GENERALIZATION OF SIGNAL POINT TARGET CODE

by

Md Munibun Billah

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

_____   _____
Todd Moon, Ph.D.   Jacob Gunther, Ph.D.
Major Professor   Committee Member

_____   _____
Janak Sodha, Ph.D.   Richard S. Inouye, Ph.D.
Committee Member   Vice Provost for Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2019

ABSTRACT

Generalization of Signal Point Target Code

by

Md Munibun Billah, MASTER OF SCIENCE

Utah State University, 2019

Major Professor: Todd Moon, Ph.D.
Department: Electrical and Computer Engineering

Signal Point Target Codes have been proposed as an alternative to Trellis Coded Modulation. In this thesis, the basic definition of Signal Point Target Code including larger signal constellations, larger state size, different rate, and different shapes are extended in several ways. A notation is introduced to describe the operations, constellations of different sizes and codes of different rates are presented. Performance is evaluated by simulation and by distance bounds on the decoding trellis. Simulations are performed using different Programming languages to determine the suitable language for studying error correction coding.

(160 pages)

PUBLIC ABSTRACT

Generalization of Signal Point Target Code

Md Munibun Billah

Detecting and correcting errors occurring in the transmitted data through a channel is a task of great importance in digital communication. In Error Correction Coding (ECC), some redundant data is added with the original data while transmitting. By exploiting the properties of the redundant data, the errors occurring in the data from the transmission can be detected and corrected. In this thesis, a new coding algorithm named Signal Point Target Code has been studied and various properties of the proposed code have been extended.

Signal Point Target Code (SPTC) uses a predefined shape within a given signal constellation to generate a parity symbol. In this thesis, the relation between the employed shape and the performance of the proposed code have been studied and an extension of the SPTC are presented.

This research presents simulation results to compare the performances of the proposed codes. The results have been simulated using different programming languages, and a comparison between those programming languages is provided. The performance of the codes are analyzed and possible future research areas have been indicated.

To my parents who shaped my knowledge about life and to Dr. Moon who shaped my knowledge about mathematics.

ACKNOWLEDGMENTS

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

ECC	Error Correction Coding
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
SER	Symbol Error Rate
SNR	Signal to Noise Ratio
LDPC	Low Density Parity Check
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
PSK	Phase Shift Keying
SPTC	Signal Point Target Code
SC	Shape Code
CAC	Constellation Arithmetic Code
SIE	Sequential Input Encoding
PPE	Partial Parallel Encoding
TCM	Trellis Coded Modulation
IDE	Integrated Development Environment

CHAPTER 1

INTRODUCTION

1.1 Overview

Error correction coding is the means of detecting and correcting errors induced by communication channels on received data by utilizing the properties of redundant data added with the message data for transmission. Since Shannon's channel coding theorem [1] in 1948, studies to find good codes which achieve near channel capacity have been conducted. Finding a code of lower complexity that achieves Shannon channel capacity is of primary interest in the field of error correction coding. Those studies led to the discovery of some good codes, such as Low Density Parity Check (LDPC) [2], Turbo Codes [3] and Polar Codes [4]. The redundant bits of error correcting codes increase the ability to detect and correct errors but reduce the rate of transmission. Ungerboeck proposed a coding scheme Trellis Coded Modulation (TCM) [5] which incorporates error correction coding and modulation as a single unit to improve the information transmission rate. In [5], Ungerboek used a convolutional encoder followed by mapping into the signal constellation to encode and a sequential decoder to decode TCM. A new coding algorithm named Shape Code (SC) was proposed in [6] and extended in [7], which takes into account a shape or predefined path in the signal constellation. That shape was used to encode input symbols to generate parity symbols. Like TCM, SC employs error correction coding within the signal space. The work of this thesis is to study, extend and evaluate multiple attributes of the encoding algorithm [7] in simulation.

1.2 Literature Review

The concept of shapes formed within the decoder and how these shapes affect the decoding capability was introduced in [8]. Using the concept of shape, Sodha devised

a rate 1/2 systematic code (Shape Code) on QPSK constellation [6]. A square shape within the signal space was formed by selecting appropriate points from the constellations. This collection of points was used to encode input symbols by calculating the numbers of clockwise 90 degree rotations to go from the input symbol to a point from that collection. This collection of points was later named as target points in [7] where the idea of Shape Code was extended for a larger constellation (16QAM). In [7], alongside with the extension of Shape Code for a larger constellation, a state variable was introduced which enabled the encoder to form a trellis by utilizing input symbols and target points. The trellis structure allowed for trellis based decoding. From [7], it can be seen that for shape code, a standard Viterbi decoder increases the coding gain compared to the performance in [6]. It will be also shown that the coding gain depends on the free Euclidean distance on the trellis and by increasing the distance is possible to achieve more coding gain.

1.3 Thesis Objectives

The primary goal of this thesis is to study and generalize Signal Point Target Code (SPTC). The performance of the codes will be measured in Bit Error Rate (BER) and Symbol Error Rate (SER) and will be compared with the results of uncoded modulations. The objectives of this thesis are:

1. Generalize the encoder which is already proposed for QPSK and 16QAM, to other QAM signal constellation.

2. Study the performance of the proposed code as shapes are modified.

3. Study encoding under different coding rates.

4. Study the performance of the encoder for the different number of states.

5. Study the effect on the performance of the code due to the free Euclidean distance in trellis to gain some understanding of the theoretical coding gain.

A secondary objective of this thesis is to compare the simulation speed using between MATLAB, Python, and Julia to determine which tool is more suitable for error correction studies.

1.4 Chapter Outlines

The chapter organization for this thesis is as follows:

Chapter 2 presents the background of SPTC. At the beginning of chapter 2, the encoding operation of SPTC is by introducing appropriate notations. Using trellis structure, the free Euclidean distance related to the shape is calculated to show that the performance of SPTC depends on the employed shape. How the codes can be decoded using the standard Viterbi algorithm is also described in this chapter. Following the motivation to increase free Euclidean distance for codes by searching for the best shape, an alternative of SPTC called Constellation Arithmetic Code (CAC) which depends on the bit assignment rather than the shape is introduced. A solution for bit assignment for CAC in the 16QAM and 64QAM signal constellation is presented that provides better coding gain compared to the performances of the shapes for encoding SPTC explored in this thesis. At the end of this chapter, two methods for increasing the rate of CAC are proposed.

Chapter 3 starts with the required calculations for simulating the performance of the proposed codes using an AWGN channel. How the noise is generated using the variance obtained for a given constellation is described here. The equations of the probability of symbol error for both theory and simulation are provided. Later in this chapter, simulation results for SPTC and CAC are presented, comparison between the performances of SPTC and CAC are made and the performances of rate 2/3 and rate 3/4 codes for CAC are presented. Finally, this chapter ends with the comparison between the runtime of MATLAB, Python, and Julia to determine a suitable language for error correction coding study.

In chapter 4, the idea of how increasing the number of states can increase the free Euclidean distance and therefore increase the coding gain has been explored. An approach for increasing the number of states and the free Euclidean distances is proposed. For rate 1/2 CAC using QPSK and 16QAM constellation, a list for the different number of states

and the associated free Euclidean distances is provided. However, the proposed approach failed to provide any coding gain for the increment of the number of states in simulations. So this chapter is presented in this thesis in order to provide an insight for future researches.

Chapter 5 concludes the thesis by discussing the findings for SPTC and CAC and by pointing out areas for possible future research. The appendix contains all of the codes written for the simulations in this thesis.

CHAPTER 2

SIGNAL POINT TARGET CODE

The focus of this chapter is the Signal Point Target Code (SPTC). In this chapter, the encoding operations of SPTC are explained first. Then, a modified version of SPTC is presented and using that modified version two methods for changing the rate of codes are proposed.

Let M denote the number of points in a digital constellation and $n = \log_2 M$ be the number of bits per symbol. Let $\{b_0, b_1, b_2, ...\}$ denote a sequence of randomly generated bits where $b \in \{0, 1\}$. At the ith symbol interval, n bits are stacked into a binary vector $\mathbf{b}_i = [b_{ni}\ b_{ni+1}..... \ b_{ni+n-1}]^T$.

Let $\mathbb{S}$ be the set of points in the signal constellation and $\phi : \{0, 1\}^n \to \mathbb{S}$ be a bijective mapping from n dimensional binary vector $\mathbf{b}$ to a point in the signal space $S \in \mathbb{S}$,

$$S = \phi(\mathbf{b}). \tag{2.1}$$

ϕ is the bit assignment operation in the signal constellation. In this thesis, for SPTC gray code indexing has been used in all the signal constellations. However, for a modified version of SPTC called Constellation Arithmetic Code, bit assignment operations beside gray code indexing are used to evaluate coding gain.

2.1 Encode

Let $\psi : \mathbb{S} \to \mathbb{Z}_{\geq 0}$ be a bijective mapping between points in the signal constellation S and an integer m where ψ assigns each point in the signal constellation to a unique integer index $m \in \mathbb{Z}_{\geq 0}$. In this thesis, the integer index m is equal to the decimal representation of the binary vector that has been assigned to that point by ϕ. The set of such integers will be denoted as the set of signal space index $\mathcal{I}$. For example, in the signal constellation shown in figure 2.1, ϕ maps binary vectors [0 0 1 0] and [1 1 0 1] to points $(-3, 3)$ and $(1, -1)$

while ψ maps the point $(-3,3) \in \mathbb{S}$ to the integer index 2 and the point $(1,-1) \in \mathbb{S}$ to the integer index 13.

Under this encoding scheme, let an "addition" operation $\oplus : \mathcal{I} \times \mathbb{Z}_{\geq 0} \to \mathcal{I}$ in the signal constellation, such that $s_{\text{new}} = s_i \oplus j$ moves from the point s_i in the signal space to a new point s_{new} by the amount j following a predefined path, where $s_{\text{new}}, s_i \in \mathcal{I}$ and $j \in \mathbb{Z}_{\geq 0}$. For example, for the path defined by blue arrows in the signal constellation shown in Fig 2.1, $7 \oplus 6 = 8$ because from point 7 moving 6 steps along the path takes to point 8. In this thesis, the predefined path which is followed by the $\oplus$ operator is defined as *shape*. Any shape can be used for encoding as long as the shape goes through all of the points in the signal constellation and the shape goes through each point only once. The total number points which are used to form the shape is M. The addition defined by $\oplus$ associated with the shape shown in Fig. 2.1 is listed in Table 2.1.
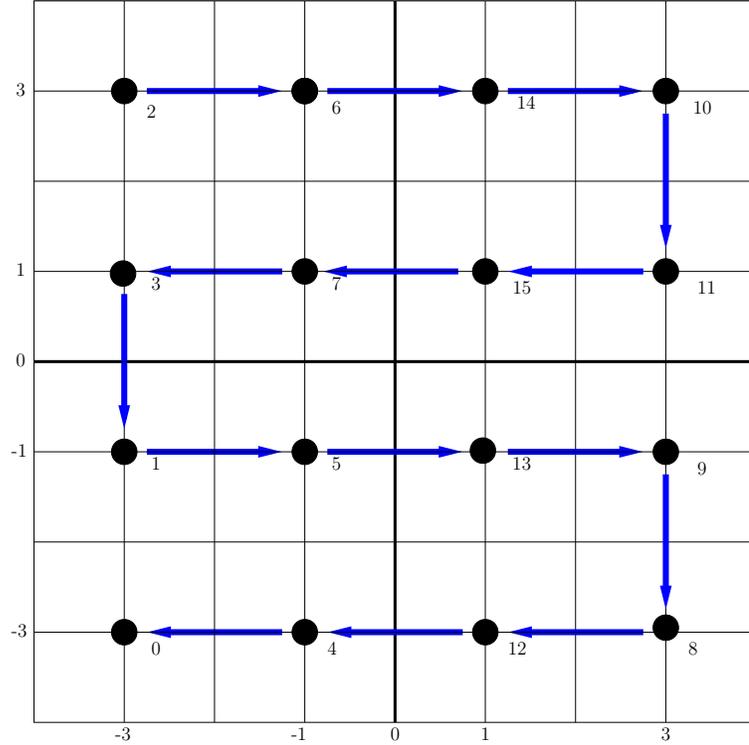
Fig. 2.1: 16-QAM signal constellation with a shape indicated by blue arrows.

⊕	j															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	2	6	14	10	11	15	7	3	1	5	13	9	8	12	4
1	1	5	13	9	8	12	4	0	2	6	14	10	11	15	7	3
2	2	6	14	10	11	15	7	3	1	5	13	9	8	12	4	0
3	3	1	5	13	9	8	12	4	0	2	6	14	10	1	15	7
4	4	0	2	6	14	10	11	15	7	3	1	5	13	9	8	12
5	5	13	9	8	12	4	0	2	6	14	10	11	15	7	3	1
6	6	14	10	11	15	7	3	1	5	13	9	8	12	4	0	2
7	7	3	1	5	13	9	8	12	4	0	2	6	14	10	11	15
8	8	12	4	0	2	6	14	10	11	15	7	3	1	5	13	9
9	9	8	12	4	0	2	6	14	10	11	15	7	3	1	5	13
10	10	11	15	7	3	1	5	13	9	8	12	4	0	2	6	14
11	11	15	7	3	1	5	13	9	8	12	4	0	2	6	14	10
12	12	4	0	2	6	14	10	11	15	7	3	1	5	13	9	8
13	13	9	8	12	4	0	2	6	14	10	11	15	7	3	1	5
14	14	10	11	15	7	3	1	5	13	9	8	12	4	0	2	6
15	15	7	3	1	5	13	9	8	12	4	0	2	6	14	10	11

(s labels the rows.)

Table 2.1: Input output relation table for $\oplus$ where column s is input symbol and row j is the number of steps along the predefined path.

The SPTC uses a target point t_k selected from a sequence of target points within the signal space to calculate the required number of steps to go from any point s_v to that target point. The sequence of target points may consist of a single point (a sequence of length 1) or multiple points. If the sequence of target points consists of multiple points, then at each i the target point t_k is time varying with $t_k = t_{i \ (mod \ u)}$, where u is the length of the sequence of target points. In this thesis, fixed target point $t = 0$ has been used in all encoding even though the target point can follow a shape by selecting appropriate

points from the signal constellation as the sequence of target points. For example, points $\{2, 6, 14, 10, 11, 9, 8, 12, 4, 0, 1, 3, 2\}$ are selected as sequence of target points so that the target point traces a square shape shown in Fig. 2.2.

Fig. 2.2: 16-QAM signal constellation with a square shape indicated by blue arrows which can be traced by target point.

Let $\ominus : \mathcal{I} \times \mathcal{I} \to \mathbb{Z}_{\geq 0}$ be defined as the operation $r = t_k \ominus s_v$ that calculates the number of steps r along the path necessary to move from point s_v to a point t_k, where, $t_k, s_v \in \mathcal{I}$ and $r \in \mathbb{Z}_{\geq 0}$. For example, for the shape shown in Fig. 2.1, $4 \ominus 5 = 5$ because from point 5 in the signal space, it takes 5 steps along the path indicated by the blue arrows to reach point 4. For fixed target points $t = 0$, $t = 1$, ..., $t = 15$, the required number of movements from each point in the signal constellation along the path shown in Fig. 2.1 is listed in the

Table 2.2.

$\ominus$	a															
t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	7	15	8	1	6	14	9	3	4	12	11	2	5	13	10
1	9	0	8	1	10	15	7	2	12	13	5	4	11	14	6	3
2	1	8	0	9	2	7	15	10	4	5	13	12	3	6	14	11
3	8	15	7	0	9	14	6	1	11	12	4	3	10	13	5	2
4	15	6	14	7	0	5	13	8	2	3	11	10	1	4	12	9
5	10	1	9	2	11	0	8	3	13	14	6	5	12	15	7	4
6	2	9	1	10	3	8	0	11	5	6	14	13	4	7	15	12
7	7	14	6	15	8	13	5	0	10	11	3	2	9	12	4	1
8	13	4	12	5	14	3	11	6	0	1	9	8	15	2	10	7
9	12	3	11	4	13	2	10	5	15	0	8	7	14	1	9	6
10	4	11	3	12	5	10	2	13	7	8	0	15	6	9	1	14
11	5	12	4	13	6	11	3	14	8	9	1	0	7	10	2	15
12	14	5	13	6	15	4	12	7	1	2	10	9	0	3	11	8
13	11	2	10	4	12	1	9	4	14	15	7	6	13	0	8	5
14	3	10	2	11	4	9	1	12	6	7	15	14	5	8	0	13
15	6	13	5	14	7	12	4	15	9	10	2	1	8	11	3	0

Table 2.2: Input output relation table for $\ominus$ where column t is the input target point and row a is the input current point in the signal space.

For 64 QAM, a shape similar to Fig. 2.1 is shown by the blue arrows in Fig. 2.3. For this shape, Tables 2.3, 2.4, 2.5, 2.6 contain the outputs of the $\oplus$ operator and the required number of movement to reach fixed target point $t = 0$ from every point in the signal constellation is listed in Tables 2.7 and 2.8.
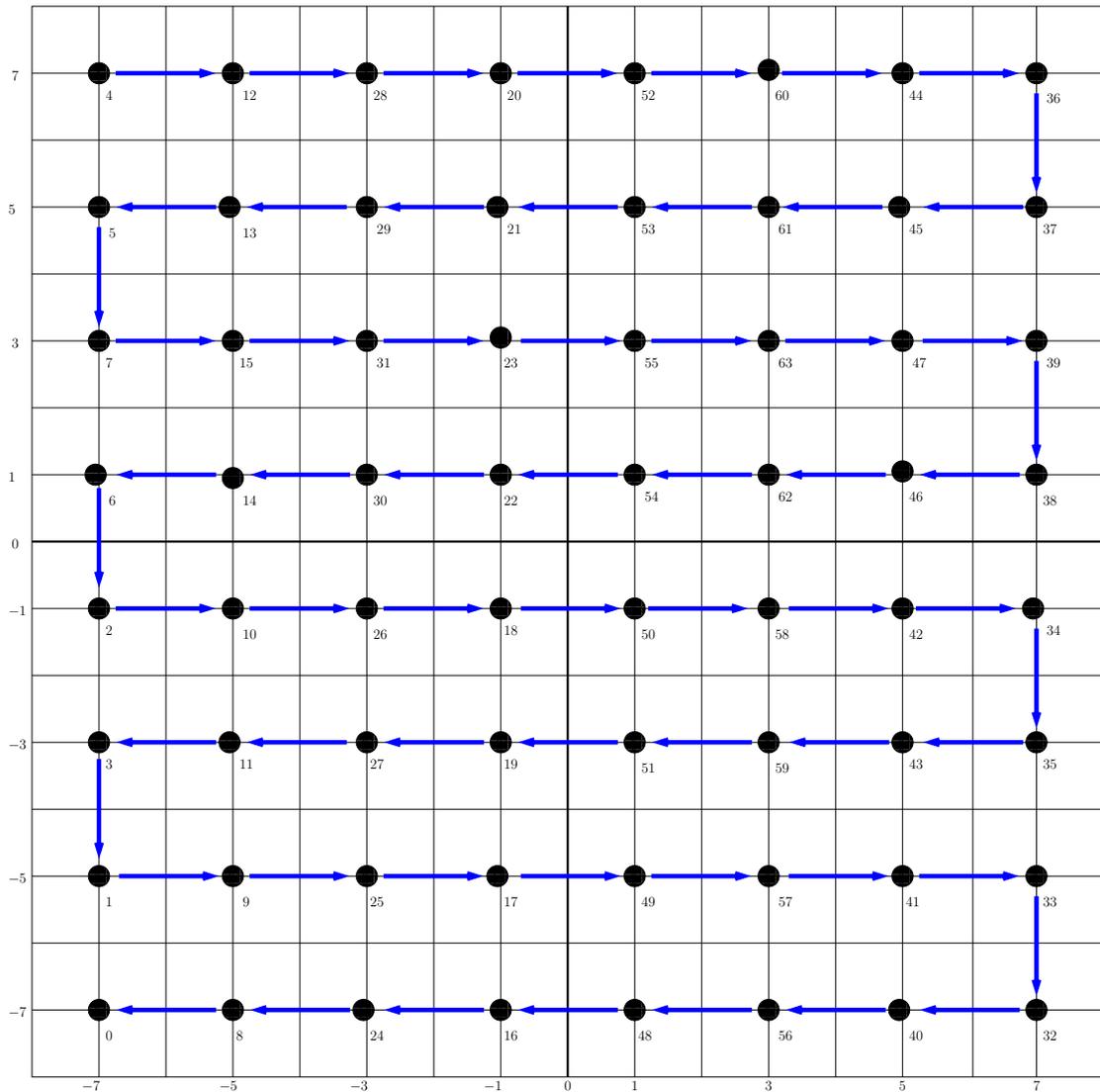
Fig. 2.3: 64-QAM signal constellation with a shape indicated by blue arrows.

