The Construction of Missile Guidance Codes Resistant to Random Interference

By A. R. ECKLER

(Manuscript received February 8, 1960)

Many types of missiles are guided by a finite set of distinct commands radioed from the ground in the form of a time-sequence of RF *pulses. The command information is contained in the $n - 1$ time spacings between successive pulses in a group of size n, and is encoded and decoded by means of multitapped delay lines combined with* AND *gates. This paper discusses the problem of encoding command information (i.e., selecting the time spacings between pulses) so that m false pulses ($m \leqq n - 2$) cannot combine with the n true pulses in any way to form a false command. Although it is very easy to state the restrictions that must be imposed on the time spacings between the pulses in the different commands, no general methods exist for finding, among codes satisfying these restrictions, those codes in which the longest command is as short as possible. This paper presents certain lower bounds, together with a few empirically derived codes approaching these lower bounds. The relationship between these codes and the well-known error-correcting binary codes of information theory is discussed in an appendix.*

I. INTRODUCTION

Many types of missiles are guided by commands consisting of a time sequence of RF pulses. The missile receiver is capable of receiving not only the true command pulses, but also any interfering RF pulses emanating from other radars in the vicinity (friendly interference) or deliberately generated by the enemy (enemy interference). The purpose of this paper is to suggest command-encoding methods that minimize the effects of this interference. Greatest emphasis is placed on random interference models, because it is difficult to make realistic general assumptions about the behavior of an enemy who knows something about the code structure.

The following sections may convey the impression that the construc-

tion of a code invulnerable to random pulse interference is the only factor entering into the choice of a code. This is not meant to be the case; accuracy and reliability of decoding equipment, weight in the missile receiver, power considerations at the transmitter and missile dynamics and maneuverability also play very important roles. However, it will be made clear to the reader that code invulnerability should not be completely ignored, and that reasonable guideposts of design are available.

In order to fix ideas, the codes presented in this paper are discussed in terms of missile guidance. However, the reader will note that other applications are possible. In particular, the codes described here can be used in radio communications to provide protection against the mutual interference of many communications transmitters operating in the same area on the same frequency.

1.1 *The Engineering Background of the Problem*

In many missile guidance systems, command information is sent to the missile by means of RF pulses. Specifically, the information is contained in the time spacings between successive pulses in a group of two or more pulses. Successive groups of pulses are spaced much more widely apart because of average power limitations on the magnetron.

There are many physical devices suitable for encoding and decoding commands. This paper discusses a typical device for encoding discrete commands, such as "yaw left one unit" or "pitch down one unit." These commands are added up in the missile receiver until the desired correction has been achieved. Discrete commands can also be used for such functions as turning rocket motors on or off or destroying the missile.

A discrete command is conveniently decoded in the missile receiver by means of a lumped-parameter or distributed-parameter delay line tapped at several points (see Fig. 1). The earlier pulses in an incoming sequence are delayed the correct amounts by the delay line in order that the inputs to the AND gate be simultaneously activated.

In practice, it is sufficient that the several AND gate inputs be activated within a short time interval. Let τ denote the maximum possible range of input times (i.e., the difference in time between the earliest and latest input) that still activates the AND gate. Because of τ (which is called the discrete command tolerance), one must be careful not to let two distinct commands have similar encodings. It is shown in the Appendix that an adequate separation of different commands is always achieved if one assigns time spacings in integral multiples of the discrete command tolerance τ.

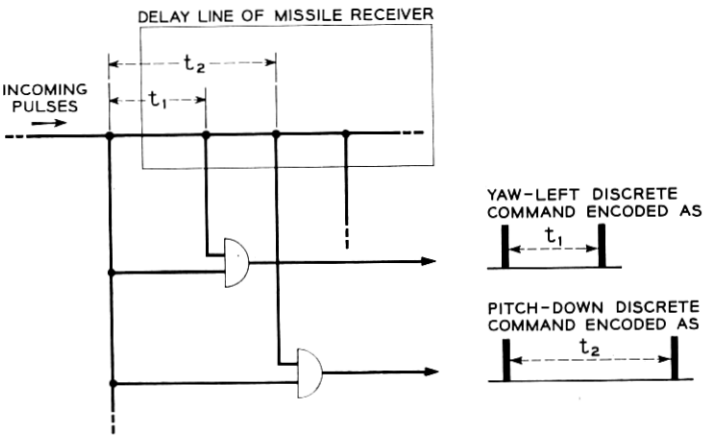

Fig. 1 — Method of decoding discrete command by means of distributed-parameter delay line tapped at several points.

It is clear that only two pulses are needed to encode a command. However, in order to obtain a greater degree of security against false pulses, it is necessary to use three or more pulses per command. The mathematical problem discussed in this paper is the construction of such security codes.

The same delay line and AND gate can be used in the communications problem already mentioned. Here the group of pulses plays the role of an address code rather than a command. Each communications transmitter uses a specific address code as a generalized "pulse" with which to communicate information; thus, different transmitters can use the same frequency without interfering with each other.

1.2 *A Criterion for Judging Code Effectiveness*

The invulnerability of codes to false pulses is measured by the following simple criterion: the *maximum* number of false pulses that cannot interfere with a code group (either by changing the position of a command or producing a different command), no matter how these false pulses are placed in time with respect to the true code group.

This criterion permits only a rough ordering of codes with respect to invulnerability; redundancy of orders and closed-loop missile response also play an obvious role. However, it does focus attention on a very important factor. No matter what random interference model is postulated — for example, a Poisson process of random pulses, a set of radars with approximately the same pulse repetition frequency operating inde-

pendently of each other, or peaks of random gaussian noise having average power a certain number of decibels below the pulse recognition threshold — false single pulses per unit time are much more frequent than false pairs having a particular spacing, false pairs are much more frequent than false triples, and so on. This situation is especially true when the discrete command tolerance τ is small with respect to the average time between false pulses.

