigstar 1_{a_3} \bigstar \cdots \bigstar (n-2 \pmod 2)_{a_{n-2}} \bigstar (n-1 \pmod 2)_{a_{n-1}}$$

where each $a_i \geq 0$ for $0 \leq i \leq n$, we say that

$$[a_0, a_1, \ldots, a_{n-1}]_R$$

is a run sequence of B. Furthermore, we write

$$[a_0, a_1, \ldots, a_{n-1}]_R = B$$

Remark. It is important to note that a run sequence is distinguished from a binary vector by an R subscript. Also, by definition, the first term in every run sequence describes the number of zeros in the associated binary vector. For example

$$[0, 1, 2, 3]_R = [1, 0, 0, 1, 1, 1]$$

But we have

$$[1, 2, 3]_R = [0, 1, 1, 0, 0, 0]$$

Furthermore, Definition 5.4 leads to the following two relationships for all nonnegative $n, m \in \mathbb{Z}$

$$\begin{aligned} 0_n &= [n]_R \\ 1_m &= [0, m]_R \end{aligned}$$

Lemma 5.1 (Length of a Run Sequence)**.** *Suppose $S = [a_0, a_1, \ldots, a_{n-1}]_R$ is a run sequence. Then the length of the binary vector represented by S is given by*

$$\sum_{i=0}^{n-1} a_i$$

Proof. Let $S = [a_0, a_1, \ldots, a_{n-1}]_R$ be a run sequence. Then, by Definition 5.4, we have that

$$S = \mathop{\bigstar}_{i=0}^{n-1} (i \pmod 2)_{a_i}$$

But, by Definition 5.2, S is simply a concatenation of consecutive sequences of zeros or ones depending on the parity of i. Thus, the length of the binary vector described by S is given by the summation of the terms in S. $\square$

Definition 5.5 (Reverse of a Binary Vector)**.** We denote the reverse of a binary vector B by $\operatorname{rev}(B)$.

Remark. If B is a run sequence instead, $\operatorname{rev}(B)$ denotes the run sequence which corresponds to the reverse of the binary vector represented by B. For example

$$\operatorname{rev}([0,1,0,1,1]) = [1,1,0,1,0]$$

But we also have $[0,1,0,1,1] = [1,1,1,2]_R$ and $[1,1,0,1,0] = [0,2,1,1,1]_R$. Thus

$$\operatorname{rev}([1,1,1,2]_R) = [0,2,1,1,1]_R$$

Lemma 5.2 (Reverse of a Run Sequence)**.** *Given a run sequence $S = [a_0, a_1, \ldots, a_{n-1}]_R$, we have*

$$\operatorname{rev}(S) = \begin{cases} [a_{n-1}, a_{n-2}, \ldots, a_0]_R, & \text{if } n \equiv 1 \pmod 2 \text{ and } a_0 > 0 \\ [0, a_{n-1}, a_{n-2}, \ldots, a_0]_R, & \text{if } n \equiv 0 \pmod 2 \text{ and } a_0 > 0 \\ [0, a_{n-1}, a_{n-2}, \ldots, a_0]_R, & \text{if } n \equiv 1 \pmod 2 \text{ and } a_0 = 0 \\ [a_{n-1}, a_{n-2}, \ldots, a_0]_R, & \text{if } n \equiv 0 \pmod 2 \text{ and } a_0 = 0 \end{cases}$$

Proof. Let $S = [a_0, a_1, \ldots, a_{n-1}]_R$ be a run sequence. Suppose that $n \equiv 1 \pmod 2$. Then, $a_0 > 0$ implies that the first and last term in S refer to the number of consecutive zeros, so the terms in S can simply be reversed. Else, $a_0 = 0$ implies that both the first nonzero term and the last term in S refer to the number of consecutive ones. Thus, a zero should be prepended to the reverse of the terms in S in order to ensure that a_{n-1} represents the number of consecutive ones according to Definition 5.4. Similarly, if $n \equiv 0 \pmod 2$, the logic is reversed. □

Definition 5.6 (Extend Function)**.** We define an extend function to capture the effect of increasing the bit-length of the BRGC by one. A given, existing column will double in length by having its reverse concatenated it to itself. Thus, for a run sequence $S = [a_0, a_1, \ldots, a_{n-1}]_R$, we have

$$\operatorname{extd}(A) := A \bigstar \operatorname{rev}(A)$$

Definition 5.7 (Nested Functions)**.** Given a function $f(x)$, we define the n^{th} functional power inductively for all $n \in \mathbb{N}$ by

$$f^{n+1}(x) = f \circ f^n = f^n \circ f$$

Lemma 5.3. *Suppose that $S = [2^a, 2^a]_R$ for a nonnegative $a \in \mathbb{Z}$. Then, for all $n \in \mathbb{N}$, we have*

$$\operatorname{extd}^n(S) = [2^a, \underbrace{2^{a+1}, \ldots, 2^{a+1}}_{2^n - 1}, 2^a]_R \tag{3}$$

Proof. We proceed by induction on n.