⊕	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14
1	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5
2	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0
3	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13
4	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6
5	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11
6	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8
7	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3
8	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30
9	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7
10	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4
11	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29
12	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2
13	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27
14	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24
15	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1
16	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54
17	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31
18	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28
19	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53
20	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26
21	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51
22	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48
23	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25
24	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22
25	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15
26	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12
27	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21
28	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10
29	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19
30	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16
31	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9

Table 2.3: Input output relation table for ⊕ where column s is input symbol and row j is the number of steps along the predefined path.

⊕		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
	0	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8
	1	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3
	2	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6
	3	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11
	4	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0
	5	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13
	6	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14
	7	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5
	8	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24
	9	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1
	10	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2
	11	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27
	12	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4
	13	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29
	14	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30
s	15	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7
	16	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48
	17	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25
	18	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26
	19	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51
	20	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28
	21	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53
	22	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54
	23	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31
	24	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16
	25	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9
	26	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10
	27	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19
	28	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12
	29	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21
	30	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22
	31	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15

Table 2.4: Input output relation table for $\oplus$ where column s is input symbol and row j is the number of steps along the predefined path.

$\oplus$																			j													
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39
33	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47
34	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44
35	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36
36	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42
37	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34
38	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33
39	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41
40	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38
41	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63
42	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60
43	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37
44	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58
45	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35
46	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32
47	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57
48	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62
49	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23
50	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20
51	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61
52	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18
53	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59
54	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56
55	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17
56	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46
57	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55
58	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52
59	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45
60	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50
61	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43
62	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40
63	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49

Table 2.5: Input output relation table for $\oplus$ where column s is input symbol and row j is the number of steps along the predefined path.

⊕	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
32	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33
33	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41
34	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42
35	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34
36	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44
37	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36
38	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39
39	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47
40	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32
41	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57
42	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58
43	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35
44	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60
45	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37
46	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38
47	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63
48	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56
49	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17
50	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18
51	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59
52	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20
53	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61
54	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62
55	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23
56	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40
57	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49
58	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50
59	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46	62	54	22	30	14	6	2	10	26	18	50	58	42	34	35	43
60	58	42	34	35	43	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52
61	59	51	19	27	11	3	1	9	25	17	49	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45
62	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55	63	47	39	38	46
63	57	41	33	32	40	56	48	16	24	8	0	4	12	28	20	52	60	44	36	37	45	61	53	21	29	13	5	7	15	31	23	55

(s is the row label for the table above.)

Table 2.6: Input output relation table for $\oplus$ where column s is input symbol and row j is the number of steps along the predefined path.

⊖	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
t 0	0	15	31	16	63	48	32	47	1	14	30	17	62	49	33	46	3	12	28	19	60	51	35	44	2	13	29	18	61	50	34	45

Table 2.7: Input output relation table for $\ominus$ where column t is the input target point and row a is the input current point in the signal space.

	a																															
⊖	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
t 0	7	8	24	23	56	55	39	40	6	9	25	22	57	54	38	41	4	11	27	20	59	52	36	43	5	10	26	21	58	53	37	42

Table 2.8: Input output relation table for $\ominus$ where column t is the input target point and row a is the input current point in the signal space.

The $\oplus$ and $\ominus$ operations can be viewed on an M sided regular polygon where the vertices are labeled in clockwise order as the points of the path defined in the signal constellation. For example, the path from Fig 2.1 can be mapped to the vertices of hexadecagon shown in Fig. 2.4. Now $7 \oplus 6 = 8$ means moving 6 edges from vertex labeled as 7 in the clockwise direction will lead to vertex 8 and $4 \ominus 5 = 5$ means there are 5 edges between vertex 5 and 4 in the clockwise direction.
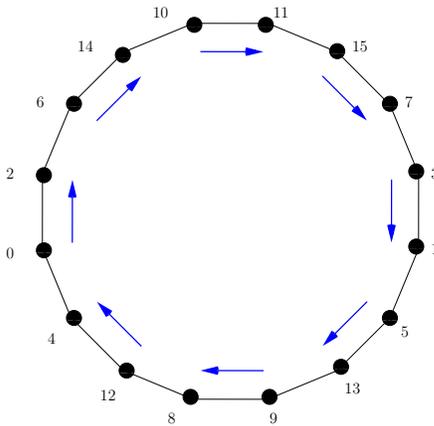

Fig. 2.4: Representation $\oplus$ and $\ominus$ operation on a M sided regular polygon.

The encoder has a state variable $j_i \in \{0, 1, 2, ..., M-1\}$ which determines the location along the path in the trellis after encoding for each time i. At each time i, the path starts at state variable j_i and ends at j_{i+1}. The state variable j_{i+1} is calculated by

$$j_{i+1} = j_i + r_i \pmod{M}. \tag{2.2}$$

The state variable accumulates parity symbol index r_i generated at time step i and the previous state j_i to determine where the path is going to end at that time step. The endpoint for a branch depends on the input symbol S_i, the state variable j_i and the target point t. For a fixed j_i, a different combination of S_i and t will lead to a different endpoint j_{i+1}. So at each i, j_i and j_{i+1} determine which branch of the trellis is taken during the encoding thus decides the direction of the path in the trellis. For this encoder, the number of states is equal to the number of points in the signal constellation.

For a given shape in the signal constellation and $\{t_0, t_1, t_2, t_3, ...\}$ as the sequence of target points, the encoding operation for SPTC at every symbol time i is given in Algorithm 1.

Algorithm 1 Signal Point Target Code Encoding

1: input: symbol S_i , previous state variable j_i

2: $s_i \leftarrow \psi(S_i)$ compute the integer index of S_i

3: $a_i \leftarrow s_i \oplus j_i$

4: determine the target point t_k where $k = i \pmod{u}$

5: $r_i \leftarrow t_k \ominus a_i$

6: $j_{i+1} \leftarrow j_i + r_i \pmod{M}$ update the state after encoding

7: $R_i \leftarrow \psi^{-1}(r_i)$ compute the parity symbol R_i from the integer index r_i

8: return parity symbol R_i and state variable j_{i+1}

Associated with each branch at the time i, there is an input symbol S_i and parity symbol R_i. An example of trellis using QPSK constellation is shown in Fig. 2.5. In this trellis all of the possible branches at the time i that can go from every initial state j_i to every final state j_{i+1} is represented by lines. Starting from the initial state 1, the branch which goes to the final state 2 is presented by a black solid line. The input symbol S_i and parity symbol R_i associated with this branch is presented on top of the branch labeled as S_i/R_i in the figure. Since the endpoint of each branch in the trellis depends on the state variable and the state variable depends on the target point, the trellis is time varying if

the sequence of target points contains multiple points. Initially, the encoder starts at state $j_0 = 0$ and with each increment of time i it creates a path which allows a trellis based decoder to determine the sequence of transmitted symbols. At each i, the encoder output is the input symbol S_i and parity symbol R_i which makes the encoder of rate $1/2$.
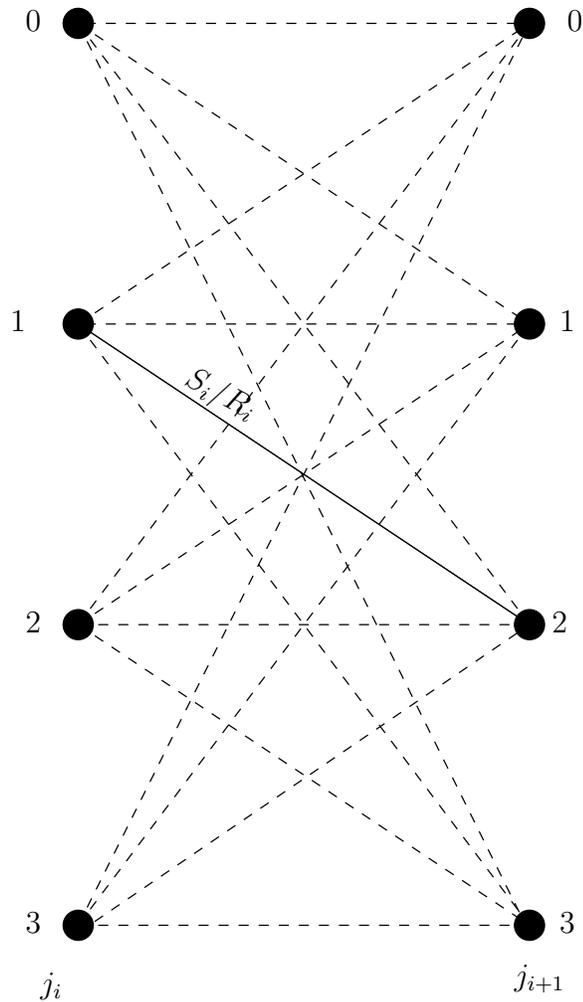

Fig. 2.5: Example of trellis structure using a QPSK constellation.

2.2 Decode

To decode SPTC, a trellis is formed in which for every symbol time i all of the branches that can go from every possible initial state j_i to every possible final state j_{i+1} are calculated. Those branches store their corresponding input symbols and parity symbols. For QPSK signal constellation with the shape shown in Fig. 2.6 and a fixed target point $t = \{0\}$, the trellis is shown in Fig. 2.7, where each branch from state j_i to state j_{i+1} is labeled as S_i/R_i which corresponds to the branch input symbol and parity symbol.
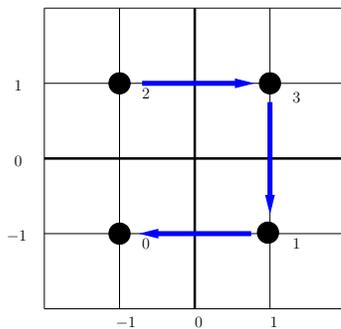

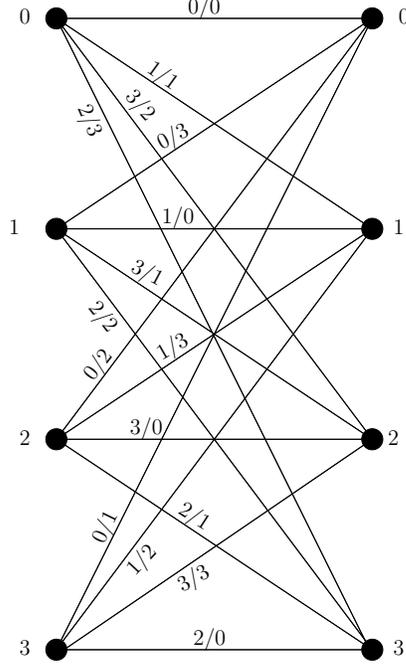Fig. 2.6: QPSK signal constellation with a shape indicated by blue arrows.

Fig. 2.7: Trellis structure for SPTC using QPSK constellation for fixed target point $t = 0$ and shape shown in Fig. 2.6.

In order to decode using the formed trellis, a standard Viterbi algorithm [9] has been implemented which uses squared Euclidean distance between branch input symbol S_i and received symbol $\hat{S}_i$ plus squared Euclidean distance between branch parity symbol R_i and received parity symbol $\hat{R}_i$ as branch metric

$$\mu_t = d_e^2(S_i, \hat{S}_i) + d_e^2(R_i, \hat{R}_i). \qquad (2.3)$$

Here squared Euclidean distance is used to calculate branch metric which is also a measure of the energy difference between the symbols. The Viterbi decoder starts at all zero-state and the trellis depth of the Viterbi algorithm is 10. Simulation results for the proposed code using different trellis depth in the Viterbi decoder are presented in Fig. 3.16. From Fig. 3.16, it can be seen that the performance of the decoder remains almost the same after a certain trellis depth. So trellis depth of 10 for the Viterbi algorithm has been selected

to decode. For each state j_{i+1} at time i branch metric μ_t is calculated for all the possible branches originated from each state j_i at time $i-1$ and the branch metric added with the path metric $M(j_i)$ for each surviving path at time $i-1$ to get path metric $M(j_{i+1})$ and the path to state j_{i+1} with the minimum path metric is selected as the surviving path at time i. The optimum output sequence is the path which has the minimum path metric through the trellis.

2.3 Free Euclidean Distance

Similar to convolutional codes and TCM, the performance of SPTC in terms of coding gain depends on the smallest distance between any two paths which diverge and then merge into the same state in the trellis. This smallest distance is also called the free Euclidean distance [9] and is denoted by d_{free}^2. The distance between these two paths is the sum of squared Euclidean distances in the signal constellation between the input symbols associated with these paths and the parity symbols associated with these paths. In Fig. 2.8, the Euclidean distances between two pair of points are shown by blue lines where the average energy per coded symbol is $E_{c,s}$. For this constellation, $d_e(0,1) = d_e(3,1) = d_e(2,0) = d_e(2,3) = 2\sqrt{E_{c,s}}$ and $d_e(2,1) = d_e(0,3) = \sqrt{8E_{c,s}}$. For example, for the trellis shown in Fig. 2.7, among all of these diverging and then merging paths, the pair of paths which has the smallest distance is indicated by black solid lines in Fig. 2.9. In Fig. 2.9, starting from state 0 one path goes through state 0 which has input symbol 0 and parity symbol 0 while the other path diverges to state 1 with input symbol 1 and parity symbol 1. Then the second path merges back to state 0 with input symbol 0 and parity symbol 3. So using the Euclidean distances from the signal constellation shown in Fig. 2.8, the free Euclidean distance

$$d_{\text{free}}^2 = d_e^2(0,1) + d_e^2(0,1) + d_e^2(0,0) + d_e^2(0,3)$$

$$= 4E_{c,s} + 4E_{c,s} + 0 + 8E_{c,s}$$
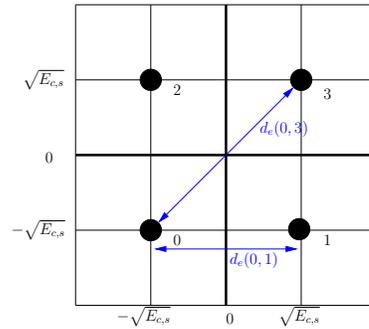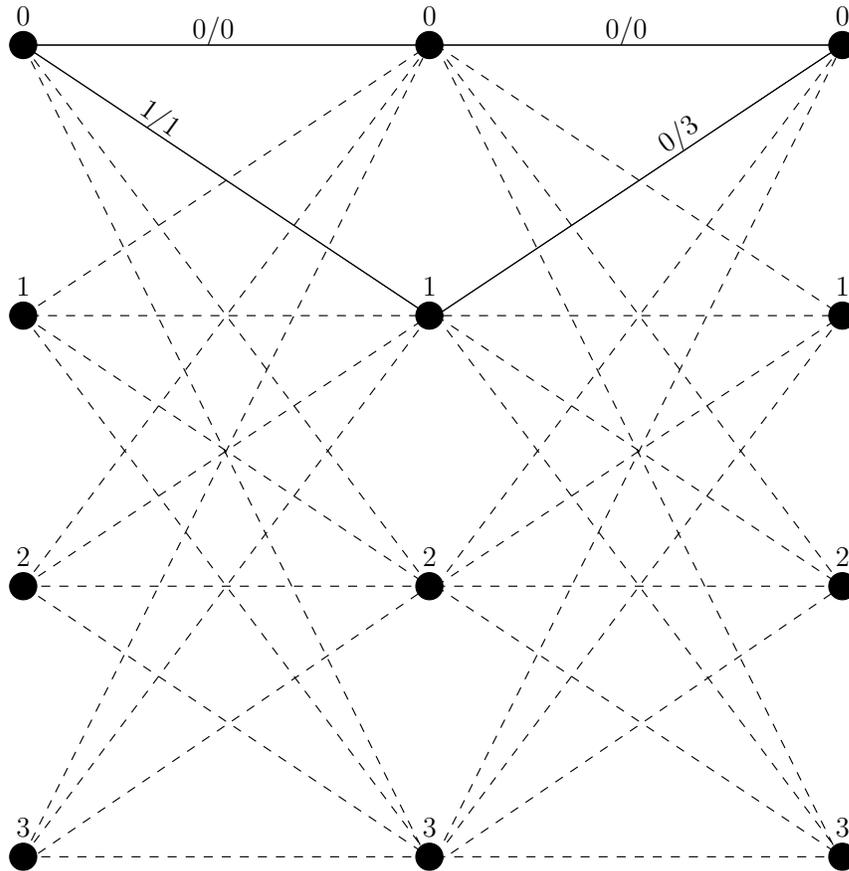
$$= 16E_{c,s}.$$

Fig. 2.8: QPSK constellation.

Fig. 2.9: Branches (black solid line) on the trellis that results in d_{free}^2 for SPTC on QPSK for fixed target point $t = 0$ and shape shown in Fig. 2.6.