II. ONE-STAGE CODES

This section discusses the problem of constructing one-stage codes (that is, codes using a delay line and AND circuit) for a set of k distinct commands. Each encoded command consists of a set of n pulses; the command information is contained in the $(n - 1)$ time spacings between pairs of successive pulses. One can encode these k commands such that i false pulses cannot combine in any way with $(n - i)$ pulses from any command to form either the same command (shifted in time) or one of the other $(k - 1)$ commands. The maximum possible value of i is equal to $(n - 2)$, because $(n - 1)$ false pulses can combine with one true pulse to form false commands in many different ways.

We first discuss methods of encoding commands invulnerable to $(n - 2)$ false pulses; later this restriction is relaxed to fewer pulses.

2.1 *The Construction of One-Stage Codes with Maximum Protection*

It may be helpful to think about a specific coding problem while reading this section. Let us assume that we wish to encode two commands $(k = 2)$ consisting of three pulses each $(n = 3)$; by suitable encoding we can insure that a single false pulse $(n - 2 = 1)$ cannot combine with two pulses to form any false command. This example will be referred to below by sentences enclosed in brackets.

As mentioned earlier, it is convenient to make all the time spacings of a discrete code integral multiples of the discrete command tolerance τ, in order to avoid overlapping problems between different commands. This being so, one can represent the various time spacings with the integers $1, 2, 3, \cdots$; the jth n-pulse command is represented by an $(n - 1)$-dimensional vector of integers $(t_1^j, t_2^j, \cdots, t_{n-1}^j)$. [Specifically, the first command is given by the pair of integers (t_1^1, t_2^1) and the second command by (t_1^2, t_2^2)].

The one-stage coding problem for discrete commands can now be stated in mathematical terms. Assume that one has k distinct n-pulse commands. All these commands are effective against $(n - 2)$ false pulses [in the sense discussed above] if and only if the following $kn(n - 1)/2$ integers are *all* different:

$$\begin{cases} t_i^{\,j} & \text{for } i = 1, 2, \cdots, n-1 \text{ and } j = 1, 2, \cdots, k; \\ t_i^{\,j} + t_{i+1}^{\,j} & \text{for } i = 1, 2, \cdots, n-2 \text{ and } j = 1, 2, \cdots, k; \\ \vdots & \\ \sum_{i=1}^{n-1} t_i^{\,j} & \text{for } j = 1, 2, \cdots, k. \end{cases}$$

[In our specific problem, we require that the six integers $t_1^{\,1}$, $t_2^{\,1}$, $t_1^{\,1} + t_2^{\,1}$, $t_1^{\,2}$, $t_2^{\,2}$, $t_1^{\,2} + t_2^{\,2}$ all be different.]

The theorem can be proved by indirect reasoning. Suppose that $(n - 2)$ false pulses have combined with two true pulses to form a false command. Then the spacing t_0 between the two true pulses must be equal to *more than one* of the $kn(n - 1)/2$ integers listed above (the set of all possible spacings between pairs of pulses in the k commands). For, if t_0 were equal to only one of these integers, the only command that could be formed with the aid of these two true pulses would be the correct command. Conversely, if two of these integers are the same, it is easy to construct a false command using $(n - 2)$ properly spaced false pulses. [For example, if the two commands are (3, 8) and (2, 6), then the second command can be falsely formed from the first command by placing an extra pulse two time units after the second pulse in the first command. In this case, the six integers are 3, 8, 11, 2, 6, 8, and t_0 is equal to 8.]

If one allows the integers $t_i^{\,j}$ to be arbitrarily large, there is no difficulty in discovering suitable codes. [For example, (31, 73) and (9, 45).] However, equipment and time restrictions usually make it desirable to keep all the commands as short as possible (measured from first to last pulse). Therefore, one may pose the following interesting mathematical problems:

i. For a given k and n, what is the minimum possible length of the longest command under the restrictions above?

ii. Are there any (simple and practical) methods for constructing codes with minimum command length (or a little longer)?

Neither of the above two problems has been solved; lower bounds can (in a limited way) be set for the first, and the second has largely been investigated by trial and error. The problem is similar to the eight queens problem* in chess, but with additional restrictions. More gener-

* The eight queens problem consists of placing eight queens on the chess board so that no queen can capture any other along a row, column, or diagonal. A specific code can be characterized by a pair of integers (a, b); this denotes a position on a generalized chessboard (not restricted to the eight-by-eight size).

ally, it appears to be a problem in partitions in the theory of numbers.

A general lower bound L for the minimum possible length of the longest command is given by $kn(n-1)/2$. However, this bound cannot always be achieved even for three-pulse codes. For a three-pulse code, it is true that

$$\sum_{i=1}^{2}\sum_{j=1}^{k} t_i^j = \sum_{j=1}^{k}(t_1^j + t_2^j).$$

If the longest command length is equal to $3k$, the set of integers

$$t_i^j (i = 1, 2; j = 1, 2, \cdots, k),\ t_1^j + t_2^j (j = 1, 2, \cdots, k)$$

is a rearrangement of the set of integers $(1, 2, \cdots, 3k)$. Therefore, the sum of the two sides of the above equation is $3k(3k+1)/2$, and the sum of one side alone is $3k(3k+1)/4$. But this last quantity is an integer if and only if $k \equiv 0 \pmod 4$ or $k \equiv 1 \pmod 4$. In other words, $3k$ is a lower bound L when $k = 1, 4, 5, 8, 9, 12, 13, \cdots$, and $3k+1$ is a lower bound L for all other k.