Base Case: Let $n = 1$ and $S = [2^a, 2^a]_R$ for a nonnegative $a \in \mathbb{Z}$. Then

$$\text{extd}^1(S) = S \bigstar \text{rev}(S)$$

Because S has an even number of terms, by Lemma 5.2, we have

$$= [2^a, 2^a]_R \bigstar [0, 2^a, 2^a]_R$$

It can easily be verified that the last term of the first run sequence and the second term of the second run sequence both describe the number of consecutive ones. Thus

$$= [2^a, 2^{a+1}, 2^a]_R$$

as required.

Inductive Step: Let $S = [2^a, 2^a]_R$ for a nonnegative $a \in \mathbb{Z}$ and suppose that there exists $k \in \mathbb{N}$ such that (3) is true. By Definition 5.7, we have

$$\text{extd}^{k+1} = \text{extd} \circ \text{extd}^k$$

Then, substitute the inductive hypothesis for extd^k to obtain

$$= \text{extd}([2^a, \underbrace{2^{a+1}, \ldots, 2^{a+1}}_{2^k - 1}, 2^a]_R)$$

By Definition 5.6, this becomes

$$= [2^a, \underbrace{2^{a+1}, \ldots, 2^{a+1}}_{2^k - 1}, 2^a]_R \bigstar \text{rev}([2^a, \underbrace{2^{a+1}, \ldots, 2^{a+1}}_{2^k - 1}, 2^a]_R)$$

Observe that the parity of the number of terms in the run sequence we wish to reverse is given by

$$1 + (2^k - 1) + 1 \equiv 1 \pmod 2 \tag{4}$$

Thus, according to Lemma 5.2

$$= [2^a, \underbrace{2^{a+1}, \ldots, 2^{a+1}}_{2^k - 1}, 2^a]_R \bigstar [2^a, \underbrace{2^{a+1}, \ldots, 2^{a+1}}_{2^k - 1}, 2^a]_R$$

From (4) we conclude that the last term of the first run sequence and the first term of the second run sequences both represent the number of consecutive zeros. It readily follows that

$$\begin{aligned} &= [2^a, \underbrace{2^{a+1}, \ldots, 2^{a+1}}_{2^k-1}, 2^{a+1}, \underbrace{2^{a+1}, \ldots, 2^{a+1}}_{2^k-1}, 2^a]_R \\ &= [2^a, \underbrace{2^{a+1}, \ldots, 2^{a+1}}_{2^{k+1}-1}, 2^a]_R \end{aligned}$$

as required. $\square$

Theorem 5.1 (Run Sequence of BRGC)**.** *The j^{th} column of the n-bit BRGC is given by*

$$[2^j, \underbrace{2^{j+1}, \ldots, 2^{j+1}}_{2^{n-j-1}-1}, 2^j]_R \tag{5}$$

where $0 \leq j \leq n-1$ and $n \geq 2$

Remark. The column at $j = 0$ contains the least significant bit of the BRGC.

Proof. We proceed by double induction on j and n.

Base Case: Let $j = 0$ and $n = 2$. The 0th column of the BRGC is given by

$$[0, 1, 1, 0] = [1, 2, 1]_R = [2^0, \underbrace{2^1}_{2^1-1}, 2^0]_R$$

Inductive Step #1: Set $j = 0$ and suppose that there exists $k \in \mathbb{Z}$ such that $k \geq 2$ and (5) is true. We construct the run sequence for the 0th column of the $k+1$-bit BRGC. According to the definition of the BRGC, we reverse the 0th column and concatenate it to itself

$$[1, \underbrace{2, \ldots, 2}_{2^{k-1}-1}, 1]_R \bigstar \mathrm{rev}([1, \underbrace{2, \ldots, 2}_{2^{k-1}-1}, 1]_R)$$

Observe that the parity of the number of terms in the run sequence we wish to reverse is given by

$$1 + (2^{k-1} - 1) + 1 \equiv 1 \pmod 2 \tag{6}$$

Thus, according to Lemma 5.2

$$[1, \underbrace{2, \ldots, 2}_{2^{k-1}-1}, 1]_R \bigstar \mathrm{rev}([1, \underbrace{2, \ldots, 2}_{2^{k-1}-1}, 1]_R) = [1, \underbrace{2, \ldots, 2}_{2^{k-1}-1}, 1]_R \bigstar [1, \underbrace{2, \ldots, 2}_{2^{k-1}-1}, 1]_R$$

From (6) we conclude that the last term of the first run sequence and the first term of the second run sequences both represent the number of consecutive zeros. It readily follows that

$$= [1, \underbrace{2, \ldots, 2}_{2^{k-1}-1}, 2, \underbrace{2, \ldots, 2}_{2^{k-1}-1}, 1]_R$$

$$= [1, \underbrace{2, \ldots, 2}_{2^{(k+1)-1}-1}, 1]_R$$

This completes the induction on n. We proceed by induction on j.