For a given bit assignment in the signal constellation, different shapes result in different trellis structure. So each shape corresponds with it's own d^2_{free} and from the simulation results presented in chapter 3, it can be seen that the shape with larger d^2_{free} has better coding gain than other shapes. For example, the trellis and the pair of paths that produces free Euclidean distance for the shape shown in Fig. 2.10 are presented in Figs. 2.11 and 2.12. For this shape, the free Euclidean distance on the trellis using the signal constellation from Fig. 2.8 is

$$
\begin{aligned}
d^2_{\text{free}} &= d^2_e(2,1) + d^2_e(2,3) + d^2_e(1,2) + d^2_e(2,1) \\
&= 8E_{c,s} + 4E_{c,s} + 8E_{c,s} + 8E_{c,s} \\
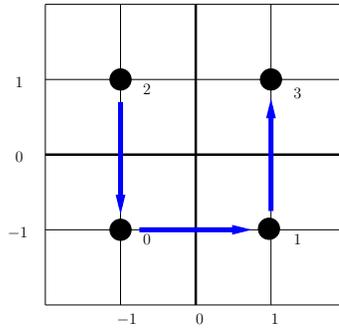&= 28E_{c,s}
\end{aligned}
$$

Fig. 2.10: QPSK signal constellation with a shape indicated by blue arrows.
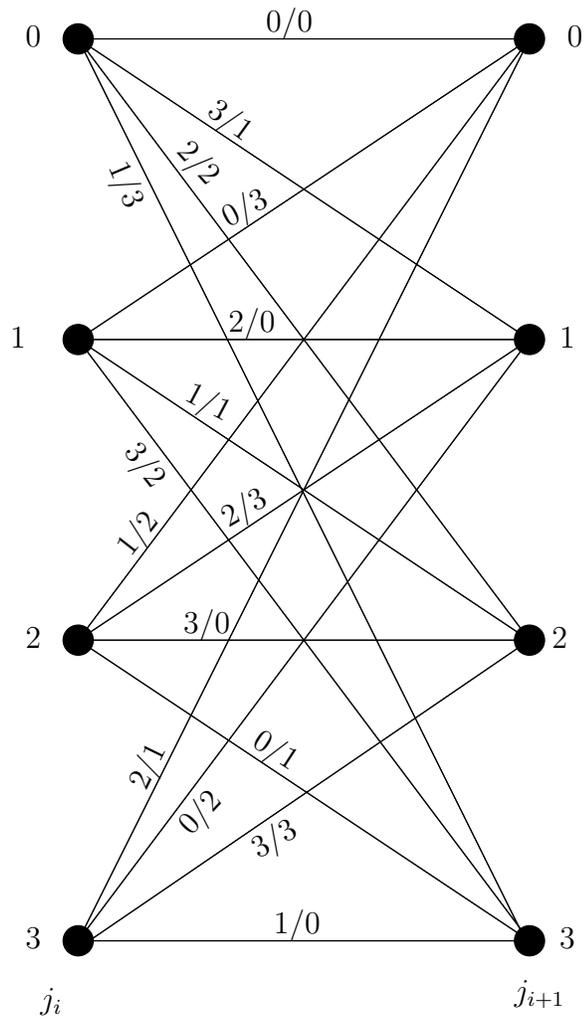
Fig. 2.11: Trellis structure for SPTC using QPSK constellation for fixed target point $t = 0$ and shape shown in Fig. 2.10.

Fig. 2.12: Branches (black solid line) on the trellis that results in d_{free}^2 for SPTC on QPSK for fixed target point $t = 0$ and shape shown in Fig. 2.10.

By rotating the target point after each input signal point does not yield an improved coding gain because the relative distance between branches within a given code trellis is the same. So a constant target point $t = 0$ has been used in this thesis to encode the input symbols. For 16QAM constellation, the shape and bit assignment in the signal constellation shown in Fig. 2.1 and fixed target point $t = 0$, the pair of paths which constitutes the free Euclidean distance is shown in Fig. 2.13 by black solid lines. For this shape, the free Euclidean distance is

$$d_{\text{free}}^2 = d_e^2(4, 13) + d_e^2(1, 5) + d_e^2(0, 0) + d_e^2(15, 11)$$

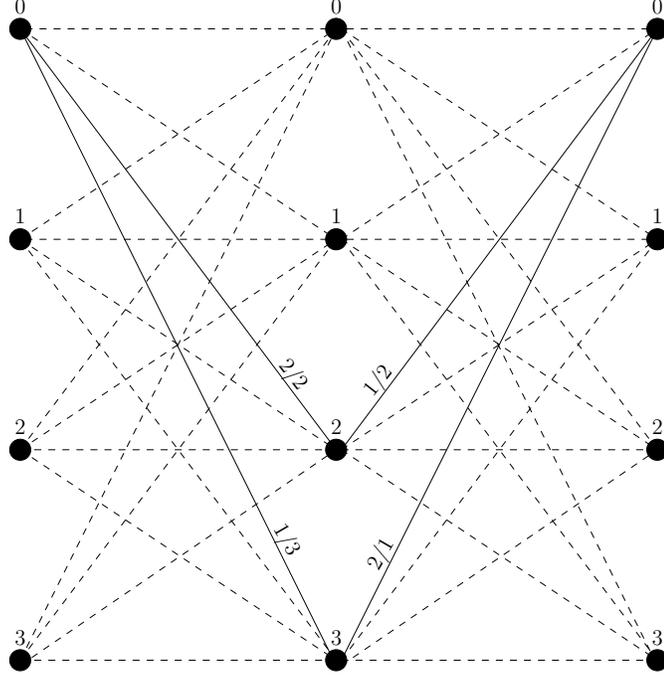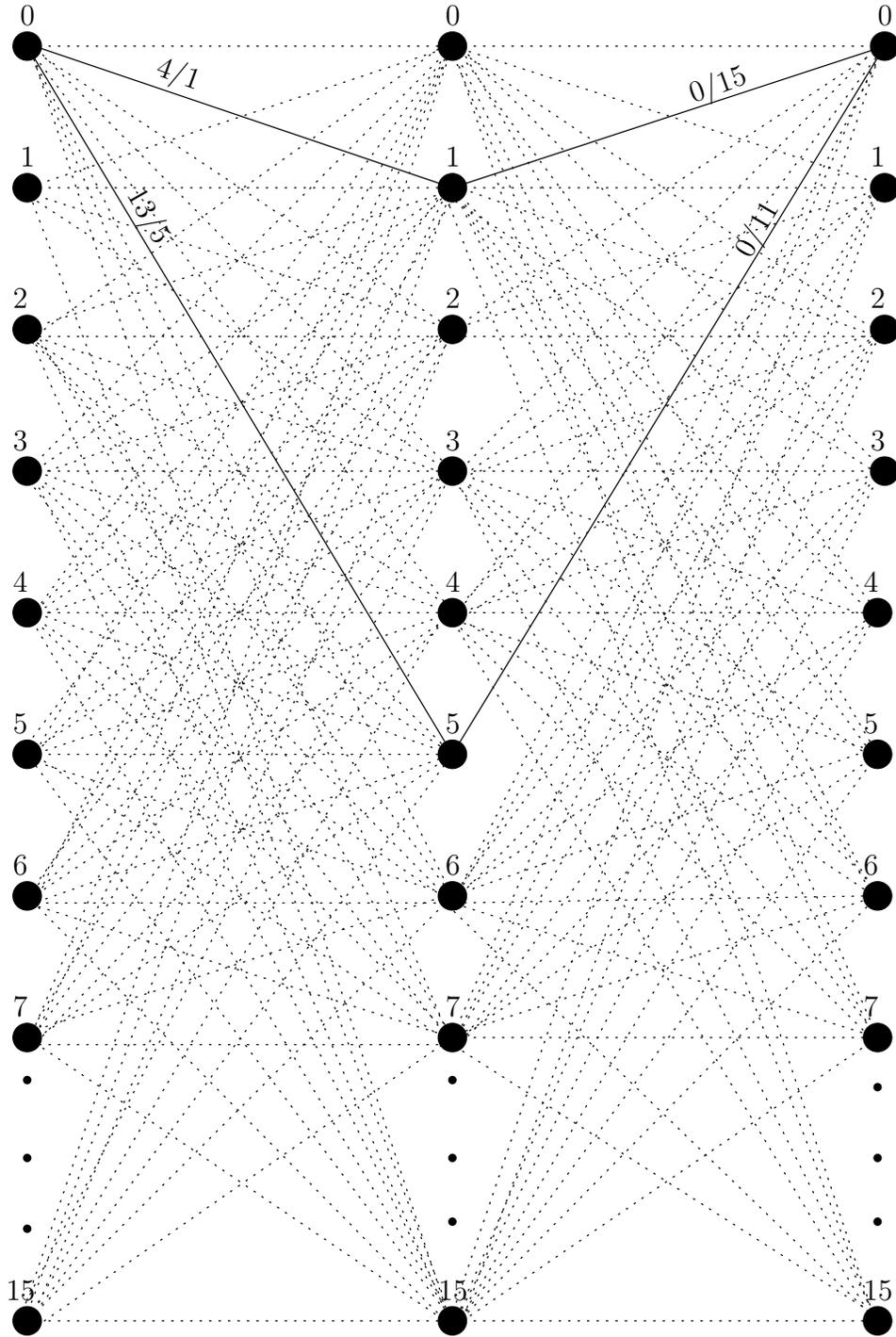$$= 8E_{c,s} + 4E_{c,s} + 0 + 4E_{c,s}$$

$$= 16E_{c,s}$$

Fig. 2.13: Branches (black solid line) on the trellis that results in d_{free}^2 for SPTC on 16QAM for fixed target point $t = 0$ and shape shown in Fig. 2.1.

2.4   Generalization to any M-ary QAM

As the performance of SPTC depends on d_{free}^2, finding the shape and the symbol assignment in the signal constellation corresponding to that shape which has the largest free Euclidean distance will result in maximum possible coding gain. However, for a signal constellation of M points, there are $M!$ possible ways the bits can be assigned to signal constellation. As the shapes go through all of the points in the signal constellation as well as they can not go through each point more than once, there are $M!$ shapes which can be used to encode. Finding the best shape that maximizes the d_{free}^2 for a given bit assignment in the signal constellation is equivalent to finding the arrangement of symbols in the signal constellation that also maximizes d_{free}^2 for a given shape in term of search space. As the $\oplus$ and $\ominus$ operations depend on the employed shape, reduction of search space for finding the larger d_{free}^2 is possible by employing an encoding scheme that depends only on bit assignment rather than shape and then by finding the bit assignment which maximizes the free Euclidean distance for that encoding operation. From Tables 2.1 and 2.2, the output of $\oplus$ and $\ominus$ operations are always an integer ranging from 0 to $M-1$ and all of the integers from 0 to $M-1$ appears exactly once in each row. So it can be concluded that the encoding works as long as the outputs of $\oplus$ and $\ominus$ satisfy:

- For a fixed state variable j, the output of the $\oplus$ operation is unique for each unique input symbol index s.

- For a fixed target point t, the output of the $\ominus$ operation is unique for each unique signal space index a.

Using the above-mentioned properties, the encoding was simplified by replacing $\oplus$ and $\ominus$ with following equations:

$$s_i \oplus j = s_i + j \ (\text{mod } M) \tag{2.4}$$

$$t_k \ominus a_i = t_k + a_i \ (\text{mod } M) \tag{2.5}$$

This modified encoding depends on how the bits are assigned in the signal constellation and doesn't require any shape to encode. To distinguish between this modification and SPTC,

this new encoding will be mentioned as Constellation Arithmetic Code (CAC) as all the encoding is computed using arithmetic on the signal constellation. The encoding is given below:

Algorithm 2 Constellation Arithmetic Code Encoding

1: input to the encoder: symbol S_i , previous state variable j_i

2: $s_i \leftarrow \psi(S_i)$ compute the integer index of S_i

3: $a_i \leftarrow s_i + j_i \pmod M$

4: determine the target point t_k where $k = i \pmod u$

5: $r_i \leftarrow t_k + a_i \pmod M$

6: $j_{i+1} \leftarrow j_i + r_i \pmod M$ update the state after encoding

7: $R_i \leftarrow \psi^{-1}(r_i)$ compute the parity symbol R_i from the integer index r_i

8: return parity symbol R_i and state variable j_{i+1}

In (2.5), adding $t = 0$ yields to the same result from (2.4) and using constant target points other than $t = 0$ results in different trellis structure with no significant coding gain. However, as the motivation of CAC is to find the best bit assignment for a fixed trellis, further simplification of Constellation Arithmetic Code has been done for calculating the parity symbol index by excluding (2.5) from the encoding operation of CAC.

Algorithm 3 Simplified Constellation Arithmetic Code Encoding

---

1: input to the encoder: symbol S_i , previous state variable j_i

2: $s_i \leftarrow \psi(S_i)$ compute the integer index of S_i

3: $r_i \leftarrow s_i + j_i \pmod M$

4: $j_{i+1} \leftarrow j_i + r_i \pmod M$ update the state after encoding

5: $R_i \leftarrow \psi^{-1}(r_i)$ compute the parity symbol R_i from the integer index r_i

6: return parity symbol R_i and state variable j_{i+1}

The trellis for CAC using QPSK constellation from Fig. 2.6 is presented in Fig. 2.14.
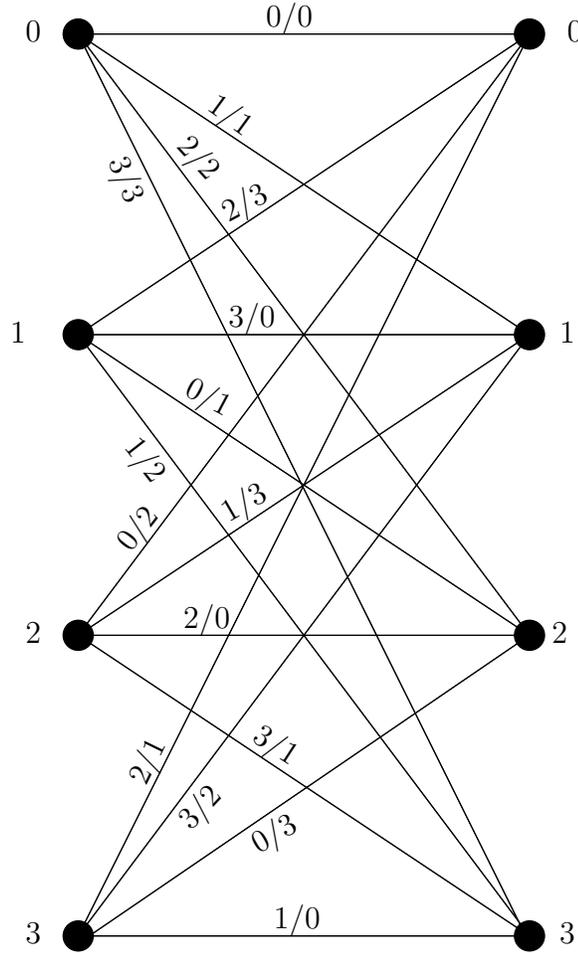
Fig. 2.14: Trellis structure for CAC using QPSK constellation shown in Fig. 2.6.

Now for this trellis and signal constellation the pair of paths on the trellis that has the minimum distance between them is shown in Fig. 2.15 by solid black lines. For this constellation

$$d_{\text{free}}^2 = d_e^2(0,2) + d_e^2(0,2) + d_e^2(0,0) + d_e^2(0,2)$$

$$= 4E_{c,s} + 4E_{c,s} + 0 + 4E_{c,s} \tag{2.6}$$

$$= 12E_{c,s}.$$

From (2.6), d_{free}^2 results from the distance between symbol 0 and 2 in the signal constellation. Symbol 0 is assigned to point $(-1,-1)$ and symbol 2 is assigned to point $(-1,1)$ and the

distance between them is $2\sqrt{E_{c,s}}$ while the distance between the point $(-1,-1)$ and $(1,1)$ is $2\sqrt{2E_{c,s}}$ for average energy per coded symbol $\sqrt{E_{c,s}}$. So it is possible to increase d_{free}^2 by swapping symbol 2 with symbol 3 in the constellation as shown in Fig. 2.16.

Fig. 2.15: Branches (black solid line) on the trellis that results in d_{free}^2 for CAC on QPSK constellation shown in Fig. 2.6.
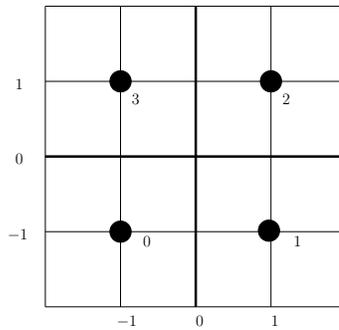
Fig. 2.16: QPSK signal constellation with bit assignment that produces $d_{\text{free}}^2 = 20E_{c,s}$.

For the signal constellation shown in Fig. 2.16 the minimum distance between sequences on the trellis is shown in Fig. 2.17 and the free Euclidean distance

$$d_{\text{free}}^2 = d_e^2(0,1) + d_e^2(0,1) + d_e^2(0,2) + d_e^2(0,3)$$

$$= 4E_{c,s} + 4E_{c,s} + 8E_{c,s} + 4E_{c,s}$$

$$= 20E_{c,s}$$

Fig. 2.17: Branches (black solid line) on the trellis that results in d_{free}^2 for CAC on QPSK constellation shown in Fig. 2.16.