2.2 *Some Typical One-Stage Encodings*

We now list specific discrete encodings for several values of k and n. For the three-pulse codes the lower bound L for the longest delay line needed has actually been achieved for $k = 1\ (1)\ 10$. The first number gives the time spacing between the first and second pulses, and the second number the time spacing between the second and third pulses:

k	L	Typical minimum-length discrete-command three-pulse encodings									
2	7	(1,5)	(3,4)								
3	10	(4,6)	(1,7)	(2,3)							
4	12	(1,9)	(3,8)	(5,7)	(2,4)						
5	15	(3,12)	(5,9)	(2,11)	(4,6)	(1,7)					
6	19	(4,15)	(2,16)	(5,12)	(3,8)	(1,9)	(6,7)				
7	22	(3,19)	(5,16)	(7,13)	(6,12)	(2,15)	(4,10)	(1,8)			
8	24	(1,23)	(2,11)	(3,18)	(4,10)	(5,15)	(6,16)	(7,12)	(8,9)		
9	27	(1,26)	(3,22)	(9,15)	(2,21)	(6,14)	(7,12)	(10,8)	(4,13)	(5,11)	
10	31	(5,26)	(14,16)	(2,27)	(8,20)	(4,21)	(9,15)	(1,22)	(7,12)	(3,10)	(6,11)

Undoubtedly this list could be indefinitely extended, since it is easy to discover many different encodings for any k which have the same length for a maximum-length command.

Unfortunately, an analogous method for improving the lower bound L does not exist for $n > 3$. As n increases, it becomes more difficult to keep the minimum delay line small, and $kn(n-1)/2$ becomes a very unrealistic lower bound.

The tables below list representative discrete-command four-, five- and six-pulse encodings; the reader is invited to find an encoding with a smaller L' if he can (L' denotes the minimum command length that has been achieved, but it may not be the true minimum):

k	L	L'	Typical discrete-command four-pulse encodings				
2	12	13	(2,5,6)	(3,1,8)			
3	18	21	(1,5,12)	(2,8,11)	(3,4,9)		
4	24	27	(1,11,15)	(2,6,17)	(3,7,14)	(4,5,13)	
5	30	31	(2,20,9)	(3,14,13)	(4,8,16)	(5,10,11)	(6,1,18)

k	L	L'	Typical discrete-command five-pulse encodings		
2	20	25	(1,8,3,13)	(2,5,10,4)	
3	30	38	(1,5,12,20)	(2,8,11,14)	(3,4,9,15)

k	L	L'	Typical discrete-command six-pulse encodings	
2	30	45	(1,8,3,13,20)	(2,5,10,4,18)

If encodings not given in these tables are desired, it should be possible to use a high-speed digital computer to search through a large number of encodings (either systematically or at random) and print out the minimum-length encoding that it finds. Unless an algorithm for computing encodings is found, this is the only practical method available.

2.3 *A General Class of Discrete Codes*

As noted in Section 2.2, the length of the longest command increases rapidly with n, the number of pulses in the command. This, of course, is the price that one pays for protecting oneself against $(n - 2)$ false pulses arranged in any pattern whatever. One can always reduce the code length by reducing the protection; that is, construct minimum-length n-pulse codes that protect against $(n - 1)$ or fewer pulses ($i = 3, 4, 5, \cdots, n - 1$). The following formulation of the problem is more realistic. Suppose that one wishes protection against m false pulses combining with $(n - m)$ pulses in any command.

i. How does one construct n-pulse codes ($n > m + 2$), keeping the length L of the longest command as short as possible?

ii. If power is no limitation (any n can be used within reason), for what value of n is L minimized?

No general solution to either of these problems is known. This section presents typical encodings for a few of the simplest codes, which were found by trial-and-error methods.

The simplest problem in this class is the construction of a four-pulse

code invulnerable to a single false pulse. It can be proved (by the method used in Section 2.2) that, in order for a four-pulse, k-command code to be invulnerable to one false pulse the following $4k$ integer-pairs must all be different:

$$(t_1^j, t_2^j), \quad (t_2^j, t_3^j), \quad (t_1^j + t_2^j, t_3^j), \quad (t_1^j, t_2^j + t_3^j)$$

$$\text{for } j = 1, 2, \cdots, k.$$

As before, a lower bound L for the length of the longest command can be obtained. There is one integer-pair (1,1) which sums to 2; there are two more which sum to 3, and in general there are $n(n-1)/2$ with sums less than or equal to n. Therefore, a lower bound L is given by the smallest value of n satisfying $4k \leqq n(n-1)/2$. Most of the specific discrete encodings listed below actually achieve this bound; the ones that exceed it by one are marked with an asterisk:

k	L	Typical four-pulse encodings invulnerable to one false pulse						
1	4	(1,1,2)						
2	5	(1,3,1)	(2,1,2)					
3	6	(4,1,1)	(1,2,2)	(2,1,3)				
4	7	(1,4,2)	(2,4,1)	(1,1,2)	(3,2,1)			
5	7	(1,3,1)	(3,2,3)	(4,2,1)	(1,2,4)	(2,5,1)*		
6	8	(1,5,2)	(2,5,1)	(1,2,4)	(4,2,1)	(3,2,3)	(1,3,1)	
7	8	(1,5,3)	(2,2,5)	(3,3,2)	(4,2,1)	(7,1,1)	(1,3,4)	(2,3,1)*

For moderate values of k, the minimum length has been reduced more than 50 per cent by adding one more pulse to the code.

In order to construct a five-pulse k-command code that is invulnerable to two false pulses, it can be shown (by the method used in Section 2.2) that the following $10k$ integer-pairs must all be different:

$$(t_1^j, t_2^j), \quad (t_2^j, t_3^j), \quad (t_3^j, t_4^j),$$

$$(t_1^j + t_2^j, t_3^j), \quad (t_2^j + t_3^j, t_4^j), \quad (t_1^j, t_2^j + t_3^j), \quad (t_2^j, t_3^j + t_4^j),$$

$$(t_1^j, t_2^j + t_3^j + t_4^j), \quad (t_1^j + t_2^j, t_3^j + t_4^j), \quad (t_1^j + t_2^j + t_3^j, t_4^j)$$

$$\text{for } j = 1, 2, \cdots, k.$$

The lower bound L is given by the smallest value of n satisfying $10k \leqq n(n-1)/2$. Three typical encodings are given below; it it believed that they are the shortest ones possible:

k	L	Typical five-pulse encodings invulnerable to two false pulses		
1	5	(1,2,2,1)		
2	7	(2,3,1,2)	(1,3,4,1)	
3	9	(3,3,1,4)	(2,2,1,6)	(1,7,1,2)