Inductive Step #2: We let n be arbitrarily chosen such that $n \geq 2$. Suppose that there exists a nonnegative $l \in \mathbb{Z}$ such that $l < n-1$ and (5) is true. Then, the l^{th} column is given by

$$[2^l, \underbrace{2^{l+1}, \ldots, 2^{l+1}}_{2^{n-l-1}-1}, 2^l]_R \tag{7}$$

Momentarily consider the l^{th} column of the $l+1$-bit BRGC. Recall that as a result of zero-based numbering, this column contains the most significant bit. Then, the l^{th} column of this BRGC, by definition, is half zeros and half ones.

$$[0_{2^l}, 1_{2^l}] = [2^l, 2^l]_R$$

Observe that these bits coincide with our assumption in (7) as expected. Similarly, the $l+1$ column of the $l+2$-bit BRGC is given by

$$[2^{l+1}, 2^{l+1}]_R$$

Note that the $l+1$ column is defined in the n-bit BRGC because we have $l < n-1$ so that $l+1 \leq n-1$. Further increases in the bit-length of the BRGC will only repeatedly reverse and concatenate this column to itself as in the previous induction on n. By Definition 5.6, this can be represented by repeated applications of the extend function. Recall that the extend function doubles the length of its input. We obtain the length of the binary vector described by $[2^{l+1}, 2^{l+1}]_R$ from Lemma 5.1 to be $2^{l+1} + 2^{l+1} = 2^{l+2}$. It immediately follows that the extend operation can be performed exactly $\frac{2^n}{2^{l+2}} = 2^{n-l-2}$ times, which is nonnegative because $l < n-1$ implies $n-l-2 \geq 0$. We apply Lemma 5.3 to calculate

$$\text{extd}^{2^{n-l-2}}([2^{l+1}, 2^{l+1}]_R) = [2^{l+1}, \underbrace{2^{l+2}, \ldots, 2^{l+2}}_{2^{n-(l+1)-1}-1}, 2^{l+1}]_R$$

as required. □

Definition 5.8 (Logical Complement)**.** For all nonnegative $a \in \mathbb{Z}$, we say that $\bar{a}$ denotes the logical complement of a.

Lemma 5.4 (Logical Complement of a Run Sequence)**.** *Let* $[a_0, a_1, \ldots, a_{n-1}]_R$ *be a run sequence such that* $a_0 > 0$*. Then*

$$\overline{[a_0, a_1, \ldots, a_{n-1}]_R} = [0, a_0, a_1, \ldots, a_{n-1}]_R$$

and

$$\overline{[0, a_0, a_1, \ldots, a_{n-1}]_R} = [a_0, a_1, \ldots, a_{n-1}]_R$$

Proof. Let $[a_0, a_1, \ldots, a_{n-1}]_R$ be a run sequence such that $a_0 > 0$. Then, by Definition 5.4, we have

$$[a_0, a_1, \ldots, a_{n-1}]_R = \bigstar_{i=0}^{n-1} (i \pmod 2)_{a_i}$$

Then, inverting every bit yields

$$= \bigstar_{i=0}^{n-1} (i + 1 \pmod 2)_{a_i}$$

$$= 0_0 \bigstar \left(\bigstar_{i=0}^{n-1} (i + 1 \pmod 2)_{a_i}\right)$$

Refer to Definition 5.4 and simplify

$$= [0, a_0, a_1, \ldots, a_{n-1}]_R$$

as required. The converse immediately follows because all steps are reversible. □

We introduce a way to decompose a multi-track code into a single-track Gray code in Definition 5.9.

Definition 5.9 (Partial Run Sequence Decomposition (PRSD))**.** Consider the following three run sequences, where all $a \geq 0$ and all $b \geq 0$.

$$\begin{aligned} A &= [a_0, a_1, \ldots, a_{n-2}, a_{n-1}]_R \\ B_1 &= [b_0, b_1, \ldots, b_{m-1}]_R \\ B_2 &= \overline{B_1} = [0, b_0, b_1, \ldots, b_{m-1}]_R \end{aligned}$$

Now consider the following run sequence

$$C = [a_0, a_1, \ldots, a_{n-3}, \max\{a_{n-2}, b_0\}, \max\{a_{n-1}, b_1\}, b_2, \ldots, b_{m-1}]_R$$

We say that $A \perp B$ denotes the PRSD of C *s.t.*

$$C = \begin{cases} A \perp B_1, & \text{if } n \equiv 1 \pmod 2 \\ A \perp B_2, & \text{if } n \equiv 0 \pmod 2 \end{cases}$$

where C is defined by sensor placement $\{0, (\sum_{k=0}^{n-2} a_k) - b_0\}_S$

Remark. If we placed a sensor at every specified sensor location on C, where C uses zero-based numbering, then A and B are the run sequences that will be obtained by moving each sensor the length of the binary vectors described by A and B, respectively. We consider cases where the lengths of the binary vectors described by A and B are equal. Also, the intuition and practical application of this definition usually works in the opposite direction: we begin with a single run sequence C and try to find the PRSD of that sequence.