Free Euclidean distance corresponding to CAC on 16QAM using the bit assignment shown in Fig. 2.1 is $d_{\text{free}}^2 = 12E_{c,s}$. For this bit assignment, the pair of paths that produces the free Euclidean distance on the trellis is shown in Fig. 2.18 by black solid lines.
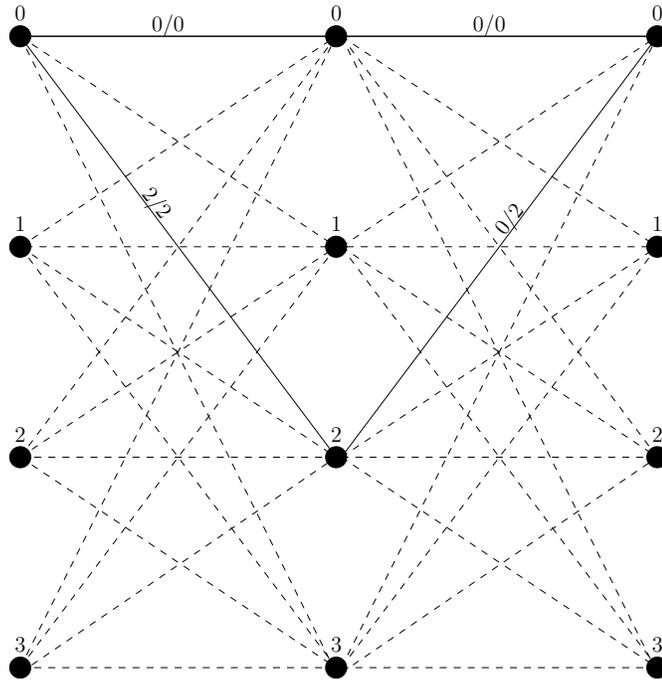
Fig. 2.18: Branches (black solid line) on the trellis that results in d_{free}^2 for CAC on 16QAM using constellation shown in Fig. 2.1.
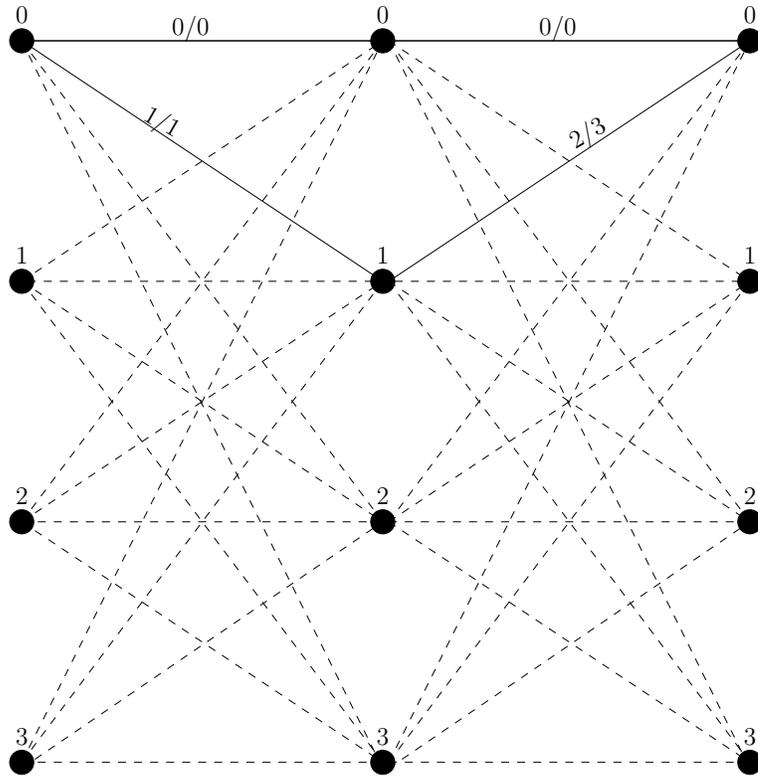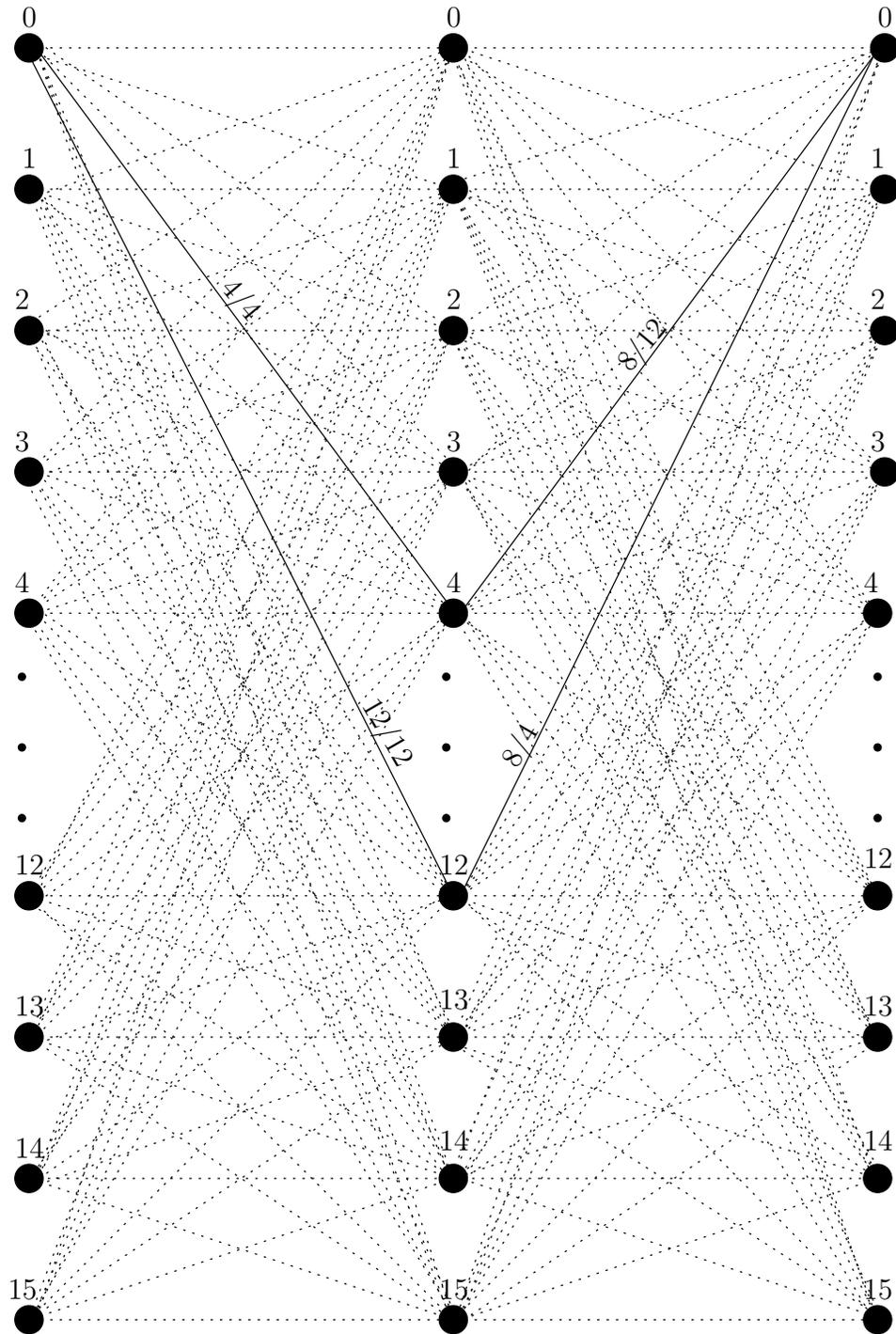
In order to show the relation between free Euclidean distance and coding gain, a bit assignment is shown in Fig. 2.19. This bit assignment produces $d_{\text{free}}^2 = 20E_{c,s}$ which is larger than the free Euclidean distance results from using gray code indexing as shown in Fig. 2.1. There are 16! possible bit assignments in the 16QAM constellation. So finding the bit assignment which has the largest d_{free}^2 among those 16! bit assignments are computationally expensive. In order to find bit assignments in the 16QAM constellation with larger d_{free}^2, 4 symbols are selected arbitrarily and they have been fixed at points $(-3, -3)$, $(3, -3)$, $(-3, 3)$ and $(3, 3)$ respectively in order to reduce the search space from 16! to 12! and find a bit assignment with larger d_{free}^2 rather than finding the largest d_{free}^2. Then an exhaustive search has been made using the other 12 symbols and points in the signal constellation to find possible bit assignment with largest d_{free}^2 from all possible 12! symbol assignments. By randomly selecting those 4 symbols, it has been found that by selecting symbols 0, 3, 12 and 15 and fixing those symbols to points $(-3, -3)$, $(3, -3)$, $(-3, 3)$ and $(3, 3)$, it is possible to produce two bit assignments with $d_{\text{free}}^2 = 28E_{c,s}$. Those two bit assignments are shown in Fig. 2.20 and 2.21. For 2.20, the pair of paths that produces the free Euclidean distance on the trellis is shown in Fig. 2.22 by black solid lines. The simulated results for these bit assignments with $d_{\text{free}}^2 = 12E_{c,s}$, $d_{\text{free}}^2 = 20E_{c,s}$ and $d_{\text{free}}^2 = 28E_{c,s}$ are presented in Figs. 3.7, 3.8 and 3.9 respectively. From these results, it can be seen that it is possible to increase coding gain for CAC by increasing d_{free}^2.
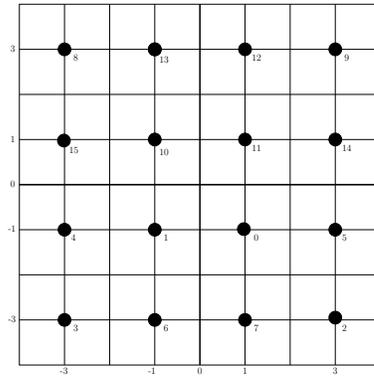
Fig. 2.19: 16-QAM signal constellation with bit assignment that produces $d_{\text{free}}^2 = 20E_{c,s}$.
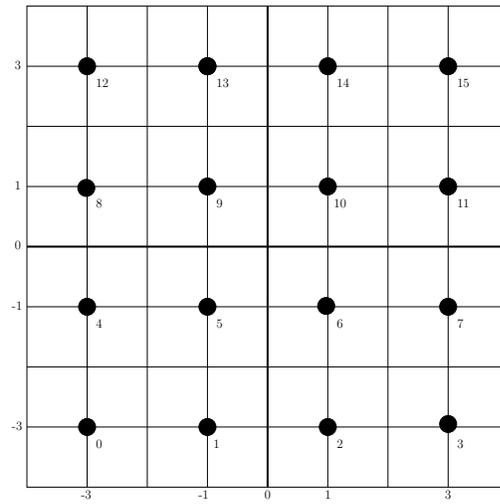
Fig. 2.20: 16-QAM signal constellation with bit assignment that produces $d_{\text{free}}^2 = 28E_{c,s}$.
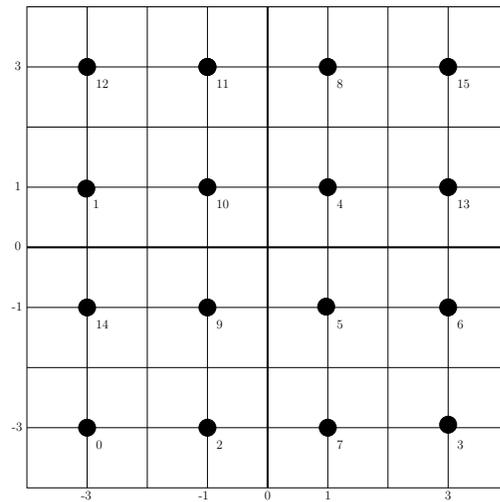
Fig. 2.21: 16-QAM signal constellation with bit assignment that produces $d_{\text{free}}^2 = 28E_{c,s}$.
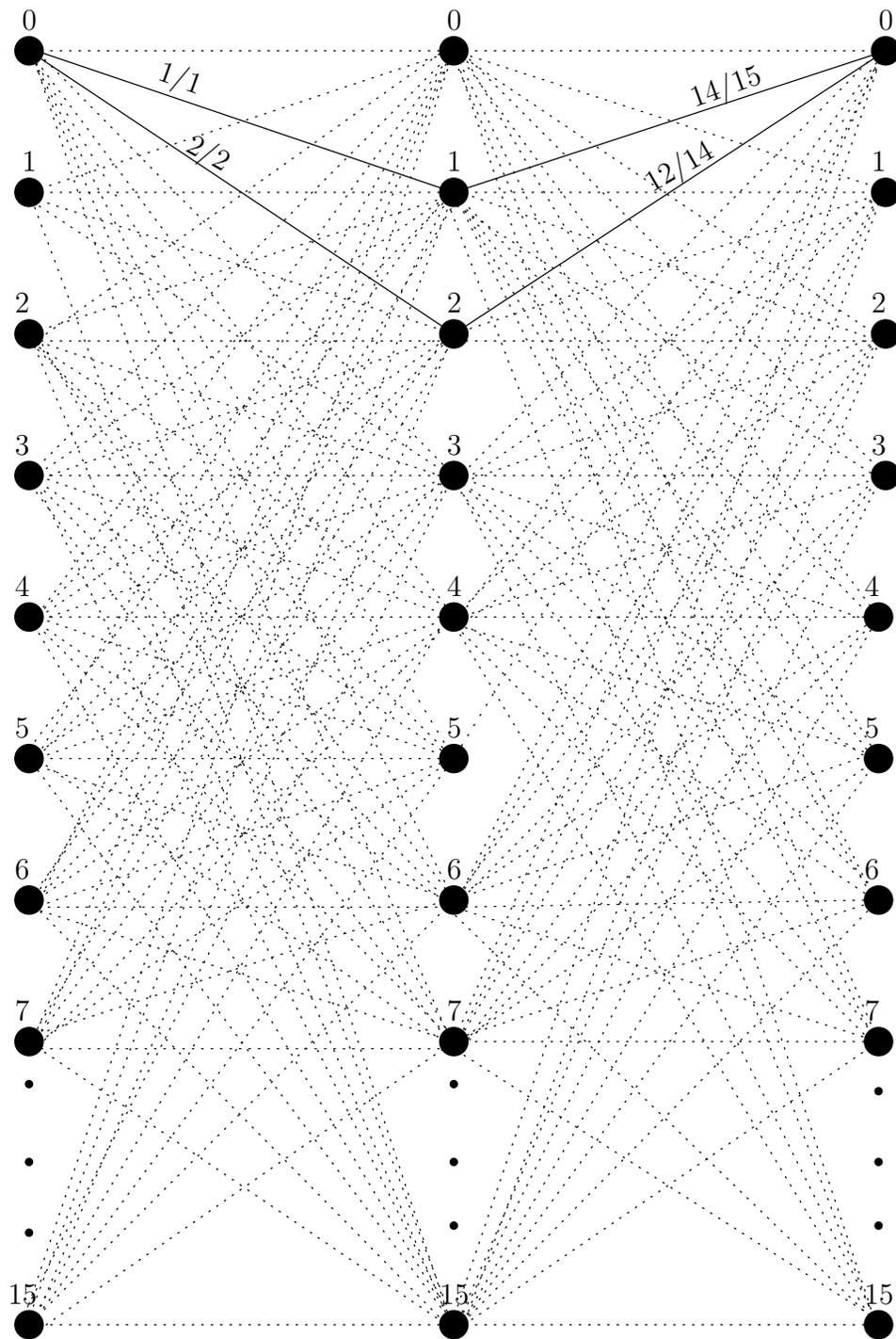
Fig. 2.22: Branches (black solid line) on the trellis that results in d_{free}^2 for CAC on 16QAM using bit assignment shown in Fig. 2.20.

In Fig. 2.20, the bits are assigned in the signal constellation as follows: first, the combination of bits that are to be assigned are represented in decimal integer values. Then starting from the bottom left point in the signal constellation and moving towards right those integer representation of bits are assigned to the points in ascending order. When all the points of that row have their assigned bits, the immediate upper row in the signal constellation is selected for bit assignment. This operation is performed until all the points in the signal constellation have their assigned bits. This method of bits assignment using CAC provides the maximum coding gain compared to all of the simulated performance in this thesis for the 16QAM constellation and a coding gain for CAC on 64QAM compared to SPTC. It is denoted as "Sequential Bits Assignment". Sequential Bit Assignment on a 64QAM constellation shown in Fig. 2.23, produces $d_{\text{free}}^2 = 28E_{c,s}$ for CAC.
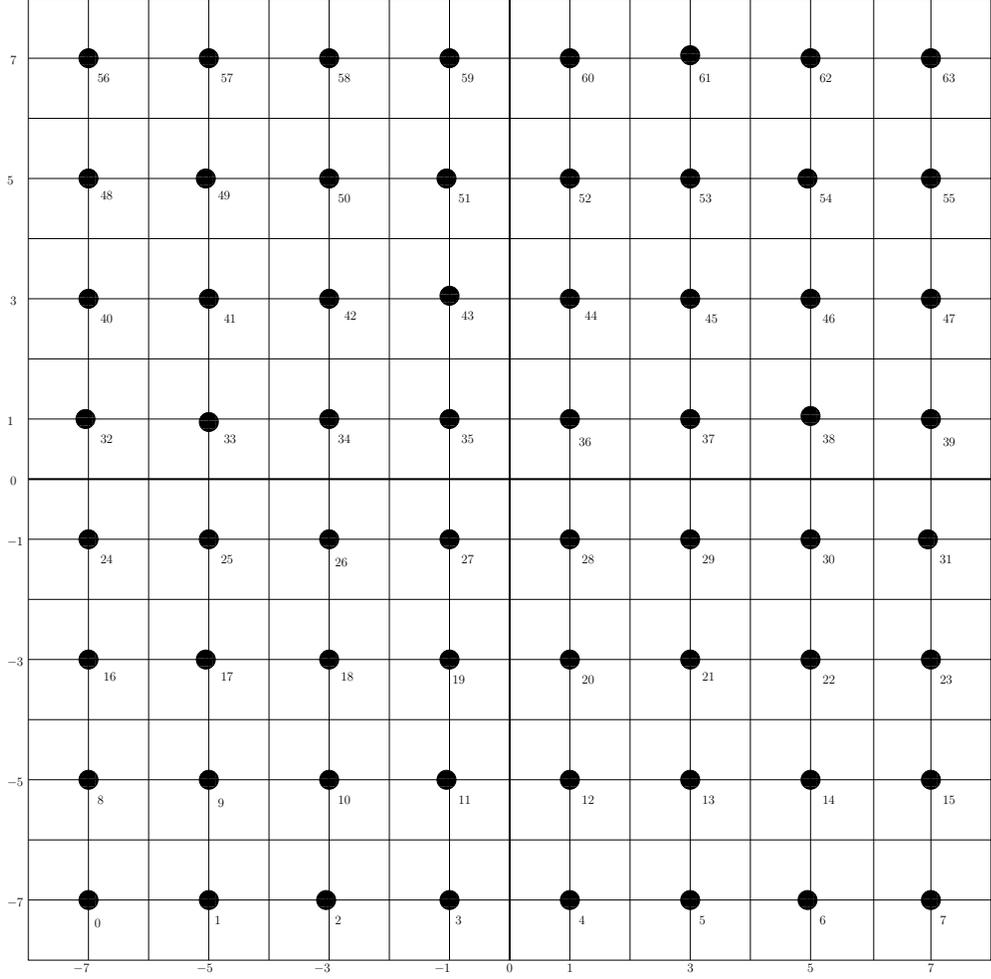
Fig. 2.23: 64-QAM signal constellation with Sequential Bits Assignment that produces $d_{\text{free}}^2 = 28E_{c,s}$.

Now, as the coding gain depends on d_{free}^2, the maximum coding gain for a given free Euclidean distance is bounded by the asymptotic coding gain [9] which is calculated as follows:

$$\gamma = \left(\frac{E_{s,m}}{E_{c,s}}\right)\left(\frac{d_{\text{free}}^2}{d_{\text{min,uncoded}}^2}\right). \tag{2.7}$$

Here d_{free}^2 is the free Euclidean distance and $d_{\text{min,uncoded}}^2$ is the smallest distance between any two points in the constellation. $E_{s,m}$ is the energy per uncoded symbol and $E_{c,s}$ is the total

energy required to send coded symbols. For code with rate R and a given constellation, $1/R$ coded symbols are transmitted for every message symbol which requires $1/R$ times more energy than for an uncoded symbol transmitted using that constellation. Therefore, $\frac{E_{s,m}}{E_{c,s}} = R$. For the constellations used in this thesis, $d^2_{\text{min,uncoded}}$ is always $4E_{c,s}$. Using the free Euclidean distances obtained for different shapes and bit assignments, the asymptotic coding gains for different codes along with the asymptotic coding gains for larger states (which will be introduced in Chapter 4) have been listed in Table 4.1. This list provides a theoretical upper bounds for coding gains.