In order to construct a five-pulse k-command code that is invulnerable to one false pulse, it can be shown that the following $5k$ integer-triples must all be different:

$$(t_1^j, t_2^j, t_3^j), \quad (t_2^j, t_3^j, t_4^j),$$
$$(t_1^j + t_2^j, t_3^j, t_4^j), \quad (t_1^j, t_2^j + t_3^j, t_4^j), \quad (t_1^j, t_2^j, t_3^j + t_4^j)$$
$$\text{for } j = 1, 2, \cdots, k.$$

The lower bound L is given by the smallest value of n satisfying $5k \leqq n(n-1)(n-2)/6$. A few typical encodings (believed to be minimum length) are given below:

k	L	Typical five-pulse encodings invulnerable to one false pulse			
1	5	(1,1,2,1)			
2	5	(1,1,3,1)	(2,1,1,2)		
3	6	(1,1,3,1)	(2,1,1,2)	(1,3,2,1)	
4	6	(1,1,3,1)	(2,1,1,2)	(1,3,2,1)	(2,2,1,2)

As mentioned before, these tables can be extended with the aid of a high-speed digital computer.

The following table summarizes the known behavior of one-stage discrete-command codes. It records the shortest known length of the longest command; most of these lengths are believed to be the smallest ones achievable. It is evident that one can often reduce this length quite markedly by adding pulses to the code, and that the savings are likely to be greater as the number of commands increases:

Codes Invulnerable to One False Pulse

	Number of pulses		
k	3	4	5
1	3	4	5
2	7	5	6
3	10	6	7
4	12	7	7
5	15	8	—
6	19	8	—

Codes Invulnerable to Two False Pulses

	Number of pulses		
k	4	5	6
1	7	6	—
2	13	9	—
3	21	11	—
4	27	—	—

III. TWO-STAGE CODES

The single-stage codes discussed in Section II should prove quite useful to the designer of missile guidance equipment. However, they suffer from the drawback of trying to do too many things at once. Sometimes it is more reasonable to break the decoding job up into several smaller jobs that are performed in sequence. Thus, the output pulse resulting

from one decoding operation becomes an input pulse to the next decoding operation.

An important advantage of such a break-up is that at least part of the security function of the code can be separated from the command function. This means that one can build in between-missile security without being forced to change the command codes from one missile to the next. Furthermore, the between-missile security codes can be used to increase the search time of a knowledgeable enemy for the correct code.

A second advantage is greater simplicity of both codes and components. If one needs to protect against (say) as many as four false pulses, it is not easy to find optimum single-stage codes, and the required delay lines may not be available. Multistage codes provide additional designs that may be easier to instrument. Balanced against this, of course, may be an increase in the number of components needed.

Multistage discrete-command codes are those that use a delay line and multiple AND circuit decoding mechanism (see Section I) at each stage. For simplicity, let us restrict ourselves to two-stage codes, which possess most of the potential advantages of multistage codes. Let the symbol $i \mid j$ denote a two-stage code containing ij pulses: the first stage decodes j clusters of i pulses each and the second stage decodes the cluster of j output pulses emitted by the first-stage AND circuit.

Usually the command part of the code occurs at the second stage. However, it makes no difference whatever which stage it is assigned to: an $i \mid j$ security-command code has precisely the same invulnerability to false pulses as does a $j \mid i$ command-security code. Let us assume for convenience that the commands are always contained in the *second* stage.

3.1 *An Upper Bound for the Protection*

How much protection can an $i \mid j$ code give against false pulses? Assume that the two stages are each designed according to single-stage rules, so that they are invulnerable to $(i - 2)$ and $(j - 2)$ false pulses, respectively. Then an upper bound to the number of false pulses that can be arranged in any pattern whatever with the ij true pulses *without* forming a false (or repeated) command is

$$M = \min [i(j - 1) - 1, j(i - 1) - 1].$$

The first term arises because false pulses can be arranged in $(j - 1)$ groups of i false pulses each, and these groups can be combined with one true i-pulse group to form a false command. The second term arises

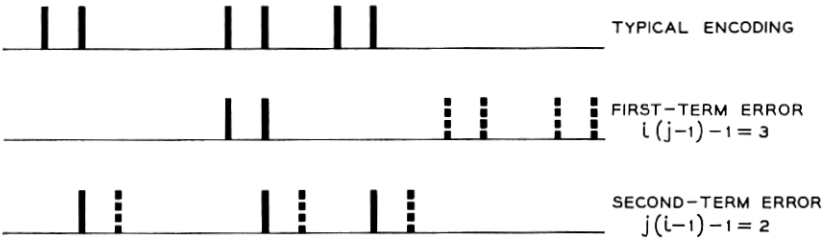

Fig. 2 — First- and second-term error for 2 | 3 code.

because false pulses can be arranged in j groups of $(i - 1)$ false pulses each, augmented by j true pulses, one from each of the j groups in the correct code. This is illustrated in Fig. 2 for the 2 | 3 code.

Although M is the same for both an $i \mid j$ and a $j \mid i$ code, the two errors are not symmetric. Note that the first type of error can result in a false command, but the second type of error can only repeat the *same* command slightly earlier or later in time. The second error is likely to be less serious, and in fact can be eliminated by introducing a device that prevents repetition of the same command within a specified period of time. Accordingly, in the rest of this section we emphasize codes that are protected against $i(j - 1) - 1$ false pulses combining with true pulses to form a *different* false command.

For i, j greater than or equal to two, the maximum possible protection is always less than that achievable with a single-stage code using ij pulses. Thus, these codes are somewhat comparable to the reduced-protection single-stage codes discussed in Section II.

3.2 *Methods for Combining Security and Command Codes*

In order to actually achieve the upper bound of the protection, one must be a little careful when combining the two stages of the code. There are many ways of doing this, and the choice of a particular method depends upon the ease of instrumentation. For example, consider the following alternatives for the 2 | 3 code (see Fig. 3).