In order to demonstrate the previous definition, we present the following numerical example:
Let $C = [1, 2, 2, 2, 1, 1, 1, 3]_R$ be a run sequence. Then a possible PRSD of C is shown below

$$C = A \perp B$$

$$[1, 2, 2, 2, 1, 1, 1, 3]_R = [1, 2, 2, 2]_R \perp [0, 2, 1, 1, 1, 3]_R$$

defined by sensor placement $\{0, 5\}_S$.

Remark (Generalized Partial Run Sequence Decomposition (GPRSD))*.* We naturally extend Definition 5.9 to allow for more than two run sequences. Simply treat $\perp$ as a binary operator on the set of finite binary vectors. It is important to note that the number of sensors is always equal to the number of run sequences in the respective GPRSD.

Definition 5.10 (Single-Track Gray Code)**.** We say that a finite binary vector B is a n-bit STGC with a specified sensor placement $\{a_0, a_1, \ldots, a_{n-1}\}_S$ if and only if B has a GPRSD into n binary vectors of the same length with the given sensor placement where successive codewords are Gray with one another.

Remark. Recall that if the last codeword is Gray with the first codeword, then the code is said to be cyclic. Otherwise, it is said to be acyclic. Furthermore, the Gray property requires that successive codewords differ by only one bit. Equivalently, there exists a nonnegative $m \in \mathbb{Z}$ such that $(g_k)_2 \oplus (g_{k+1})_2 = (2^m)_2$, where $0 \leq m \leq n-1$.

Theorem 5.2 (Construction)**.** *Given a bit-length n, then there exists a STGC with a run sequence of the form*

$$G = [2^{n-1}, 2^{n-1}, \underbrace{2^{n-2}, \dots, 2^{n-2}}_{2^2-1=3}, \underbrace{2^{n-3}, \dots, 2^{n-3}}_{2^3-1=7}, \dots, \underbrace{2^1, \dots, 2^1}_{2^{n-1}-1}, 1]_R$$

where the placement of the k^{th} sensor S is given by

$$S_k = 2^{n-k-1}(2^k(2k-3)+3) \text{ for } 0 \leq k \leq n-1$$

Proof. We proceed by induction on the number of sensors n.

Case 1 (n odd)

Base Case: Let $n = 3$. Consider $[4,4,2,2,2,1]_R$ defined by sensor placement $\{0,2,7\}_S$. A GPRSD of this run sequence is given by

$$[4,4] \perp [2,4,2] \perp [1,2,2,2,1]$$

It is easy to verify that this GPRSD is a STGC by applying Definition 5.10.

Inductive Hypothesis: Let $n = k$. Suppose that

$$G = [2^{k-1}, 2^{k-1}, \underbrace{2^{k-2}, \dots, 2^{k-2}}_{3}, \underbrace{2^{k-3}, \dots, 2^{k-3}}_{7}, \dots, \underbrace{2, \dots, 2}_{2^{k-1}-1}, 1]_R \tag{8}$$

where the placement of the j^{th} sensor S is given by

$$S_j = 2^{k-j-1}(2^j(2j-3)+3) \text{ for } 0 \leq j \leq k-1 \tag{9}$$

and

$$D = [2^{k-1}, 2^{k-1}]_R \perp [2^{k-2}, 2^{k-1}, 2^{k-2}]_R \perp [0, 2^{k-3}, 2^{k-2}, 2^{k-3}]_R \perp \cdots$$
$$\cdots \perp [0, 4, \underbrace{8, \dots, 8,}_{2^{k-3}-1} 4]_R \perp [2, \underbrace{4, \dots, 4,}_{2^{k-2}-1} 2]_R \perp [0, 1, \underbrace{2, \dots, 2,}_{2^{k-1}-1} 1]_R \quad (10)$$

denotes a GPRSD of G, which implies that every sensor reads exactly one unique column of the k-bit BRGC as outlined in Lemma 5.1.