2.5 Encoding of Rate $\frac{k}{k+1}$ Code

Initially, the proposed encoders of both SPTC and CAC are of rate $\frac{1}{2}$ because for each input symbol the encoder generates one parity symbol. Though lower rate increases the error correction capability of codes, it decreases the information transmission rate per coded symbol. Therefore increasing the rate of codes while maintaining a good coding gain is of significant interest in error correction coding. In this section, two methods for changing rate from $\frac{1}{2}$ to $\frac{k}{k+1}$ using Constellation Arithmetic Code have been proposed. In the encoding algorithms shown in Figs. 2.24 and 2.25, all the inputs s and outputs r are integer indexes of the input symbol S and the output parity symbol R and the $\sum$ represents $a + b \ (mod \ M)$. For this section, at every time i, k symbols are to encoded to generate a parity symbol R_i which makes the proposed encodes of rate $\frac{k}{k+1}$. The encoding methods are described below.

2.5.1 Sequential Input Encoding

In this encoding scheme, only one target point is used to calculate the parity symbol index. The motivation for this encoding is to first encode all of the input symbols sequentially and then generate a parity symbol using only one target point and the output obtained from the sequential encoding of the input symbols. As this method encodes the input symbols sequentially to generate a parity symbol, it is called Sequential Input Encoding (SIE). To encode, each of k input symbol is mapped to their corresponding signal space index. Then

among the k signal space indexes, the signal space index corresponding to the first input symbol is added with the state variable j_i using (2.4) to move to a new signal space index a corresponding to a point of the signal constellation. Then for the rest of $k-1$ symbols, this a is updated by adding previous a with the next symbol's signal space index using (2.4). Finally this updated signal space index a is added with the target symbol's signal space index using (2.5) to get the signal space index r of the parity symbol R. This encoding process is shown in Fig. 2.24. This encoding is also presented in algorithm 4.
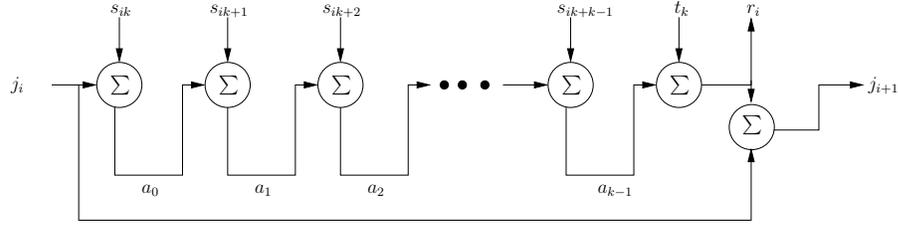
Fig. 2.24: Schematic diagram for rate $\frac{k}{k+1}$ using Sequential Input Encoding (SIE).

Algorithm 4 Rate k/k+1: Sequential Input Encoding (SIE)

1: input to the encoder: symbol $S_{ik}, S_{ik+1}, ..., S_{ik+k-1}$, previous state variable j_i

2: $s_{ik} \leftarrow \psi(S_{ik})$ compute the integer index of S_{ik}

3: $a_0 \leftarrow s_{ik} + j_i \pmod{M}$

4: **for** $l = 1 : k-1$ **do**

5:     $s_{ik+l} \leftarrow \psi(S_{ik+l})$ compute the integer index of S_{ik+l}

6:     $a_l \leftarrow s_{ik+l} + a_{l-1} \pmod{M}$

7: **end for**

8: determine the target point t_k where $k = i \pmod{u}$

9: $r_i \leftarrow t_k + a_{k-1} \pmod{M}$

10: $R_i \leftarrow \psi^{-1}(r_i)$ compute the parity symbol R_i from the integer index r_i

11: $j_{i+1} \leftarrow j_i + r_i \pmod{M}$ update the state after encoding

12: return parity symbol R_i and state variable j_{i+1}

2.5.2   Partial Parallel Encoding

Presented here is another method for changing the rate of CAC. In this method, k input symbols are encoded individually using k target points to generate the parity symbol. All of those k target points might have signal space index of same values or might have different values. For this thesis, while simulating results using this method, target point with 0 signal space index has been selected for all k of those target points. In this encoding, there is a signal space index defined as an intermediate state variable q that acts as a state variable during each encoding of symbol and is used to update the state variable j after generating the parity symbol R. Using target points and q, all of the k input symbols are encoded individually to generate total k different signal space index g. Initially, the q is equal to the input state variable j_i which gets updated after encoding of each symbol and acts as an input state variable for the next symbol. Those k signal space indexes g, generated from each individual encoding of input symbols can be viewed as temporary parity symbol indexes for k input symbols which are added together using $a + b \pmod{M}$ to generate the parity symbol index r. This encoding algorithm is called Partial Parallel Encoding (PPE) because all of the input symbols are encoded in a partially parallel method. The schematic diagram for this encoding is shown in Fig. 2.25 and also presented in algorithm 5.
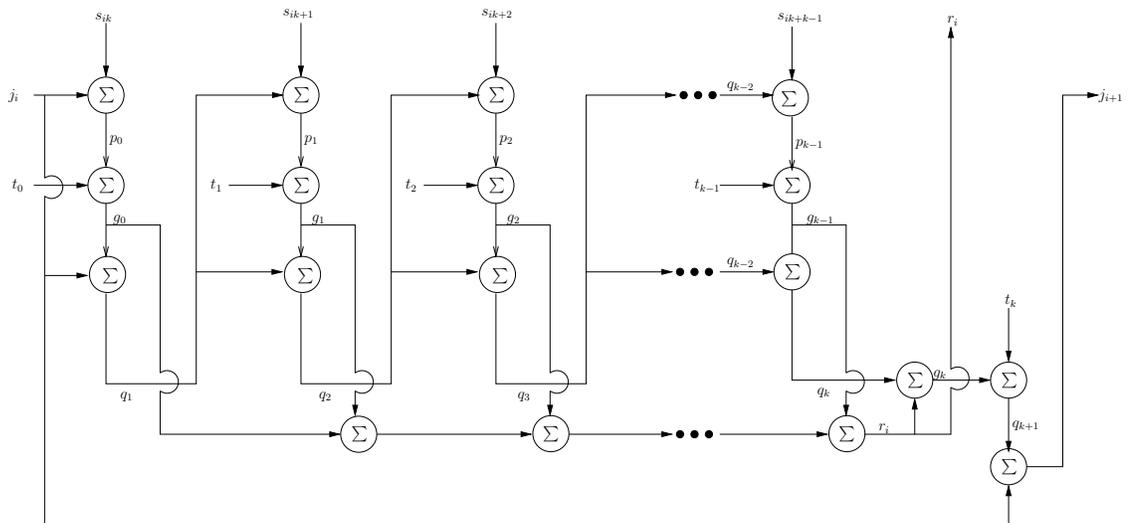
Fig. 2.25: Schematic diagram for rate $\frac{k}{k+1}$ using Partial Parallel Encoding (PPE).

Algorithm 5 Rate k/k+1: Partial Parallel Encoding (PPE)

1: input to the encoder: symbol $S_{ik}, S_{ik+1}, ..., S_{ik+k-1}$, previous state variable j_i

2: $q_0 \leftarrow j_i$

3: **for** $l = 0 : k - 1$ **do**

4:      $s_{ik+l} \leftarrow \psi(S_{ik+l})$

5:      $p_l \leftarrow s_{ik+l} + q_l \ (\text{mod } M)$

6:      determine the target point t_c where $c = l \ (\text{mod } u)$

7:      $g_l \leftarrow p_l + t_c \ (\text{mod } M)$

8:      $q_{l+1} = g_l + q_l \ (\text{mod } M)$

9: **end for**

10: $r_i \leftarrow \sum\limits_{l=0}^{k-1} g_l \ (\text{mod } M)$

11: $R_i \leftarrow \psi^{-1}(r_i)$

12: $g_k \leftarrow r_i + q_k \ (\text{mod } M)$

13: $q_{k+1} \leftarrow g_k + t_k \ (\text{mod } M)$

14: $j_{i+1} \leftarrow q_{k+1} + j_i \ (\text{mod } M)$

15: return parity symbol R_i and state variable j_{i+1}

2.5.3   Decoding a k/(k+1) Rate Code

For codes with a rate higher than $1/2$, the trellis contains parallel branches from state j_i to state j_{i+1} at each i. At every i, k symbols are encoded to generate a parity symbol R_i. Each of the branches of the trellis has k input symbols and one parity symbol. A trellis for SIE rate 2/3 CAC using QPSK constellation is shown in Fig. 2.26. In Fig. 2.26, all of the branches are indicated by black dashed lines and from each of the initial state to each of the final state, there are 4 parallel branches. As this trellis is for rate 2/3 code each branch has two input symbols and one parity symbols and the symbols associated with each branch are labeled as $S_0, S_1/R$, located in the left of each state where S_0 is the first input symbol, S_1 is the second input symbol and R is the parity symbol. In Fig. 2.26, at the left of each initial state there are four horizontal lines which contain all the input and

parity symbols associated with the branches that originated from that state. For each of the collection of such four lines, by using zero-based indexing, the order of the lines represents the branch's final state. For example, among the four lines located on the left of state 0, line 0 represents all of the parallel branches that end at state 0 starting from state 0. In each of those lines, there are four triplets of symbols $(S_0, S_1/R)$, each associated with a branch. The first symbol triplet represents the top branch among all of the parallel branches that started from the same state and ended at the same state, similarly the second symbol triplet represents the second topmost branch and so on. For example from line 0 located on the left of initial state 0, the third symbol triplet $2, 0/0$ (surrounded by an oval box) is associated with the third topmost branch (indicated by solid black arrow) that starts from start 0 and ends at state 0. Now in order to decode, the Viterbi algorithm searches through all of the branches including the parallel branches at each time i and calculates the branch metric μ_t using

$$\mu_t = d_e^2(S_{ik}, \hat{S}_{ik}) + d_e^2(S_{ik+1}, \hat{S}_{ik+1}) + \dots + d_e^2(S_{ik+k-1}, \hat{S}_{ik+k-1}) + d_e^2(R_i, \hat{R}_i) \qquad (2.8)$$

and selects the branch symbols associated with the path with minimum path metric.

$$
\begin{array}{llll}
0,0/0 & 1,2/0 & \boxed{2,0/0} & 3,2/0 \\
0,3/3 & 1,1/3 & 2,3/3 & 3,1/3 \\
0,2/2 & 1,0/2 & 2,2/2 & 3,0/2 \\
0,1/1 & 1,3/1 & 2,1/1 & 3,3/1
\end{array}
$$

$$
\begin{array}{llll}
0,3/2 & 1,1/2 & 2,3/2 & 3,1/2 \\
0,2/1 & 1,0/1 & 2,2/1 & 3,0/1 \\
0,1/0 & 1,3/0 & 2,1/0 & 3,3/0 \\
0,0/3 & 1,2/3 & 2,0/3 & 3,2/3
\end{array}
$$

$$
\begin{array}{llll}
0,2/0 & 1,0/0 & 2,2/0 & 3,0/0 \\
0,1/3 & 1,3/3 & 2,1/3 & 3,3/3 \\
0,0/2 & 1,2/2 & 2,0/2 & 3,2/2 \\
0,3/1 & 1,1/1 & 2,3/1 & 3,1/1
\end{array}
$$

$$
\begin{array}{llll}
0,1/2 & 1,3/2 & 2,1/2 & 3,3/2 \\
0,0/1 & 1,2/1 & 2,0/1 & 3,2/1 \\
0,3/0 & 1,1/0 & 2,3/0 & 3,1/0 \\
0,2/3 & 1,0/3 & 2,2/3 & 3,0/3
\end{array}
$$
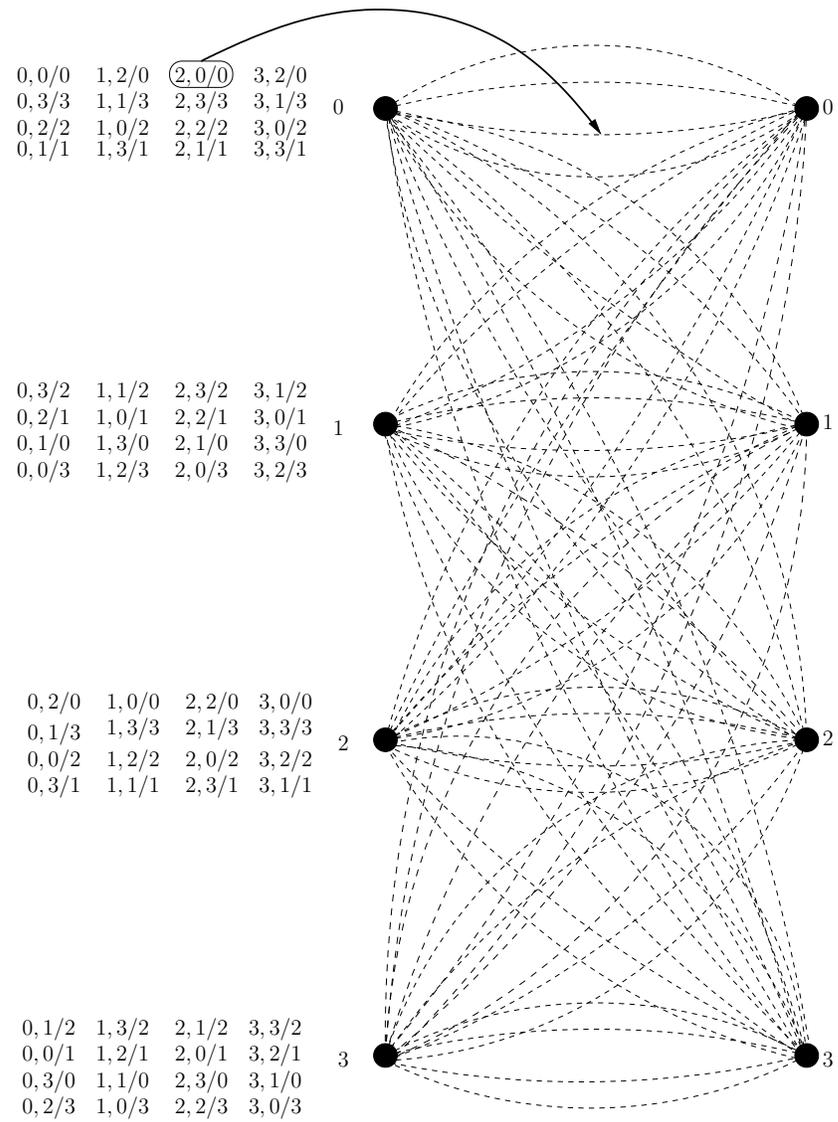
Fig. 2.26: Trellis structure for rate 2/3 CAC (SIE) using QPSK constellation.

CHAPTER 3

SIMULATION AND RESULTS

In this chapter, the required calculations for simulating the Signal Point Target Code and Constellation Arithmetic Code as well as the simulation results for both of the proposed codes are provided.

3.1 Calculation of Energy Per Bit for Simulation

In the simulated performance of an error correction coding system, the horizontal axis of the performance plot is usually the signal-to-noise ratio, E_b/N_0 expressed in dB, where E_b is the energy per transmitted message bit (that is, a bit that actually conveys information, as opposed to merely a bit in a codeword), and N_0 is the noise power spectral density. Since E_b/N_0 is expressed as a ratio, either E_b or N_0 may be fixed, and other quantity determined from the ratio. In simulations presented here, the amplitude A of the signal constellation is fixed. For that signal constellation, E_b is calculated and N_0 is determined from that E_b. Simulated additive white Gaussian noise is produced with the variance

$$\sigma^2 = \frac{N_0}{2}. \tag{3.1}$$

This section describes the relationship between the various quantities.

First, the BPSK modulation, with transmitted symbol amplitude fixed at $\pm A$ is considered. These amplitudes are used to transmit all the bits, that is, the coded bits. Let the energy associated with this transmission be denoted as E_c, the energy per coded bit. Then

$$E_c = A^2 \quad \text{Joules/coded bit.}$$

For a given constellation, E_c is fixed in the simulation. In a coded system, the rate R is the ratio

$$R = \frac{\text{number of message bits}}{\text{number coded bits}}.$$

R is always less than 1.

The energy per message bit, E_b is thus

$$E_b \, \frac{\text{Joules}}{\text{message bits}} = E_c \, \frac{\text{Joules}}{\text{coded bits}} \, \frac{1}{R} \, \frac{\text{number of coded bits}}{\text{number of message bits}}.$$

$$E_b = E_c/R.$$

By looking at from another point of view, to transmit coded bits using the same energy that has been allocated for uncoded message bits, the energy per message bit E_b must be distributed among the coded bits, so

$$E_c = RE_b.$$

In a simulation, given an SNR $\gamma = (E_b/N_0)_{dB}$, solve for N_0 as

$$N_0 = \frac{E_b}{10^{\gamma/10}}. \tag{3.2}$$

from which the noise variance is computed using (3.1). For higher-order modulation with M symbols, let $n = log_2 M$ be the number of bits per symbol. Let $E_{c,s}$ denote the fixed energy per symbol in the constellation (carrying coded symbols). For QAM with the square constellations, where the points in the in-phase axis and quadrature axis are at $\pm A, \pm 3A, ..., \pm(M-1)A$, the energy per coded symbol is

$$E_{c,s} = \frac{2}{3} (M-1)A^2 \, \frac{\text{Joules}}{\text{coded symbol}}.$$

This energy is fixed by the constellation employed. So the energy per uncoded message symbol $E_{s,m}$ is

$$E_{s,m} = E_{c,s}/R.$$

Since each symbol carries n bits of information, the energy per bit is

$$E_b = E_{s,m} \frac{\text{Joules}}{\text{uncoded message symbol}} \frac{1 \text{ symbol}}{n \text{ bits}}$$
$$= \frac{E_{s,m}}{n} \frac{\text{Joules}}{\text{message bits}}$$
$$= \frac{E_{c,s}}{nR}.$$

Using the energy per bit for coded symbol E_b, σ^2 and N_0 are calculated from (3.1) and (3.2) for a given SNR γ. For $M = 16$, $n = 4$, $R = \frac{1}{2}$, let $A = 1$. So $E_b = \frac{2\times15}{3\times4\times.5} = 5$. Using E_b, σ is calculated for SNR γ and then $N \sim \mathcal{N}(0, \sigma^2)$ is generated where N is the noise added with the transmitted signal at that SNR γ.