3.2.1 *Short Command Codes, Long Security Codes*

Let the three-pulse commands be encoded according to the methods in Section II. The longest command will be approximately $3k$ if there are k different commands. The two pulses forming the security code should be more than $6k$ apart in order to avoid the possibility of three false pulses forming a false command. This design is probably the easiest to instrument.

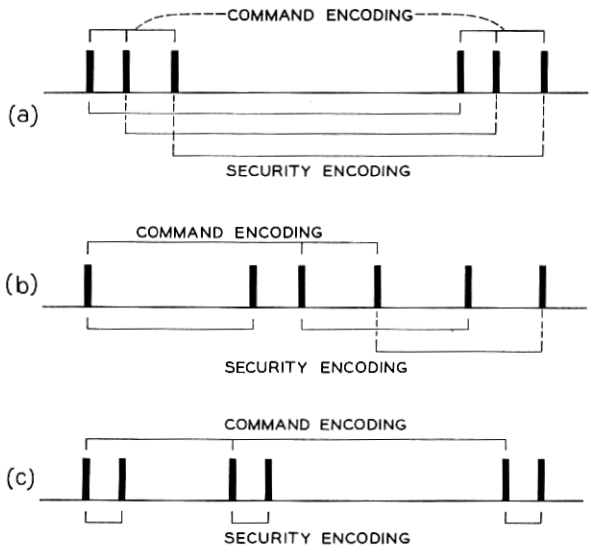

Fig. 3 — Methods for combining security and command codes: (a) short command codes, long security codes: (b) interleaved security and command codes; (c) long command codes, short security codes.

3.2.2 *Interleaved Security and Command Codes*

Let the spacing between the two-pulse security codes consist of *odd* integers, and encode the three-pulse commands in *even* integers using the methods discussed in Section II. To illustrate: if four commands are to be sent, let them be (2, 18), (6, 16), (10, 14) and (4, 8). More security and command codes can be easily added.

3.2.3 *Long Command Codes, Short Security Codes*

Let the spacing between the two-pulse security codes be given by the integers 1, 2, $\cdots$, m. Encode the three-pulse commands in multiples of $2m$ (or greater) according to the methods of Section II. This is likely to be the most difficult design to instrument.

These alternatives all provide the maximum possible protection; four false pulses are needed to form a false command, and three false pulses are needed to shift the true command in time.

The first and third alternatives are very easy to encode for any values of i and j (if the necessary single-stage codes are available), and are not discussed further. False pulses that combine with true pulses to form false

security codes cannot possibly form false commands as well, because these false pulses are either too far from or too close to the other true pulses to do the job. Frequently the longest delay line is somewhat longer for these alternatives than it would be if an interleaved code were used. Accordingly, we devote the rest of this part of the paper to the construction of interleaved codes.

3.3 *The Problem of Interleaving Security and Command Codes*

This section discusses the problem of interleaving i-pulse security codes with j-pulse command codes in such a way that one is protected against $i(j-1)-1$ false pulses forming a false command. Probably the simplest way to interleave the codes is to reserve the integers $f(i)$, $2f(i)$, $3f(i)$, $\cdots$ for the command codes. Then one can construct minimum-length command codes invulnerable to $(j-2)$ false input pulses according to the methods discussed in Section II. [It will be shown later that the minimum possible value for $f(i)$ is i.]

The interleaving problem is now reduced to the problem of selecting the security code spacings so that one is protected against $i(j-1)-1$ false pulses. The following sections present two restrictions on the security codes and show that these restrictions are sufficient to guarantee this protection. No claim is made that this is the only way to interleave codes; however, the resulting codes do have optimum properties, which are described later.

3.3.1 *Two Restrictions on the Security Code*

The security code spacings $(t_1, t_2, \cdots, t_{i-1})$ should be chosen in such a way that

(a) no set of $(i-2)$ false pulses can combine with two true pulses to form a false security code, and

(b) none of the $i(i-1)$ integers

$$\begin{cases} \pm t_k, & k = 1, 2, \cdots, i-1; \\ \pm(t_k + t_{k+1}), & k = 1, 2, \cdots, i-2; \\ \vdots \\ \pm(t_1 + t_2 + \cdots + t_{i-1}) \end{cases}$$

should be a multiple of $f(i)$ [that is, equal to $0 \bmod f(i)$].

The first restriction states that security codes should be single-stage codes consisting of only one command (see Section II). The second restriction accomplishes two goals. First, since none of the integers t_1,

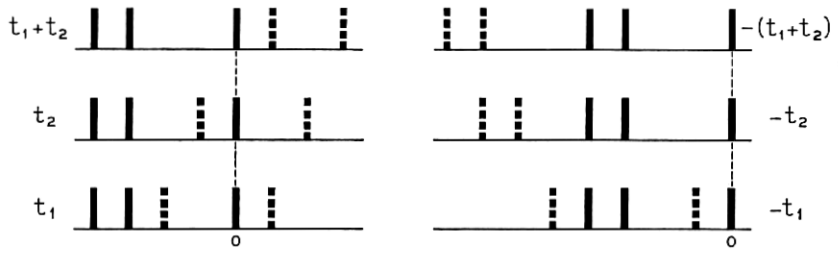

Fig. 4 — Six false security codes for a three-pulse code.

$t_1 + t_2 , \cdots , t_1 + t_2 + \cdots + t_{i-1}$ is a multiple of $f(i)$, any $i \mid j$ code will consist of the full set of ij pulses. (Any pulse serving two functions at the same time is a natural candidate for a false pulse.) Second, the above set of integers lists the starting times of all the false security codes consisting of $(i - 1)$ false pulses and one true pulse (relative to the starting time of the true security code). For example, for a three-pulse security code, the six false security codes are shown in Fig. 4.

Now, consider what happens if one of the above spacings is a multiple of $f(i)$. One of the command code spacings is formed by the false security code corresponding to this spacing and one of the true security codes; the other $(j - 2)$ command spacings are formed by $i(j - 2)$ false pulses. But $(i - 1) + i(j - 2) = i(j - 1) - 1$, which is the number of false pulses we wish to protect against. The $3 \mid 4$ code illustrates this point (see Fig. 5).