Inductive Step: Observe that multiplying every term in every run sequence in (10) by a factor of 2 yields

$$D' = [2^{k}, 2^{k}]_R \perp [2^{k-1}, 2^{k}, 2^{k-1}]_R \perp [0, 2^{k-2}, 2^{k-1}, 2^{k-2}]_R \perp \cdots$$
$$\cdots \perp [0, 8, \underbrace{16, \dots, 16,}_{2^{k-3}-1} 8]_R \perp [4, \underbrace{8, \dots, 8,}_{2^{k-2}-1} 4]_R \perp [0, 2, \underbrace{4, \dots, 4,}_{2^{k-1}-1} 2]_R \quad (11)$$

Also, observe that if we multiply the placement of the j^{th} sensor in (9) by a factor of 2, we obtain

$$S'_j = 2^{k-j}(2^j(2j-3)+3) \text{ for } 0 \le j \le k-1 \quad (12)$$

It can be easily seen that every sensor will read exactly one unique column of the $k+1$-bit BRGC expect for the LSB column from the binary vector described by (11) and (12). Note that after multiplying both the GPRSD (11) and the corresponding sensor placement (12) by a factor of 2, the STGC given by (8) from the inductive hypothesis has every term in every one of its run sequence multiplied by a factor of 2. This can easily be verified by Definition 5.9. Thus, we conclude that

$$G' = [2^k, 2^k, \underbrace{2^{k-1}, \dots, 2^{k-1}}_{3}, \underbrace{2^{k-2}, \dots, 2^{k-2}}_{7}, \dots, \underbrace{4, \dots, 4,}_{2^{k-1}-1} 2]_R \quad (13)$$

where the placement of the j^{th} sensor S is given by

$$S'_j = 2^{k-j}(2^j(2j-3)+3) \text{ for } 0 \le j \le k-1$$

In order to complete the induction, we need to construct G'' (the new STGC for $n = k+1$) as a PRSD of G' and $[1, \underbrace{2, \dots, 2,}_{2^k-1} 1]_R$ or $\overline{[1, \underbrace{2, \dots, 2,}_{2^k-1} 1]_R} = [0, 1, \underbrace{2, \dots, 2,}_{2^k-1} 1]_R$ depending on the number of terms in G'. Determining the number of terms in G' will offer insight into whether the last term, namely 2, describes the number of consecutive zeros or ones. In particular, if we let t denote the number of terms in G', calculating $\overline{t \pmod 2}$ will immediately yield the desired result. This can easily be verified by recalling the properties of run sequences delineated in Definition 5.4. From (13), we conclude that the number of terms in the run sequence of G' taken modulo 2 is given by

$$t \equiv \overline{2 + \sum_{i=2}^{k-1} (2^i - 1) + 1 \pmod 2}$$
$$t \equiv \overline{\sum_{i=2}^{k-1} 1 + 1 \pmod 2}$$
$$t \equiv \overline{((k-1) - 2 + 1) + 1 \pmod 2}$$
$$t \equiv \overline{k + 1 \pmod 2}$$

But k is odd implies that $k + 1 \equiv 0 \pmod 2$, and that $t = \overline{0} = 1$. Thus, according to Definition 5.9

$$G'' = G' \perp [1, \underbrace{2, \ldots, 2,}_{2^k - 1} 1]_R$$
$$G'' = ([2^k, 2^k, \underbrace{2^{k-1}, \ldots, 2^{k-1}}_{3}, \ldots, \underbrace{4, \ldots, 4}_{2^{k-1}-1}, 2]_R) \perp [1, \underbrace{2, \ldots, 2,}_{2^k - 1} 1]_R$$
$$G'' = [2^k, 2^k, \underbrace{2^{k-1}, \ldots, 2^{k-1}}_{3}, \ldots, \underbrace{4, \ldots, \max\{4,1\}}_{2^{k-1}-1}, \underbrace{\max\{2,2\}, \ldots, 2,}_{2^k - 1} 1]_R$$
$$G'' = [2^k, 2^k, \underbrace{2^{k-1}, \ldots, 2^{k-1}}_{3}, \ldots, \underbrace{4, \ldots, 4,}_{2^{k-1}-1} \underbrace{2, \ldots, 2,}_{2^k - 1} 1]_R$$
$$G'' = [2^{(k+1)-1}, 2^{(k+1)-1}, \underbrace{2^{(k+1)-2}, \ldots, 2^{(k+1)-2}}_{3}, \ldots, \underbrace{2, \ldots, 2,}_{2^{(k+1)-1}-1} 1]_R$$

as required.

Case 2 (n even)

Base Case: Let $n = 2$. Consider $[2, 2, 1]_R$ defined by sensor placement $\{0, 1\}_S$. The GPRSD of this run sequence is given by

$$[2, 2] \perp [1, 2, 1]$$

It is easy to verify that this GPRSD is a STGC by applying Definition 5.10.

Inductive Hypothesis: Let $n = k$. Suppose that

$$G = [2^{k-1}, 2^{k-1}, \underbrace{2^{k-2}, \ldots, 2^{k-2}}_{3}, \underbrace{2^{k-3}, \ldots, 2^{k-3}}_{7}, \ldots, \underbrace{2, \ldots, 2}_{2^{k-1}-1}, 1]_R \tag{14}$$

where the placement of the j^{th} sensor S is given by

$$S_j = 2^{k-j-1}(2^j(2j-3)+3) \text{ for } 0 \leq j \leq k-1 \tag{15}$$

and

$$D = [2^{k-1}, 2^{k-1}]_R \perp [2^{k-2}, 2^{k-1}, 2^{k-2}]_R \perp [0, 2^{k-3}, 2^{k-2}, 2^{k-3}]_R \perp \cdots$$
$$\cdots \perp [4, \underbrace{8, \ldots, 8}_{2^{k-3}-1}, 4]_R \perp [0, 2, \underbrace{4, \ldots, 4}_{2^{k-2}-1}, 2]_R \perp [1, \underbrace{2, \ldots, 2}_{2^{k-1}-1}, 1]_R \tag{16}$$

denotes the GPRSD of G, which implies that every sensor reads exactly one unique column of the k-bit BRGC as outlined in Lemma 5.1.