The received signal is

$$\hat{S}_i = S_i + N$$

$$\hat{R}_i = R_i + N,$$

where R_i =transmitted parity symbol, S_i =transmitted message symbol. Received symbols $\hat{S}_i$ and $\hat{R}_i$ are used as input of Viterbi decoder to get decoded message symbol $\tilde{S}_i$. If $\tilde{S}_i \neq S_i$ then an error has occurred. Let $z_{\gamma,i}$ is a Bernoulli random variable with probability p_γ, where p_γ is the probability of symbol error at SNR γ. So

$$z_{\gamma,i} = \begin{cases} 1 & \text{if an error occurs in the decoder} \\ 0 & \text{if there is no error.} \end{cases}$$

Let e_γ be the total number of errors occurred for SNR γ and $N_{s,\gamma}$ be the total number of symbols sent through the channel at that SNR γ. As so $e_\gamma = \sum_{i=1}^{N_{s,\gamma}} z_{\gamma,i}$, e_γ is binomially distributed $\sim B(N_{s,\gamma}, p_\gamma)$ with $\mathrm{E}[e_\gamma] = N_{s,\gamma}p_\gamma$ and $\mathrm{var}(e_\gamma) = N_{s,\gamma}p_\gamma(1-p_\gamma)$. The likelihood

function for e_γ

$$f_{e_\gamma}(e_\gamma|p_\gamma) = \binom{N_{s,\gamma}}{e_\gamma} p_\gamma^{e_\gamma} (1 - p_\gamma)^{(N_{s,\gamma} - e_\gamma)}.$$

The log likelihood function is

$$\begin{aligned}
\Lambda(e_\gamma, p_\gamma) &= \log f_{e_\gamma}(e_\gamma|p_\gamma) \\
&= \log \binom{N_{s,\gamma}}{e_\gamma} + e_\gamma \log p_\gamma + (N_{s,\gamma} - e_\gamma) \log(1 - p_\gamma).
\end{aligned}$$

Now by taking gradient with respect to p_γ and equating to zero, maximum likelihood estimation of the probability of symbol error $\hat{p}_\gamma$ at SNR γ is obtained.

$$\frac{\partial \Lambda(e_\gamma, p_\gamma)}{\partial p_\gamma} = 0$$

$$\frac{e_\gamma}{p_\gamma} - \frac{(N_{s,\gamma} - e_\gamma)}{1 - p_\gamma} = 0$$

$$\hat{p}_\gamma = \frac{e_\gamma}{N_{s,\gamma}} \tag{3.3}$$

The variance of $\hat{p}_\gamma$ is

$$\begin{aligned}
\text{var}(\hat{p}_\gamma) &= \text{var}(\frac{e_\gamma}{N_{s,\gamma}}) \\
&= \frac{1}{N_{s,\gamma}^2} N_{s,\gamma} p_\gamma (1 - p_\gamma) \\
&= \frac{p_\gamma(1 - p_\gamma)}{N_{s,\gamma}} \\
&= \frac{\hat{p}_\gamma(1 - \hat{p}_\gamma)}{N_{s,\gamma}} \qquad \text{using the estimated } \hat{p}_\gamma \\
&= \frac{\frac{e_\gamma}{N_{s,\gamma}}(1 - \frac{e_\gamma}{N_{s,\gamma}})}{N_{s,\gamma}} \\
&= \frac{e_\gamma N_{s,\gamma} - e_\gamma^2}{N_{s,\gamma}^3}.
\end{aligned}$$

So the standard deviation is

$$\begin{aligned} \sigma_{\hat{p}_\gamma} &= \sqrt{\mathrm{var}(\hat{p}_\gamma)} \\ &= \sqrt{\frac{e_\gamma N_{s,\gamma} - e_\gamma^2}{N_{s,\gamma}^3}}. \end{aligned} \tag{3.4}$$

For simulation, $N_{s,\gamma}$ is counted until $e_\gamma = 100$. Using $N_{s,\gamma}$ and $e_\gamma = 100$ probability of symbol error is calculated from (3.3). The bit error rate of coded symbols is calculated by taking the ratio of the total number of erroneous bits and the total number of transmitted bits. The total number of transmitted bits is calculated from $N_{s,\gamma}$ and the total number of erroneous bits is calculated by counting how many bits vary between $\tilde{S}_i$ and S_i when $\tilde{S}_i \neq S_i$. In this thesis, the level of confidence for calculating $\hat{p}_\gamma$ is considered as inversely proportional to standard deviation of $\hat{p}_\gamma$. The smaller the $\hat{p}_\gamma$ is, the higher the level of confidence is. For $e_\gamma = 100$, at each SNR γ, the standard deviation of $\hat{p}_\gamma$ is calculated from (3.4). In Figs. 3.1 and 3.3, the error bars represent one standard deviation above and below from the estimated probability of symbol errors. As the error bars for each $\hat{p}_\gamma$ is very small, the confidence level is high. So the error bars are not presented in the rest of the BER and SER plots for this thesis.

In this thesis, the performances of codes are evaluated as BER and SER for constellations of different sizes. Gray code indexing, as well as other bit assignments in the signal constellation, have been used to calculate d_{free}^2 and simulate their coding gain. As there is no theoretical bound for bit error rate if the bits are not assigned using gray code indexing so to evaluate BER of proposed encoding schemes, first theoretical SER for uncoded symbols are calculated using (3.5) and (3.6). The plots of those theoretical SERs have been labeled by "th" with their corresponding signal constellation name. For example, in Fig. 3.1 the blue dotted line labeled by "QPSK(th) SER" represents the theoretical symbol error rate of QPSK constellation. The simulated SER and SER are calculated by sending uncoded symbols over an AWGN channel with SNR γ and by counting the total number of erroneous bits and symbols after decoding the received symbols. The plots of such SERs and BERs, obtained from simulations are labeled by "ex" with their corresponding signal constellation name. For example, in Fig. 3.1, the red solid line labeled by "QPSK(ex) SER" represents

the SER of uncoded symbols using QPSK constellation and the red dotted line labeled by "QPSK(ex) BER" represents BER of uncoded symbols using QPSK constellation. In these labels, "(th)" represents theoretical result and "(ex)" represents experimental results. After calculating SER for uncoded symbols from simulations, the results have been compared with theoretical SER for uncoded symbols of the same constellation size to make sure the simulations are working properly. If the simulated SER overlays with theoretical SER then the BER obtained from that simulation has been used to compare with the BER of coded symbols. The theoretical BER for BPSK [9] is calculated using

$$P_e = Q\left(\sqrt{\frac{2E_b}{N_0}}\right),\tag{3.5}$$

where $Q(.)$ is standard Q function defined in A. The theoretical SER [10] for constellation larger than BPSK is calculated using

$$P_e = 1 - \left(1 - 2\left(1 - \frac{1}{\sqrt{M}}\right)Q\left(\sqrt{\frac{3nE_b}{(M-1)N_0}}\right)\right)^2.\tag{3.6}$$

3.2 Performance of SPTC and CAC for Rate $1/2$ Codes

In this section, the simulation results for both rate 1/2 SPTC and rate 1/2 CAC are presented. For SPTC different shapes and constellations are used to generate BER and SER while for CAC different constellations and bit assignments are used. The performances of both codes are evaluated using BER and SER obtained from simulations.

3.2.1 Performance of SPTC

From Fig. 3.1, using the shape shown in Fig. 2.6 and QPSK constellation with gray code indexing provides approximately 2.25 dB coding gain for both SER and BER compared to uncoded QPSK at $P_s = 10^{-5}$. The shape shown in Fig. 2.6, has $d_{\text{free}}^2 = 16E_{c,s}$.
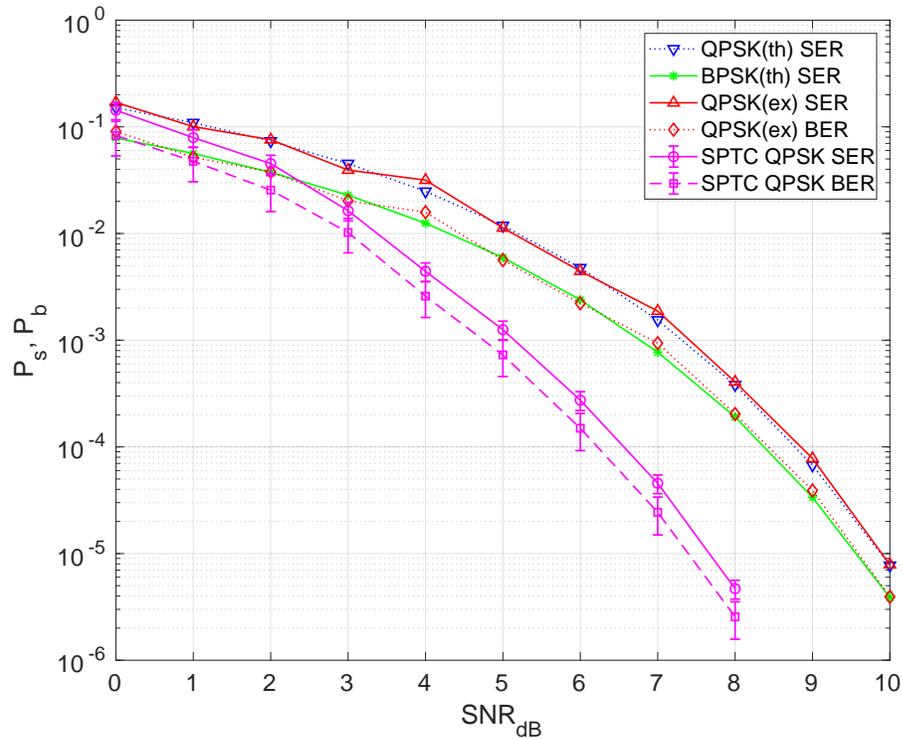
Fig. 3.1: BER and SER Curve for rate $\frac{1}{2}$ SPTC with target point $t = 0$ on a QPSK constellation for the shape shown in Fig. 2.6.

The performance of SPTC using the shape from Fig. 2.10 which has $d_{\text{free}}^2 = 28E_{c,s}$ and QPSK constellation with gray code indexing, is presented in Fig. 3.2. As this shape has a larger d_{free}^2 than the shape shown in Fig. 2.6, at $P_s = 10^{-5}$ it provides a coding gain of 3 dB compared to uncoded QPSK and a coding gain of 0.75 dB compared to the performance of the shape shown in Fig. 2.6.
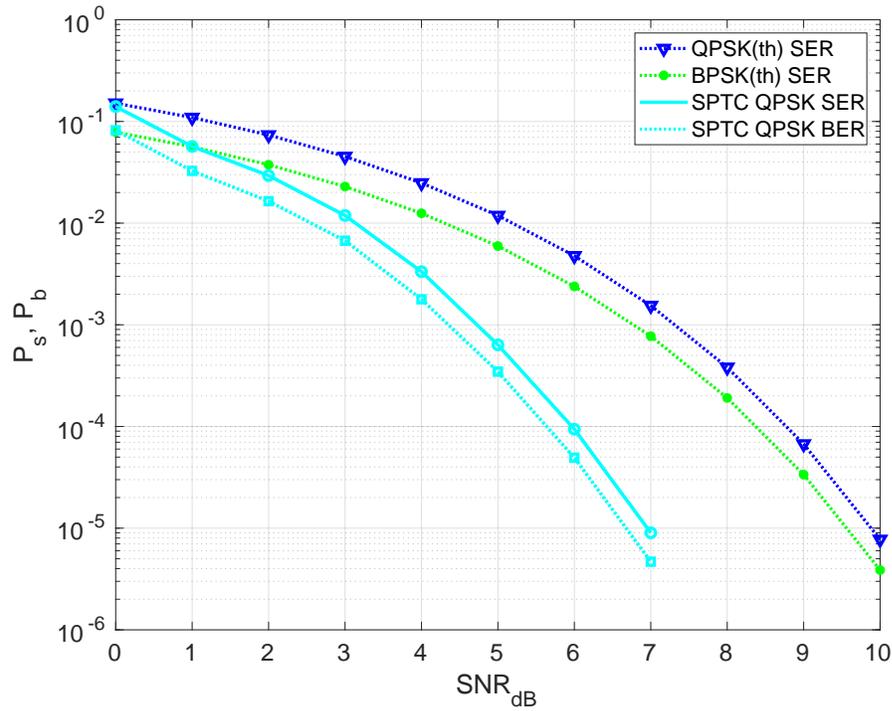
Fig. 3.2: BER and SER Curve for rate $\frac{1}{2}$ SPTC with target point $t = 0$ on a QPSK constellation for shape shown in Fig. 2.10.

To simulate the performance of SPTC on 16QAM constellation with gray code indexing, the shape shown in Fig. 2.1 has been used and the performance is presented in Fig. 3.3. The shape used for this simulation has $d_{\text{free}}^2 = 16E_{c,s}$. At $P_s = 10^{-4}$, this code has a coding gain of 2.75 dB compared to the performance of uncoded 16QAM constellation.
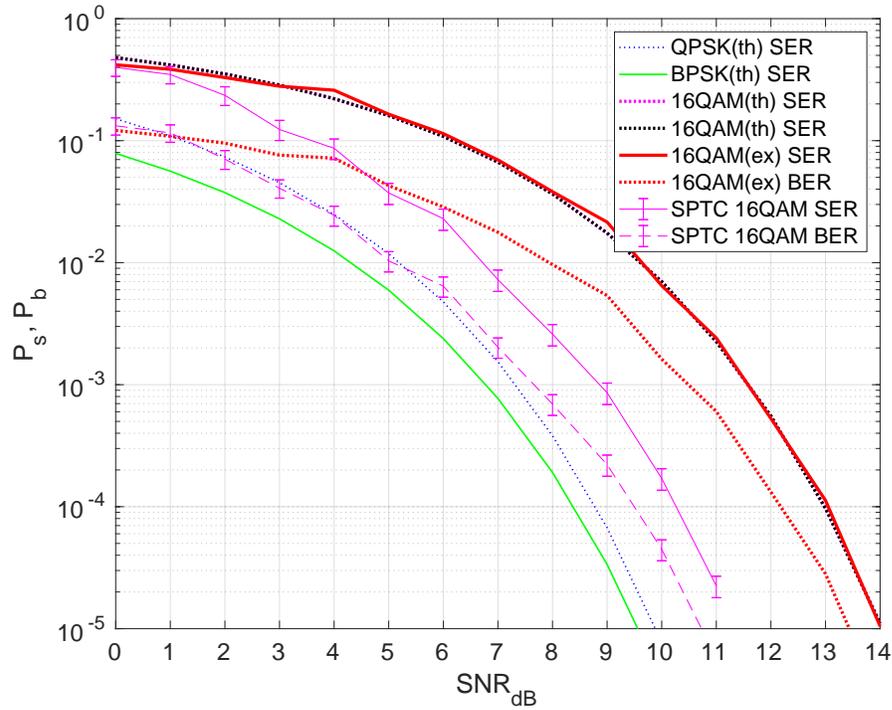
Fig. 3.3: BER and SER Curve for rate $\frac{1}{2}$ SPTC with target point $t = 0$ on a 16QAM constellation for shape shown in Fig. 2.1.

Fig. 3.4 shows the performance of SPTC using 64QAM constellation with gray code indexing, constant target point $t = 0$ and the shape shown in Fig. 2.3. From the BER and SER curves, it can be seen that using this shape a coding gain of 2.5 dB compared to the performance of uncoded 64QAM can be achieved.
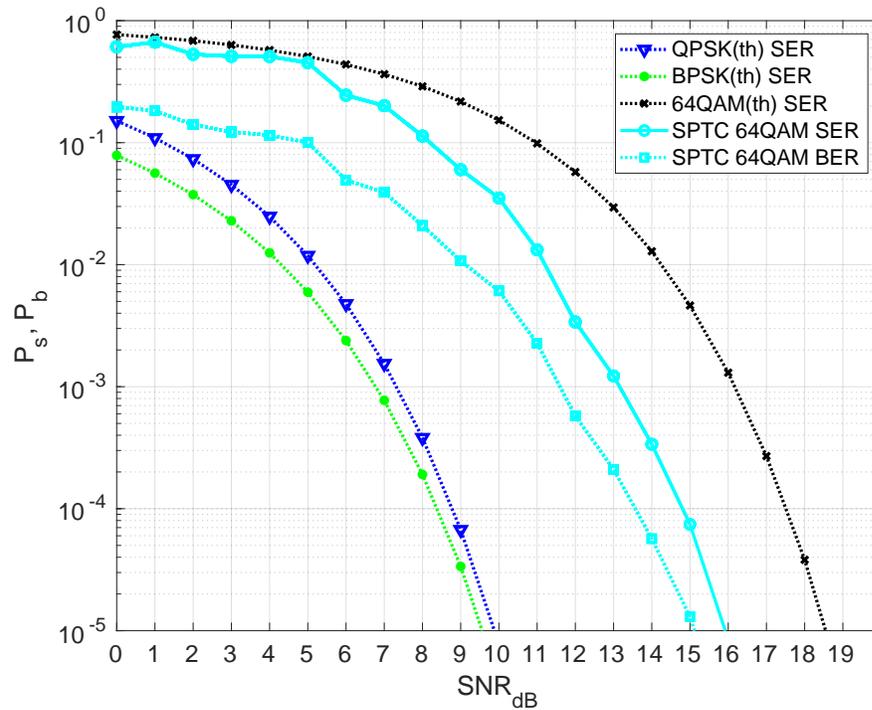
Fig. 3.4: BER and SER Curve for rate $\frac{1}{2}$ SPTC with target point $t = 0$ on a 64QAM constellation for shape shown in Fig. 2.3.

3.2.2   Performance of CAC

The performance of CAC using QPSK constellation and bit assignment shown in Fig. 2.6. This bit assignment results in similar performance with SPTC using QPSK constellation and shape shown in Fig. 2.6. For this bit assignment, the free Euclidean distance is $12E_{c,s}$.
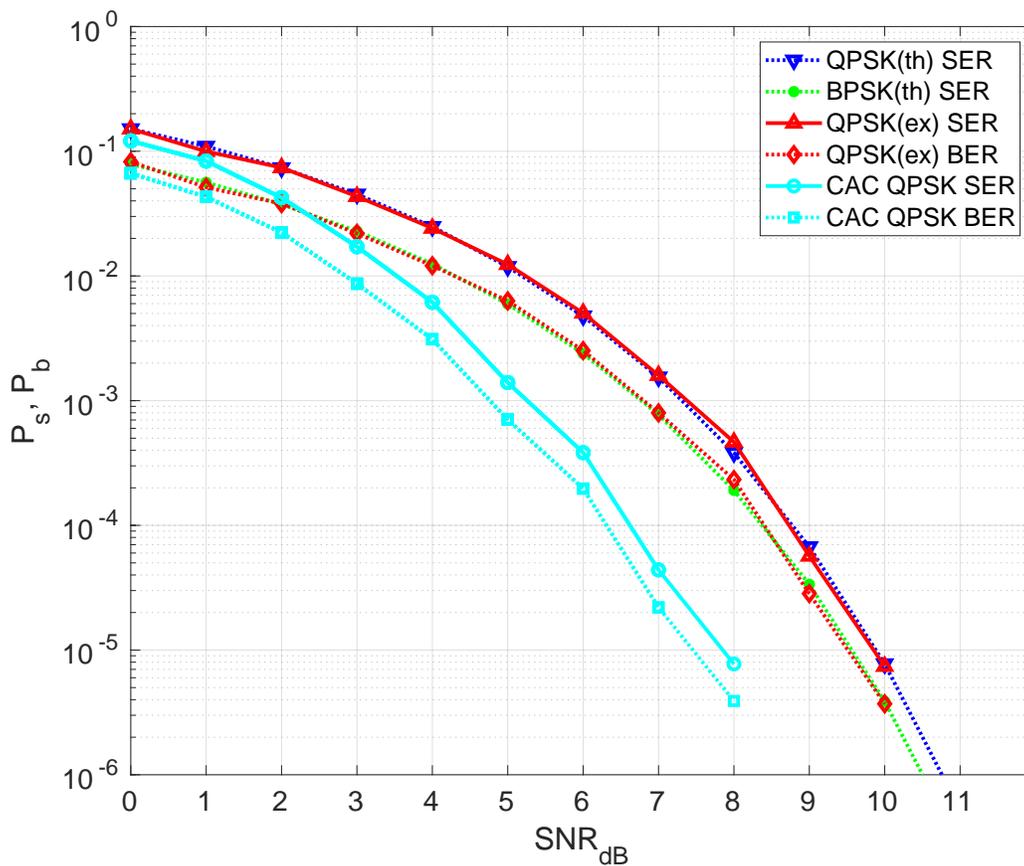
Fig. 3.5: BER and SER of rate $\frac{1}{2}$ CAC using QPSK signal constellation and bit assignment shown in Fig. 2.6.