It is shown in Section 3.3.2 that codes satisfying these restrictions are protected against $i(j - 1) - 1$ false pulses, and that the minimum possible value of $f(i)$ is i.

3.3.2 *Proof That These Restrictions Insure Protection*

Given these two restrictions, how can false commands be formed? Each security code in the false command can include at most one true

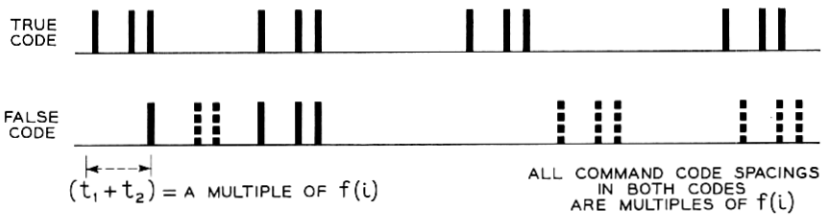

Fig. 5 — A $3 \mid 4$ code with spacing a multiple of $f(i)$.

pulse, because of restrictions (a) and (b). [There is one exception: a false command can be constructed out of one true security code and $(j - 1)$ false security codes of i pulses each.] Furthermore, ij pulses are needed for the false command, according to restriction (b). Therefore, if we can show that the false command contains at most i true pulses (one in each of i different false security codes), then we are done, for the $(ij - i)$ or more false pulses needed to make up the balance of the false command will exceed $i(j - 1) - 1$, the number of false pulses we wish to protect against. Since all the true command spacings are multiples of $f(i)$, it is sufficient to show that one can select at most i false security codes consisting of one true pulse and $(i - 1)$ false pulses each, which are all spaced $f(i)$ apart from each other.

In the preceding section, we listed all the $i(i - 1)$ possible false security code starting times (relative to the true security code). If we can show that exactly i of these times are equal to $a \pmod{i}$, where $a = 1, 2, \cdots, i - 1$, then we will have proved that at most i different false security codes can have the proper command spacing. Furthermore, we will have proved that $f(i) = i$; and it is clear that this is the smallest integer it can be.

Consider the following arrangement of the $i(i - 1)$ false security code starting times in a matrix with i rows and $(i - 1)$ columns:

$$
\begin{array}{l}
t_1 , t_1 + t_2 , t_1 + t_2 + t_3 , \cdots \quad \cdots , t_1 + \cdots + t_{i-1} \\
t_2 , t_2 + t_3 , \cdots \quad \cdots , t_2 + \cdots + t_{i-1} , - t_1 \\
t_3 , t_3 + t_4 , \cdots \quad \cdots , t_3 + \cdots + t_{i-1} , -t_2 , -(t_1 + t_2) \\
\quad \vdots \\
t_{i-1} , -t_{i-2} , -(t_{i-3} + t_{i-2}), \cdots \quad \cdots , -(t_1 + \cdots + t_{i-2}) \\
-t_{i-1} , -(t_{i-2} + t_{i-1}), \cdots \quad \cdots , -(t_2 + \cdots + t_{i-1}), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad -(t_1 + \cdots + t_{i-1}).
\end{array}
$$

If the integers in each row are reduced modulo i, they will be a permutation of the integers $1, 2, \cdots, i - 1$. We prove this by showing that the opposite conclusion leads to a contradiction. If two integers in a row are both equal to the same integer mod i, then their difference is equal to 0 mod i. But their difference is contained elsewhere in the matrix, which contradicts the assumption that no starting times are 0 mod i [restriction (b) above]. Since we have proved that each row is a permutation of the first $(i - 1)$ integers, it is clear that the matrix consists of i starting times equal to $a \pmod{i}$, $a = 1, 2, \cdots, (i - 1)$.

It is not too difficult to actually find security codes satisfying restriction (b). For example, consider the security code ($t_1 = 1, t_2 = 2, t_3 = 3$). We have

$$\begin{aligned} \pm t_1 = \pm 1 &= 1 \bmod 4, 3 \bmod 4; \\ \pm t_2 = \pm 2 &= 2 \bmod 4, 2 \bmod 4; \\ \pm t_3 = \pm 3 &= 3 \bmod 4, 1 \bmod 4; \\ \pm (t_1 + t_2) = \pm 3 &= 3 \bmod 4, 1 \bmod 4; \\ \pm (t_2 + t_3) = \pm 5 &= 1 \bmod 4, 3 \bmod 4; \\ \pm (t_1 + t_2 + t_3) = \pm 6 &= 2 \bmod 4, 2 \bmod 4. \end{aligned}$$

There are exactly four starting times for each of the three values of a. Note that this security code does not satisfy restriction (a), but that ($t_1 = 1, t_2 = 2 + 4 = 6, t_3 = 3$) does.

If $i < j$ in an $i \mid j$ two-stage code, the above two restrictions on the construction of security codes should be applied. However, if $i > j$, then we automatically have more than the minimum protection $i(j - 1) - 1$ against false pulses — the false command can contain at most $ij - j$ false pulses (ignoring the one exception mentioned above). One can trade off this unnecessary additional protection for a larger set of security codes. To be specific, one can replace the first restriction with one of the following less stringent restrictions:

(a′) no set of $(i - k)$ false pulses, for $k = 3, 4, \cdots, i - 1$, can combine with k true pulses to form a false security code.

These single-stage codes have already been discussed in Section II. For example, if one wishes to construct a $4 \mid 2$ code, then one can use security codes that protect against only one false pulse (instead of two, the greatest possible protection).

3.4 *Some Typical Interleaved Security and Command Codes*

The preceding section presented two restrictions on the security code, which guarantee that the resulting $i \mid j$ interleaved two-stage code will be protected against $i(j - 1) - 1$ false pulses forming a false command. Nothing, however, was said about the actual construction of such codes. This section presents sample codes for low values of i and j.