Induction Step: Similarly, we use the same technique used in Case 1 to obtain

$$D' = [2^k, 2^k]_R \perp [2^{k-1}, 2^k, 2^{k-1}]_R \perp [0, 2^{k-2}, 2^{k-1}, 2^{k-2}]_R \perp \cdots$$
$$\cdots \perp [8, \underbrace{16, \ldots, 16}_{2^{k-3}-1}, 8]_R \perp [0, 4, \underbrace{8, \ldots, 8}_{2^{k-2}-1}, 4]_R \perp [2, \underbrace{4, \ldots, 4}_{2^{k-1}-1}, 2]_R \tag{17}$$

$$G' = [2^k, 2^k, \underbrace{2^{k-1}, \ldots, 2^{k-1}}_{3}, \underbrace{2^{k-2}, \ldots, 2^{k-2}}_{7}, \ldots, \underbrace{4, \ldots, 4}_{2^{k-1}-1}, 2]_R \tag{18}$$

where the placement of the j^{th} sensor S is given by

$$S'_j = 2^{k-j}(2^j(2j-3)+3) \text{ for } 0 \leq j \leq k-1 \tag{19}$$

In order to complete the induction, we need to construct G'' (the new STGC for $n = k+1$) as the PRSD of G' and $[1, \underbrace{2, \ldots, 2}_{2^k-1}, 1]_R$ or $\overline{[1, \underbrace{2, \ldots, 2}_{2^k-1}, 1]_R} = [0, 1, \underbrace{2, \ldots, 2}_{2^k-1}, 1]_R$ depending on the number of terms in G'. Recall that we let t denote the number of terms in G'. From (18), we conclude that the number of terms in the run sequence of G' taken modulo 2 is given by

$$\begin{aligned}
t &\equiv \overline{2 + \sum_{i=2}^{k-1}(2^i - 1) + 1 \pmod 2} \\
t &\equiv \overline{\sum_{i=2}^{k-1} 1 + 1 \pmod 2} \\
t &\equiv \overline{((k-1) - 2 + 1) + 1 \pmod 2} \\
t &\equiv \overline{k + 1 \pmod 2}
\end{aligned}$$

But k is even implies that $k + 1 \equiv 1 \pmod 2$, and that $t = \overline{1} = 0$. Thus, according to Definition 5.9