Fig. 3.6 shows that it is possible to increase the coding gain for CAC in QPSK constellation by increasing the d_{free}^2 to $20E_{c,s}$ results from the bit assignment shown in Fig. 2.16. The bit assignment from Fig. 2.16 provides 3.5 dB coding gain compared to the performance of uncoded QPSK.
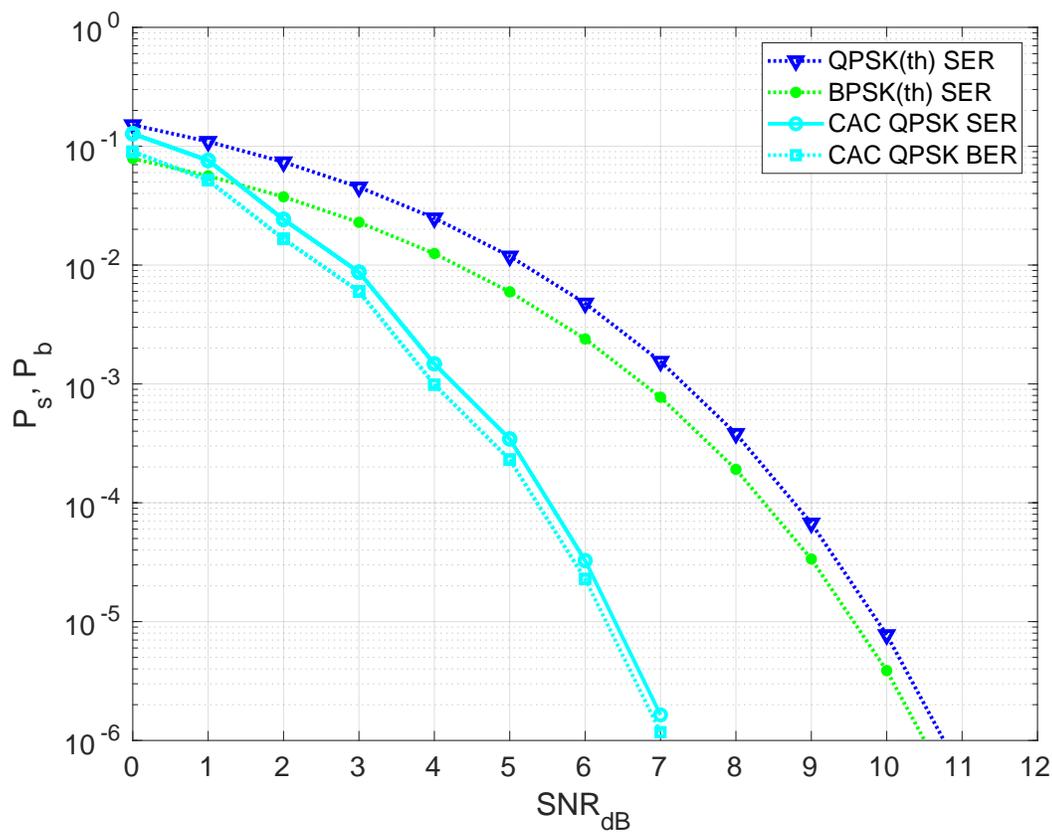
Fig. 3.6: BER and SER of rate $\frac{1}{2}$ CAC using QPSK signal constellation and bit assignment shown in Fig. 2.16.

To evaluate the performance of CAC on 16QAM constellation for d_{free}^2, different bit assignments have been used. Using gray code indexing shown in Fig. 2.1 the simulation results for rate 1/2 CAC on 16QAM constellation is presented in Fig. 3.7. Gray code indexing results in $d_{\text{free}}^2 = 16E_{c,s}$ and a coding gain of 3 dB compared to the performance of uncoded 16QAM at $P_s = 10^{-5}$.
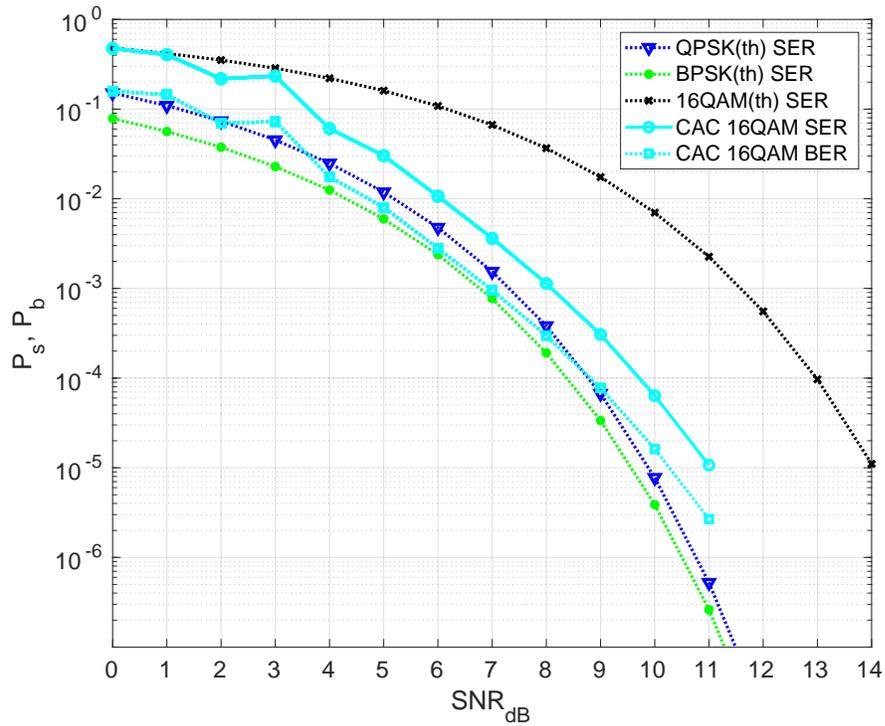
Fig. 3.7: BER and SER for rate 1/2 CAC using 16QAM constellation and bit assignment shown in Fig. 2.1.

The bit assignment shown in Fig. 2.19, has $d^2_{\text{free}} = 20E_{c,s}$ and the performance of CAC using this bit assignment is provided in Fig. 3.8. This bit assignment results in a coding gain of 4 dB at $P_s = 10^{-5}$ compared to uncoded 16QAM BER and SER.
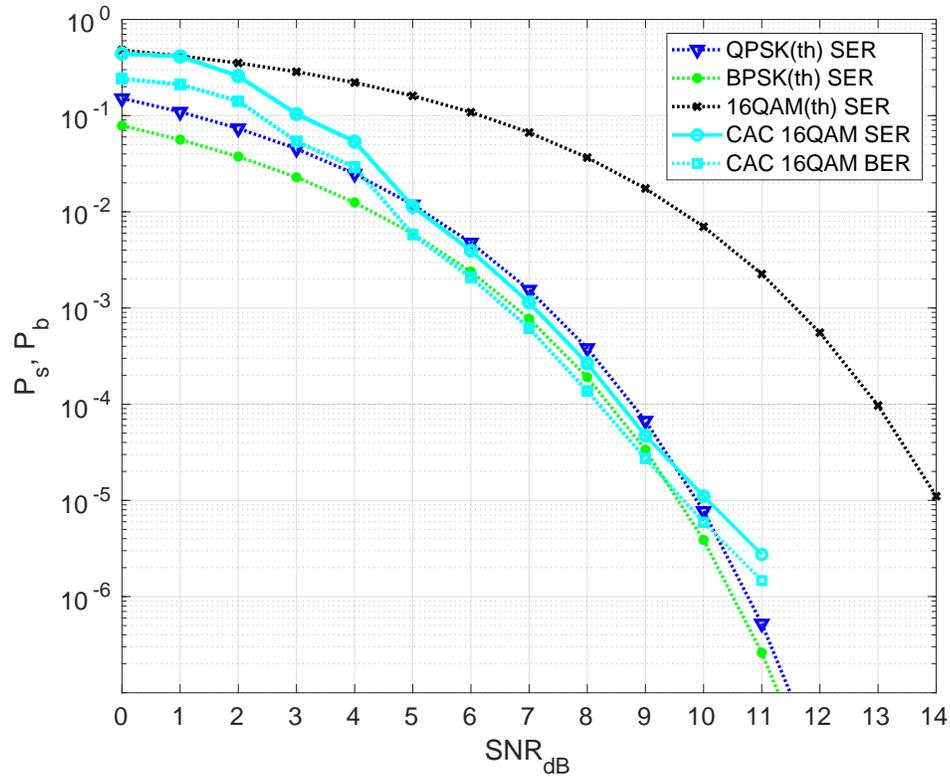
Fig. 3.8: Ber and SER of rate $\frac{1}{2}$ CAC using 16QAM constellation and bit assignment shown in Fig. 2.19.

Using the Sequential Bit Assignment shown in Fig. 2.20, the performance of the rate 1/2 CAC on 16QAM constellation is presented in Fig. 3.9. For this bit assignment with $d_{\text{free}}^2 = 28E_{c,s}$, at $P_s = 10^{-5}$, 5 dB coding gain is possible compared to uncoded performance of 16QAM signal constellation. This simulation also shows that rate 1/2 CAC on 16QAM constellation using this bit assignment provides 1 dB coding gain for SER at $Ps = 10^{-5}$ compared to the SER of uncoded QPSK. The uncoded QPSK has the same information per symbol as the rate 1/2 code using 16QAM constellation.
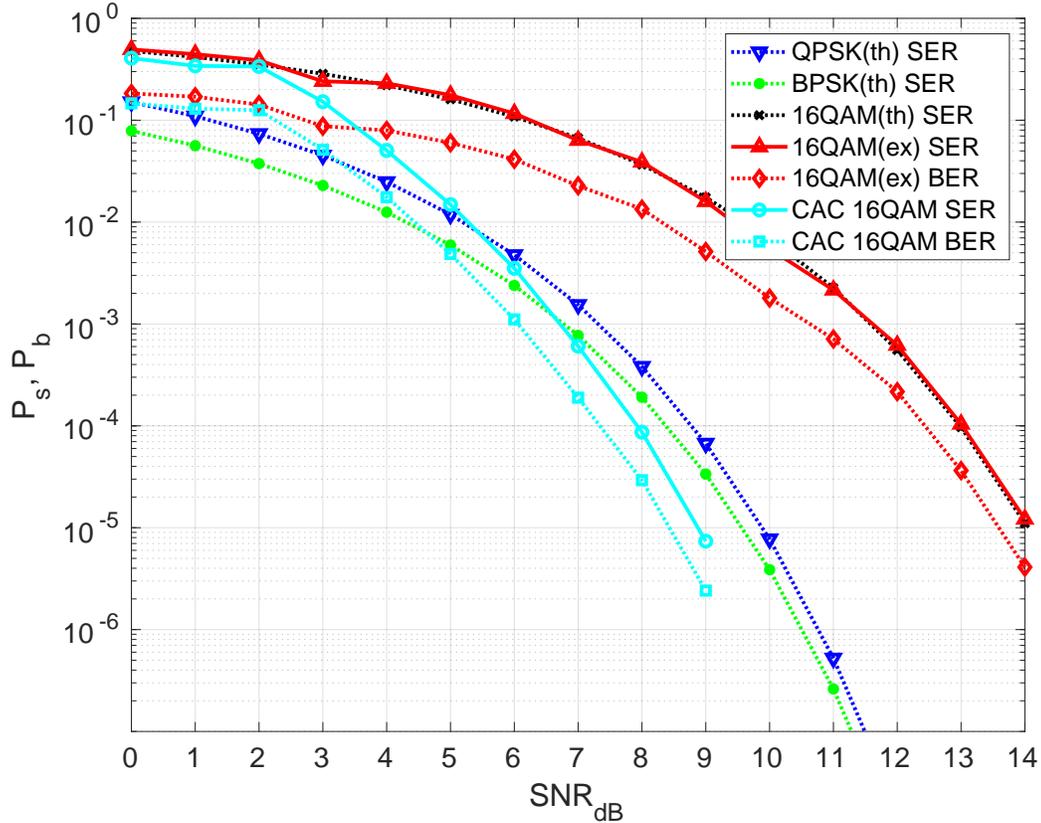
Fig. 3.9: Ber and SER of rate $\frac{1}{2}$ CAC using 16QAM constellation and bit assignment shown in Fig. 2.20.

For 64QAM, the BER and SER for rate 1/2 CAC using Sequential Bit Assignment shown in Fig. 2.23 are provided in Fig. 3.10. From Fig. 3.10 it can be seen that the Sequential Bit Assignment provides a 5.5 dB coding gain at $P_s = 10^{-5}$ compared to the BER and SER of uncoded 64QAM.
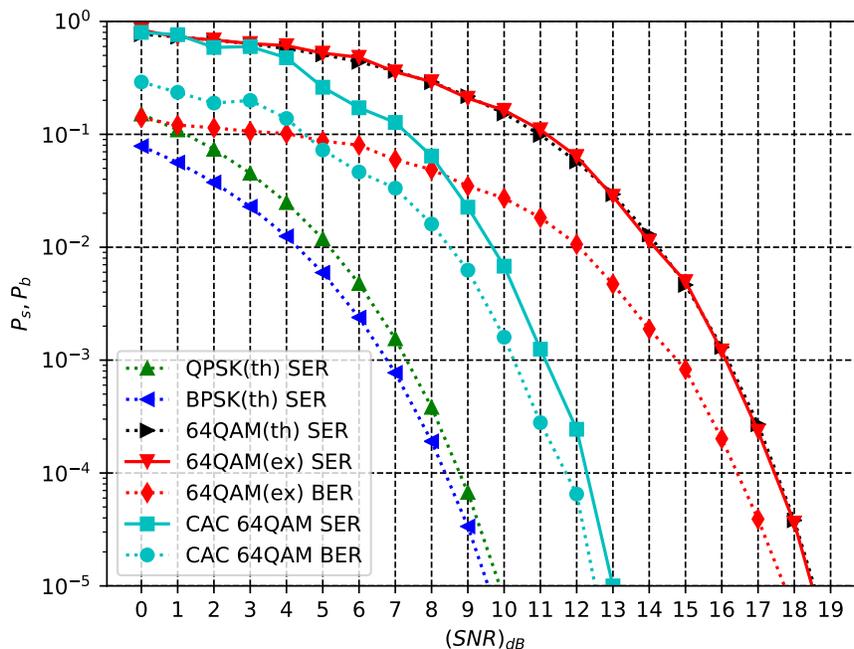
Fig. 3.10: Ber and SER of rate $\frac{1}{2}$ CAC using 64QAM constellation and bit assignment shown in Fig. 2.23.

Simulation results for rate 1/2 CAC using 256QAM signal constellation are shown in Fig. 3.11. For this simulation, Sequential Bit Assignment has been used in the signal constellation. The rate 1/2 CAC provides 5 dB coding gain for both BER and SER compared to uncoded BER and SER for 256QAM.
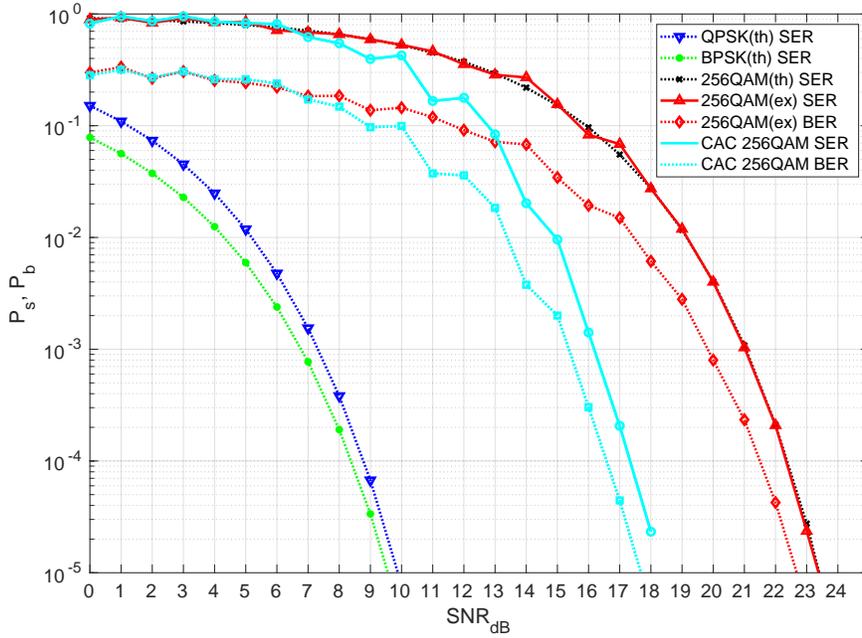
Fig. 3.11: Ber and SER of rate $\frac{1}{2}$ CAC using 256QAM constellation and Sequential Bit Assignment.