The command codes of an $i \mid j$ code are restricted to the spacings i, $2i$, $3i$, $\cdots$ and are constructed according to the methods discussed in Section II. The security codes are constructed by trial-and-error methods to satisfy the two restrictions. For $i = 2$, the odd integers all form se-

curity encodings. For $i = 3$, the pairs (1, 4), (2, 5), (4, 7), (5, 8), $\cdots$ and their reverses are legitimate security encodings. In general, such encodings must be of the form (t_1, t_2), where $t_1 \neq t_2$ and (t_1, t_2) equals (1 mod 3, 1 mod 3) or (2 mod 3, 2 mod 3). For $i = 4$, the triples (1, 5, 9), (2, 1, 6), (1, 6, 3), (2, 3, 6) and (6, 5, 2) are examples of short security encodings. In general, these security encodings must be of the form (t_1, t_2, t_3) where none of the integers $t_1, t_2, t_3, t_1 + t_2, t_2 + t_3$ are equal, and (t_1, t_2, t_3) is in one of the following forms: (1 mod 4, 1 mod 4, 1 mod 4); (2 mod 4, 1 mod 4, 2 mod 4); (1 mod 4, 2 mod 4, 3 mod 4); (2 mod 4, 3 mod 4, 2 mod 4); (3 mod 4, 3 mod 4, 3 mod 4) or (3 mod 4, 2 mod 4, 1 mod 4).

These interleaved two-stage codes can be regarded as minimum-length codes in the following sense: given that the command codes are restricted to the integers $f(i), 2f(i), 3f(i), \cdots$, then

(a) $f(i)$ is equal to its minimum value i;

(b) the longest command code is as short as possible; and

(c) the security codes are as short as possible.

Assuming that one wishes to protect against a given number of false pulses, which $i \mid j$ code should one select? The answer depends upon the relative importance of minimizing the number of pulses ij in the code, and keeping the longest command delay line as short as possible. There are two possible ways to balance these factors against each other:

(a) apply the methods used in one-stage codes (Section II) to the command code;

(b) change the values of both i and j; codes with large i and small j have shorter delay lines and more pulses, while codes with small i and large j have the opposite characteristics.

Let us illustrate these ideas with a few simple examples. Assume that we wish to transmit six commands. The $2 \mid 3$ and $4 \mid 2$ codes both provide protection against at most three false pulses. The number of pulses per command is six and eight, while the maximum delay-line length is $19(2) = 38$ and $6(4) = 24$, respectively. Suppose that one uses a $2 \mid 4$ code but designs the command code so that it is invulnerable to one false input pulse (no matter how placed) but not two. Then, referring to Section II, we see that the number of pulses per command is eight and the maximum delay line is $8(2) = 16$.

How do two-stage codes compare with one-stage codes? The only direct comparison available is the $2 \mid 2$ code with the four-pulse one-stage code protected against one false pulse: both codes have the same number of pulses and the same protection against false pulses. The length of the longest delay line needed is tabled below for various commands:

Number of commands	2	3	4	5	6	7
Two-stage code	4	6	8	10	12	14
One-stage code	5	6	7	8	8	9

IV. ACKNOWLEDGMENT

The author is indebted to W. J. Albersheim for the suggestion that a general study of missile codes be made. He is also indebted to W. L. Roach for a very careful reading of an earlier draft; his suggestions led to many improvements in this paper.

APPENDIX

Relationship to Error-Correcting Binary Codes

There is a certain relationship between the missile guidance one-stage code problem presented in Section II of this paper and the error-correcting binary codes that have been extensively studied in information theory. Suppose that one wishes to transmit any one of a large set of messages over a binary channel, and suppose that there is a probability p that a one will be changed to a zero or a zero to a one in the course of transmission. One can encode the messages in such a way that every message differs from every other message in at least four places: for example, 100000, 011100, 111011 and 000111. (If four messages are to be sent, these are the shortest messages possible.) Note that single errors can be corrected immediately (111111 must be 111011), and double errors can be detected but not corrected (110100 can be either 100000 or 011100).

In general, one attempts to encode N messages in lengths as short as possible so that n simultaneous errors in the encoding can be corrected (that is, the original message can be identified). Viewed geometrically, each encoded message has a cluster of closely related correctable messages associated with it (for example, all messages differing from the correct one in only one unit). These clusters are packed as tightly as possible into a binary k-dimensional space (k is the encoded message length) having a total of 2^k points. When the encoded messages are transmitted, synchronization of some sort must be provided between the transmitter and the receiver in order that the receiver may know when each binary message starts. The message length is kept as short as possible in order to maximize the information rate in the channel.*

The missile guidance one-stage codes of this paper can very easily

* For a detailed discussion of these error-correcting binary codes, see Slepian.[1]

be represented as binary messages: each binary digit corresponds to τ time units, and the pulses of the message are represented by ones. For example, the two three-pulse commands (1, 6) and (2, 3) become (11000001) and (101001). To make the representation more precise, the shorter messages can be extended with zeros so that all messages are of equal length.

There are two differences between the missile guidance single-stage codes and the error-correcting binary codes; both are discussed in the paragraphs below. The first difference is concerned with synchronization, and the second with the shapes of the clusters of correctable messages.

A.1 *Synchronization*

The error-correcting binary decoder accepts messages in non-overlapping sets of k digits each (where k is the message length). This implies that some sort of synchronization between the sender and receiver has been established; otherwise the receiver does not know when to start decoding. This synchronization is not easy to provide in missile communications, because of the rapidly changing position of the missile relative to the ground transmitter. Therefore, the missile receiver is arranged to start decoding any time a pulse (that is, a "one") is received.

Some work has been done on self-synchronized codes. For example, Golomb, Gordon and Welch[2] have derived upper bounds for the number of k-digit messages that can be constructed using an n-digit alphabet. They require that the set of messages have the property that the final $(k - i)$ digits of any message in the set followed by the first i digits of any message in the set does not form a message in the set. A set of messages with this property automatically provides synchronization; even when the decoder looks at all the intermediate "messages", it recognizes none of them. However, the work described above has not yet been applied to error-correcting codes. It is possible, in fact, that self-synchronized error-correcting binary codes may be of limited interest to information theorists because of the reduction in the number of messages per second that can be sent over a channel. However, channel capacity and rate of information flow are not objectives of the missile guidance codes.