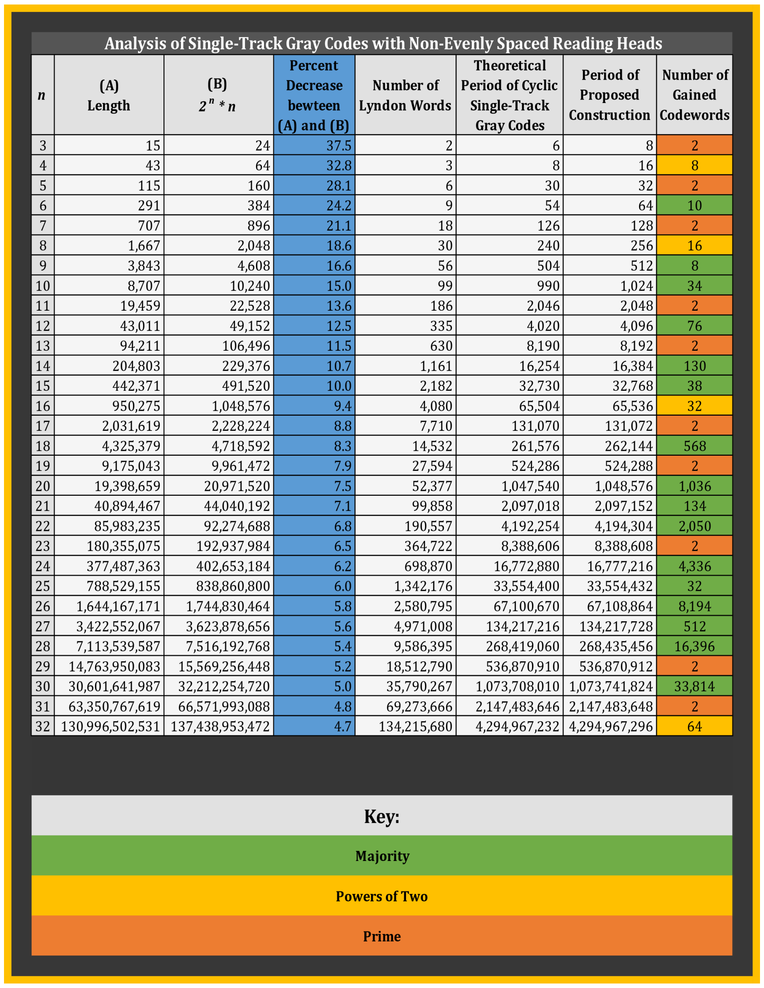
$$\begin{aligned}
G'' &= G' \perp [0, 1, \underbrace{2, \ldots, 2,}_{2^k - 1} 1]_R \\
G'' &= ([2^k, 2^k, \underbrace{2^{k-1}, \ldots, 2^{k-1}}_{3}, \ldots, \underbrace{4, \ldots, 4,}_{2^{k-1}-1} 2]_R) \perp [0, 1, \underbrace{2, \ldots, 2,}_{2^k - 1} 1]_R \\
G'' &= [2^k, 2^k, \underbrace{2^{k-1}, \ldots, 2^{k-1}}_{3}, \ldots, \underbrace{4, \ldots, \max\{4, 1\}}_{2^{k-1}-1}, \underbrace{\max\{2, 2\}, \ldots, 2,}_{2^k - 1} 1]_R \\
G'' &= [2^k, 2^k, \underbrace{2^{k-1}, \ldots, 2^{k-1}}_{3}, \ldots, \underbrace{4, \ldots, 4,}_{2^{k-1}-1} \underbrace{2, \ldots, 2,}_{2^k - 1} 1]_R \\
G'' &= [2^{(k+1)-1}, 2^{(k+1)-1}, \underbrace{2^{(k+1)-2}, \ldots, 2^{(k+1)-2}}_{3}, \ldots, \underbrace{2, \ldots, 2,}_{2^{(k+1)-1}-1} 1]_R
\end{aligned}$$

as required. □

Remark. We use the modified BRGC with the constant defined in Section 3 because the least significant bit of the $n + 1$-bit BRGC is inverted during the induction step whenever $n \equiv 1 \pmod 2$.

6 Discussion

The results of our work are summarized in Figure 8. Column A represents the overall track length of our general STGC construction while Column B represents the overall track length of the trivial construction investigated in Section 3. The percent decrease in the overall track length of the STGC between the trivial construction and our construction is displayed in Column C. We can see that even for 20-bit codes, there is a 7.5% decrease in overall track length. This is a sizable step in achieving a maximally compressed single track, which we leave as an open problem in Section 8. The last column demonstrates the number of additional codewords that would be gained from our proposed STGC construction compared to cyclic STGCs of the same bit-length introduced in Section 2.4. It is important to note that the data used for calculations were based on theoretical maximal periods. In fact, some STGCs of these lengths have not yet been constructed. We clearly see that the number of codewords gained is insignificant for prime bit-lengths as well as for bit-lengths that are powers of two. In all other cases, which are highlighted in green, the gains are substantially larger.



Analysis of Single-Track Gray Codes with Non-Evenly Spaced Reading Heads

n	(A) Length	(B) $2^n * n$	Percent Decrease bewteen (A) and (B)	Number of Lyndon Words	Theoretical Period of Cyclic Single-Track Gray Codes	Period of Proposed Construction	Number of Gained Codewords
3	15	24	37.5	2	6	8	2
4	43	64	32.8	3	8	16	8
5	115	160	28.1	6	30	32	2
6	291	384	24.2	9	54	64	10
7	707	896	21.1	18	126	128	2
8	1,667	2,048	18.6	30	240	256	16
9	3,843	4,608	16.6	56	504	512	8
10	8,707	10,240	15.0	99	990	1,024	34
11	19,459	22,528	13.6	186	2,046	2,048	2
12	43,011	49,152	12.5	335	4,020	4,096	76
13	94,211	106,496	11.5	630	8,190	8,192	2
14	204,803	229,376	10.7	1,161	16,254	16,384	130
15	442,371	491,520	10.0	2,182	32,730	32,768	38
16	950,275	1,048,576	9.4	4,080	65,504	65,536	32
17	2,031,619	2,228,224	8.8	7,710	131,070	131,072	2
18	4,325,379	4,718,592	8.3	14,532	261,576	262,144	568
19	9,175,043	9,961,472	7.9	27,594	524,286	524,288	2
20	19,398,659	20,971,520	7.5	52,377	1,047,540	1,048,576	1,036
21	40,894,467	44,040,192	7.1	99,858	2,097,018	2,097,152	134
22	85,983,235	92,274,688	6.8	190,557	4,192,254	4,194,304	2,050
23	180,355,075	192,937,984	6.5	364,722	8,388,606	8,388,608	2
24	377,487,363	402,653,184	6.2	698,870	16,772,880	16,777,216	4,336
25	788,529,155	838,860,800	6.0	1,342,176	33,554,400	33,554,432	32
26	1,644,167,171	1,744,830,464	5.8	2,580,795	67,100,670	67,108,864	8,194
27	3,422,552,067	3,623,878,656	5.6	4,971,008	134,217,216	134,217,728	512
28	7,113,539,587	7,516,192,768	5.4	9,586,395	268,419,060	268,435,456	16,396
29	14,763,950,083	15,569,256,448	5.2	18,512,790	536,870,910	536,870,912	2
30	30,601,641,987	32,212,254,720	5.0	35,790,267	1,073,708,010	1,073,741,824	33,814
31	63,350,767,619	66,571,993,088	4.8	69,273,666	2,147,483,646	2,147,483,648	2
32	130,996,502,531	137,438,953,472	4.7	134,215,680	4,294,967,232	4,294,967,296	64

Key:

🟩 Majority

🟨 Powers of Two

🟧 Prime

Figure 8: A table of the several important characteristics of our construction.

7 Conclusion

The introduced general STGC construction encodes a maximal 2^n codewords for all bit-lengths n. This is made possible through the implementation of sensors with uneven spacing as opposed to the traditional even spacing of the BRGC. In our construction, the sensors are exponentially distributed. As a result, sensors are more densely packed on one side of the single-track than the other. These sensor arrangements may be preferable in some software or hardware applications. Furthermore, our code can be efficiently decoded by utilizing only logical bitwise operators. This is, in fact, one of the main advantages over traditional cyclic STGCs for which a decoding scheme is not yet known. Consequently, decoding can occur directly within electronics through the use of logic gates, eliminating the need for any computer computations. This makes our code particularly useful for real-time applications or any other situation where using a look-up table would be computationally expensive. When a high resolution STGC is required, and other constructions either lack the sensor to resolution ratio or the desired decoding efficiency, we hope our general construction would be a viable alternative.

8 Future Research and Extensions

A general construction for all cyclic STGCs would be a major breakthrough. The discovered STGC of maximal length shows that it is possible to create STGCs with uneven sensor spacing. Further investigation on whether it is possible to reduce the overall track length of a code with unevenly spaced sensors would also be a major development. We propose finding a cyclic STGC with uneven spacing as an important open problem. A general construction of the like that can also generalize the positioning of the sensors to achieve an arbitrarily spaced code would be a practical and flexible construction. The researched STGCs dealt solely with binary codewords, but they could have been extended to account for other radices. Although the use of these k-ary codes in the real-world is limited, they might offer insight into the fields of theoretical mathematics and computer science.

References

[1] F. Gray, "Pulse code communication," Mar. 17, 1953. US Patent 2,632,058.

[2] D. E. Knuth, *The Art of Computer Programming: Combinatorial Algorithms, Part 1.* Addison-Wesley Professional, 1st ed., 2011.

[3] G. R. Stibitz, "Binary counter," Jan. 12, 1943. US Patent 2,307,868.

[4] S. Cormier-Iijima, "Alternative combinatorial gray codes," Dec. 17, 2010.

[5] C. Savage, "A survey of combinatorial gray codes," *SIAM Review*, vol. 39, no. 4, pp. 605–629, 1997.

[6] E. N. Gilbert, "Gray codes and paths on the n-cube," *The Bell System Technical Journal*, vol. 37, pp. 815–826, May 1958.

[7] T. Etzion and K. G. Paterson, "Near optimal single-track gray codes," *IEEE Transactions on Information Theory*, vol. 42, no. 3, pp. 779–789, 1996.

[8] M. Schwartz and T. Etzion, "The structure of single-track gray codes," *Information Theory, IEEE Transactions on*, vol. 45, pp. 2383–2396, Nov. 1999.

[9] A. Hiltgen, K. Paterson, and M. Brandestini, "Single-track gray codes," *Information Theory, IEEE Transactions on*, vol. 42, pp. 1555–1561, Sept. 1996.

[10] T. Sillke, "Gray-codes with few tracks," Mar. 1, 1993.

[11] N. B. Spedding, "A position encoder," Oct. 28, 1994. New Zealand Provisional Patent 264738.

[12] R. C. Lyndon, "On burnside's problem," *Transactions of the American Mathematical Society*, vol. 77, no. 2, pp. 202–215, 1954.

[13] H. Fredricksen and I. J. Kessler, "An algorithm for generating necklaces of beads in two colors," *Discrete Mathematics*, vol. 61, no. 2–3, pp. 181–188, 1986.

[14] J. Arndt, *Matters Computational: Ideas, Algorithms, Source Code.* New York, NY, USA: Springer-Verlag New York, Inc., 1^{st} ed., 2010.

[15] E. L. Hauck, "Data compression using run length encoding and statistical encoding," Dec. 2, 1986. US Patent 4,626,829.