A.2 *Clusters of Correctable Errors*

The second difference between the missile codes and the binary codes is the shape of the cluster of correctable errors associated with a given message. The binary code cluster usually consists of those k-digit mes-

sages that differ in only a few digits (say, one or two) from the true message. The missile code cluster, on the other hand, is neither "close" to the true message nor easy to describe geometrically. It consists of all k-digit messages that contain ones in n specified places: for example, if 100101 is the message, then 111101, 101101 and 111111 will be decoded as the message. Obviously, the correctable clusters of different messages overlap each other. To separate the clusters as much as possible, the messages are selected so that no member of one cluster with $(2n - 2)$ or fewer ones is a member of any other cluster. [This is simply a restatement of the requirement that $(n - 2)$ false pulses added to any command cannot possibly form a false command.]

In order to visualize these correctable clusters, it is convenient to transform from the k-dimensional discrete binary space of messages to the $(n - 1)$-dimensional continuous space of times between any set of n pulses (true or otherwise). Let us examine the shape of this cluster a little more precisely for n equal to low values. For $n = 2$, the cluster is a line segment 2τ units long* centered on the correct spacing t_1. For $n = 3$, the set of three-pulse messages that will be decoded as a command (t_1, t_2) is contained in the polygon of Fig. 6. This figure corresponds to a delay line that delays the first pulse for a time $t_1 + t_2$, the second pulse for a time t_2, and the third pulse not at all. A set of pulses with original spacing (t_1, t_2) will be brought into exact coincidence by this delay line. The upper right corner of the square has been sawed off because, for original spacings (t_1', t_2') in this region, $t_1' + t_2' > t_1 + t_2 + \tau$, and therefore the first and third pulses will be separated by more than τ after passing through the delay line.

For $n = 4$, all decodable messages are contained in the three-dimen-

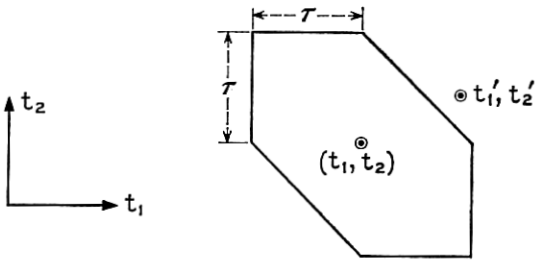

Fig. 6 — Polygon containing set of three-pulse messages that will be decoded as a command (t_1, t_2).

* The quantity τ is the discrete command tolerance defined in Section I. In order for a command to be recognized, the time between the earliest and latest of the n pulses arriving at the AND gate must be less than τ.

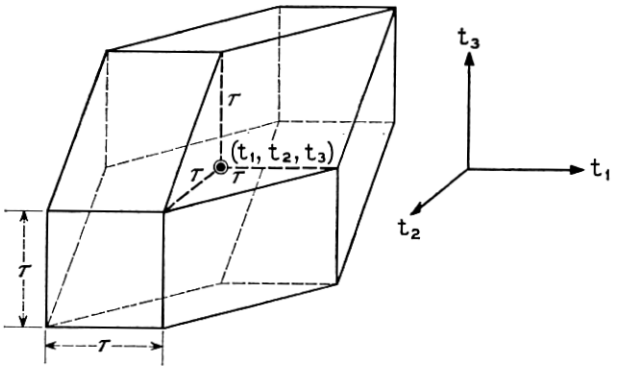

Fig. 7 — Polytope containing all decodable messages for $n = 4$.

sional polytope of Fig. 7. This figure corresponds to a delay line that delays the first pulse $t_1 + t_2 + t_3$, the second pulse $t_2 + t_3$, the third pulse t_3, and the fourth pulse not at all.

For $n \leqq 4$, it can be shown that these polytopes can be packed in such a way as to fill the space completely; it is probably not difficult to show that this is true for arbitrary n. The ratio of the volume of the $(n - 1)$-dimensional polytope to the $(n - 1)$-dimensional hypercube of side 2τ (which encloses it) is

$$\int_0^{\frac{1}{2}} n(n-1)x^{n-2}(1-x)\,dx = \frac{n+1}{2^n}.$$

This integral is equal to the probability that n points drawn at random from a uniform distribution on $(0, 1)$ will all be located within one-half of each other.

To what use can these polytopes be put? In the binary code problem, one places a message at the center of each polytope and packs them into the space as tightly as possible. However, in the missile guidance problem the messages cannot be packed so closely. Consider an n-pulse command plus $(n - 2)$ false pulses. There are $(2n - 2)!/n!(n - 2)!$ possible ways of choosing an n-pulse group, and none of these groups (except for the one consisting of n true pulses) is allowed to lie within any polytope centered on a true command. The polytopes must be very sparsely scattered through $(n - 1)$-space.

Let us assume that the true commands are transmitted with timing errors that are very small with respect to the discrete command tolerance τ. (This is ordinarily true in practice; if it were not, some of the commands might not be received by the missile.) Then, if the true com-

mands are encoded according to the rules stated in Section II, any false command consisting of $(n - 2)$ or fewer false pulses will have at least one spacing that is at least τ different from any true command spacing. But the polytope centered on any true command is bounded by the hypercube of side 2τ; therefore the false command cannot be decoded as any true command. In general, if the maximum timing error is ϵ, the commands should be encoded in integral multiples of a basic time-quantum $\tau + \epsilon$.

It is not possible to relax the encoding rules of Section II to allow for the fact that we are dealing with polytopes rather than hypercubes. The reason for this is simple: the polytope intersects each of the $2^{2(n-1)}$ faces of the hypercube.

REFERENCES

1. Slepian, D., A Class of Binary Signaling Alphabets, B.S.T.J., **35,** 1956, p. 203.
2. Golomb, S. W., Gordon, B. and Welch, L. P., Comma-Free Codes, Can. J. Math., **10,** 1958, p. 202.