3.2.3 Comparison of Performance Between CAC and SPTC

Though the performance of SPTC depends on the employed shape and the performance of CAC depends on bit assignment in the signal constellation, in this part of the thesis the performance of SPTC and CAC is compared using free Euclidean distance. In Figs. 3.12 and 3.13, comparison between SPTC and CAC on QPSK constellation is shown. In Fig. 3.12, the performance of SPTC is calculated using the shape shown in Fig. 2.6. For SPTC and CAC, gray code indexing has been used. While for this shape the SPTC has $d_{\text{free}}^2 = 16E_{c,s}$ and for this bit assignment the CAC has $d_{\text{free}}^2 = 12E_{c,s}$, they provided approximately similar

performance. In Fig. 3.13, the shape shown in Fig. 2.10 has been used for SPTC with gray code indexing and for CAC, the bit assignment shown in Fig. 2.16 has been used. Though the shape provides $d_{\text{free}}^2 = 28E_{c,s}$ and the bit assignment provides $d_{\text{free}}^2 = 20E_{c,s}$, CAC has a 0.5 dB coding gain compared to SPTC at $P_s = 10^{-5}$.
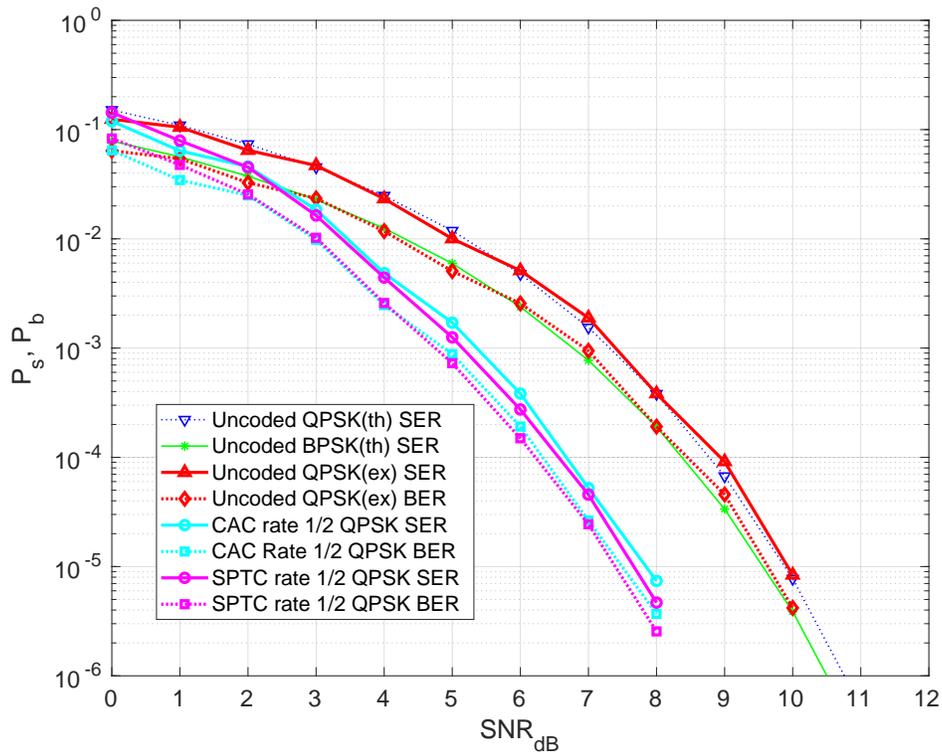
Fig. 3.12: Comparison between rate $\frac{1}{2}$ Constellation Arithmetic Code and SPTC on a QPSK constellation.
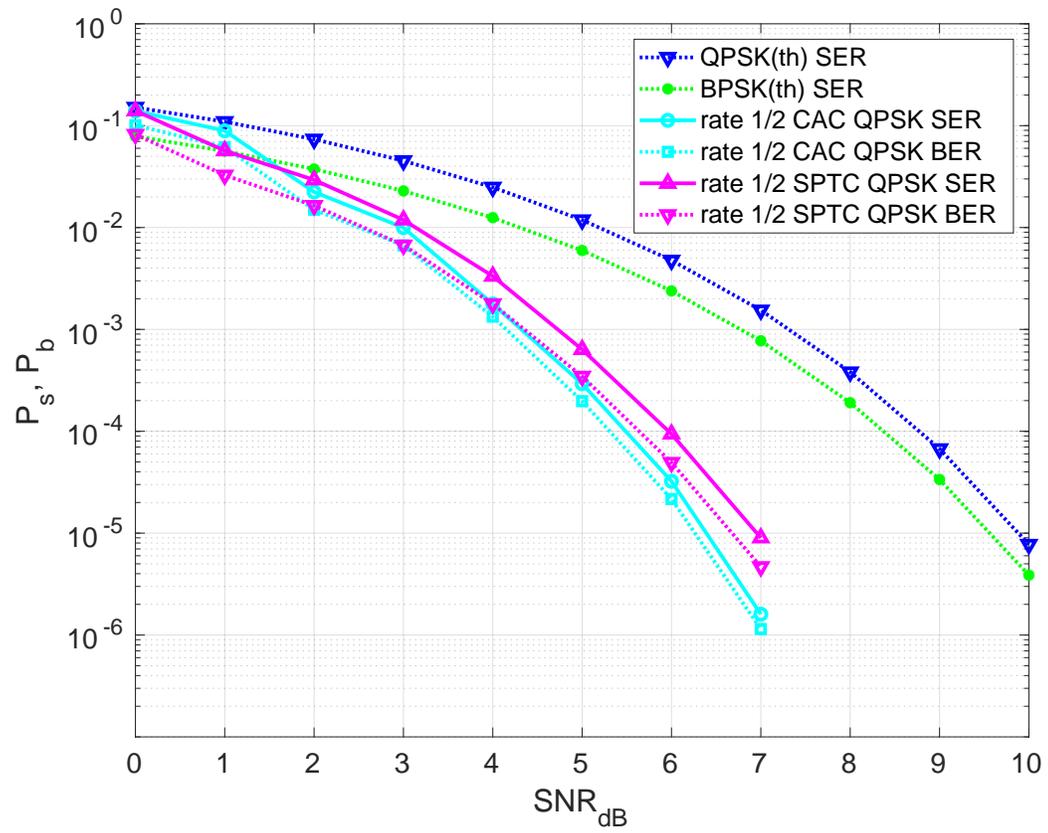
Fig. 3.13: Comparison of performance between rate $\frac{1}{2}$ Constellation Arithmetic Code and SPTC on a QPSK constellation.

Fig. 3.14 shows the comparison of performance between SPTC and CAC on 16QAM constellation where CAC has 2 dB coding gain at $Ps = 10^{-4}$. The shape from Fig. 2.1 used for encoding SPTC has $d_{\text{free}}^2 = 16E_{c,s}$ while the Sequential Bit Assignment used in CAC has $d_{\text{free}}^2 = 28E_{c,s}$. The comparison between SPTC and CAC using 64QAM constellation is shown in Fig. 3.15. The SPTC has been simulated using the shape shown in Fig. 2.3 using gray code indexing and the CAC has been simulated using the bit assignment shown in Fig. 2.23. From Fig. 3.15, CAC has 2.5 dB coding gain compared to SPTC at $P_s = 10^{-4}$.
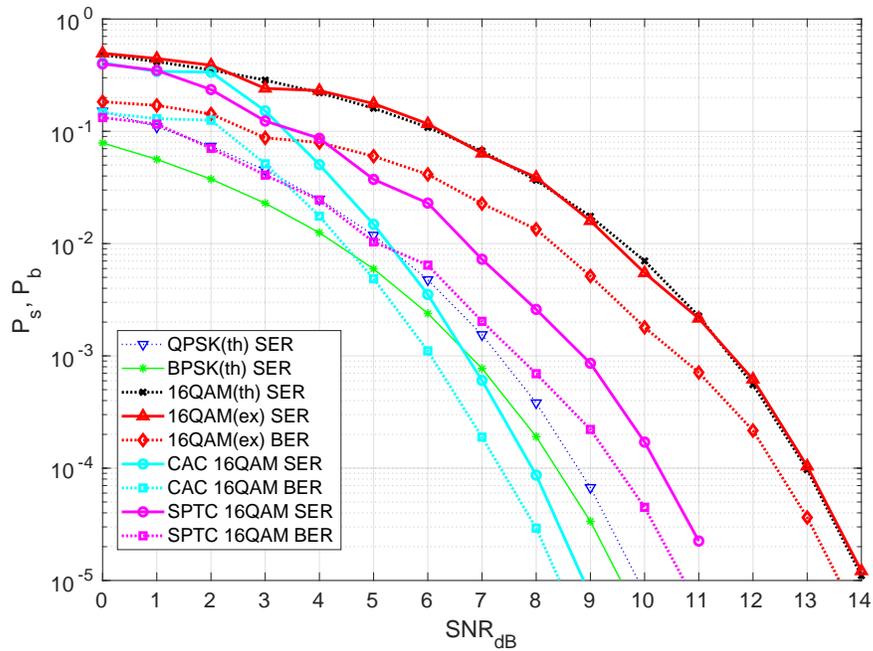
Fig. 3.14: Comparison of performance between rate $\frac{1}{2}$ Constellation Arithmetic Code and SPTC on a 16QAM constellation.
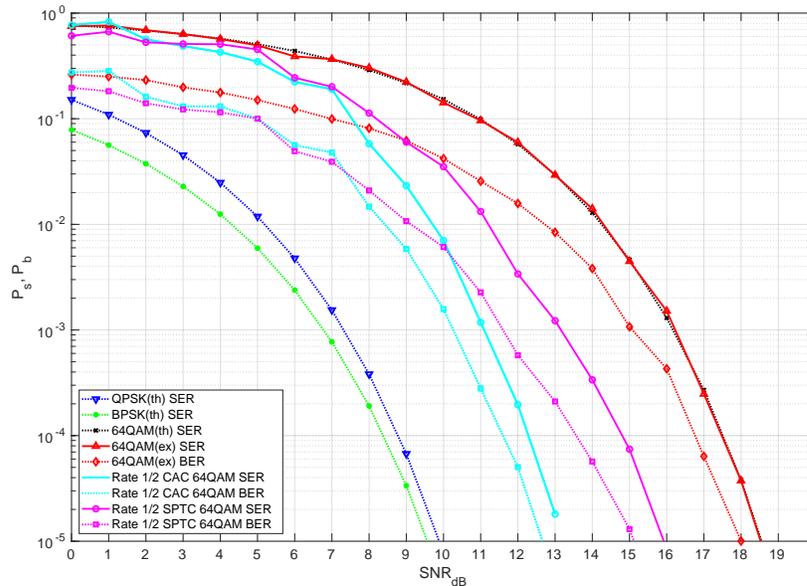
Fig. 3.15: Comparison of performance between rate $\frac{1}{2}$ Constellation Arithmetic Code and SPTC on a 64QAM constellation.

3.3 Performance of Codes for Different Trellis Depth

Fig. 3.16 presents the performance of the Viterbi decoder for different trellis depth using QPSK signal constellation with rate 1/2 CAC and bit assignment shown in Fig. 2.16. From this figure, it can be seen that by increasing the trellis depth the performance of the decoder can be increased. However, for larger trellis depth no noticeable coding gain is achieved. The performance of the decoder remains same for trellis depth of 5, 10 and 20. Therefore trellis depth of 10 has been used to decode the received symbols using the Viterbi algorithm.
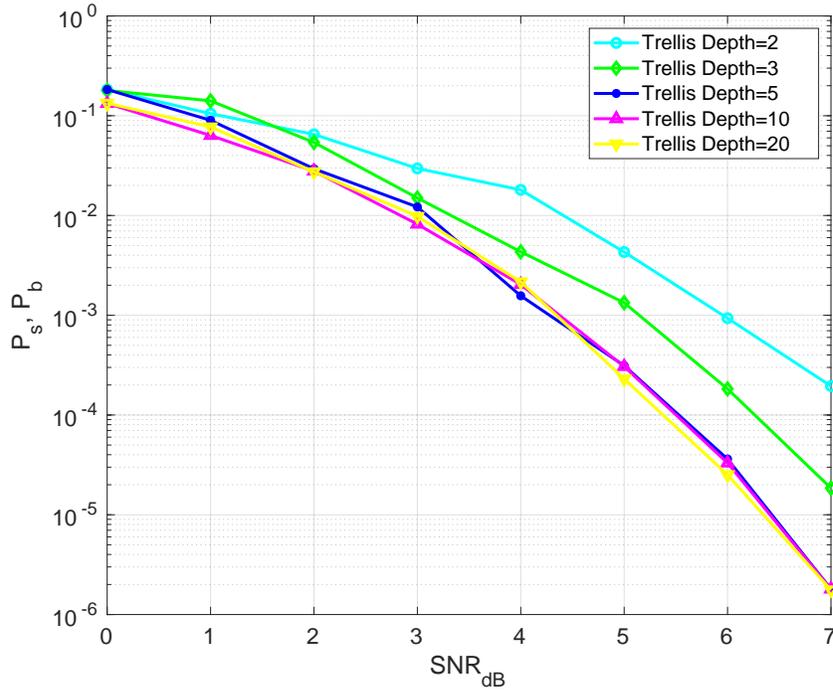
Fig. 3.16: Comparison of performance for rate $\frac{1}{2}$ CAC using QPSK signal constellation and bit assignment shown in Fig. 2.16 with various trellis depths.

3.4   Performance of CAC for Rate $2/3$ and Rate $3/4$ Codes

In this section, the simulated performances of CAC for rate 2/3 and rate 3/4 have been provided. In QPSK constellation, gray code indexing has been used and for 16QAM and 64QAM constellation bit assignments shown in Figs. 2.20 and 2.23 have been used. Results for both SIE and PPE are provided below. For evaluating the performance of the proposed methods, comparisons have been made using BER and SER between the codes for a given constellation and uncoded modulation for that constellation.

3.4.1   Simulation Results for Rate $2/3$ and Rate $3/4$ Codes Using SIE

Simulation results for CAC using QPSK constellation for rate 2/3 and rate 3/4 are provided in Figs. 3.17 and 3.18. From these results it can be seen that this method provides 1.5 dB coding gain $P_s = 10^{-4}$ compared to uncoded QPSK for both 2/3 and 3/4

rate codes on QPSK constellation. The BER and SER curves for rate 2/3 and rate 3/4 codes using 16QAM constellation are shown in Figs. 3.19 and 3.20. In Figs. 3.19 and 3.20, SIE has a 1.5 dB coding gain at $P_s = 10^{-5}$ for both rate 2/3 and rate 3/4 codes from uncoded 16QAM. For SIE, the BER and SER curves for rate 2/3 CAC using 64QAM is shown in Fig. 3.21 and from this plot it can be seen that there is a 1 dB coding gain at $P_s = 10^{-4}$.
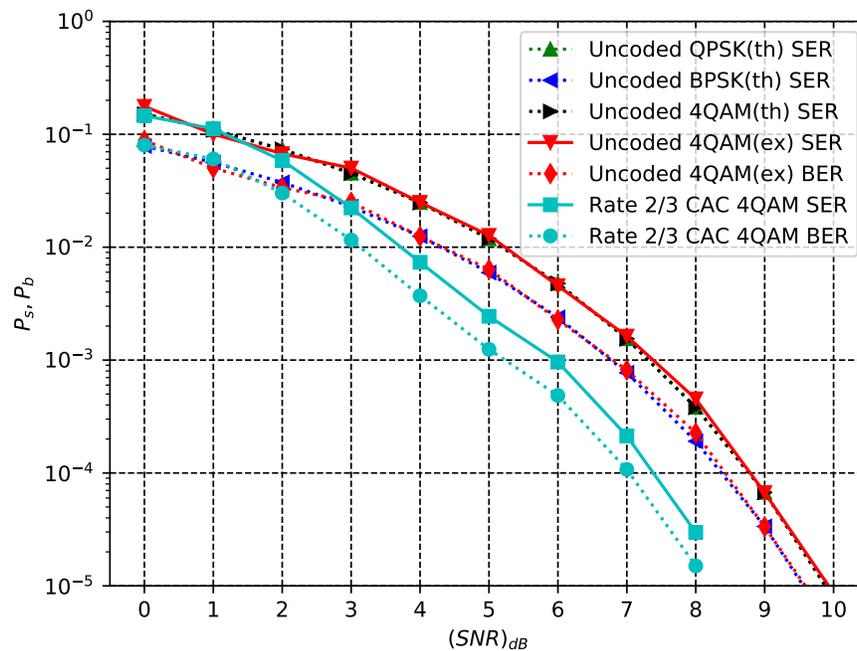
Fig. 3.17: BER and SER curve of rate $\frac{2}{3}$ CAC (SIE) on a QPSK constellation.
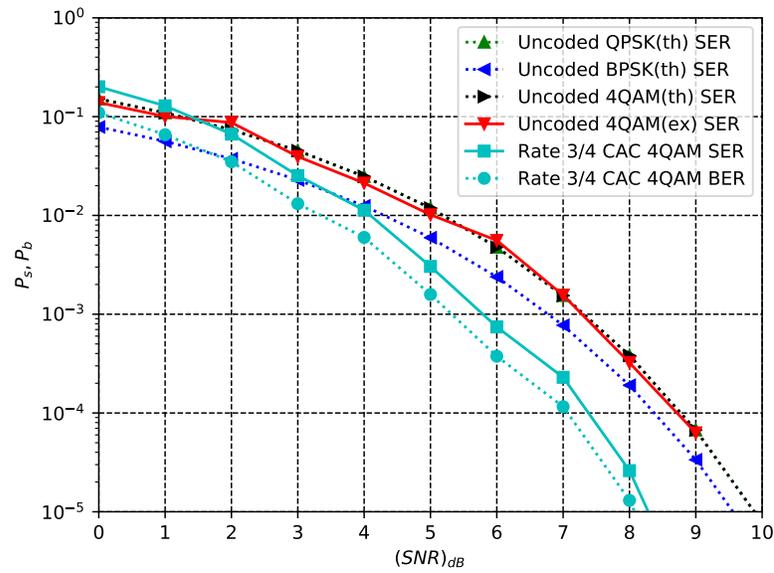
Fig. 3.18: BER and SER curve of rate $\frac{3}{4}$ CAC (SIE) on a QPSK constellation.
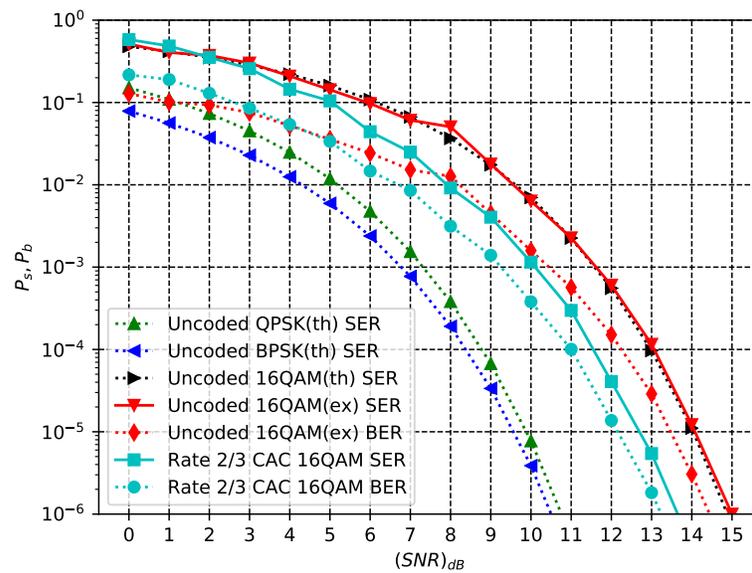
Fig. 3.19: BER and SER curve of rate $\frac{2}{3}$ CAC (SIE) on a 16QAM constellation.
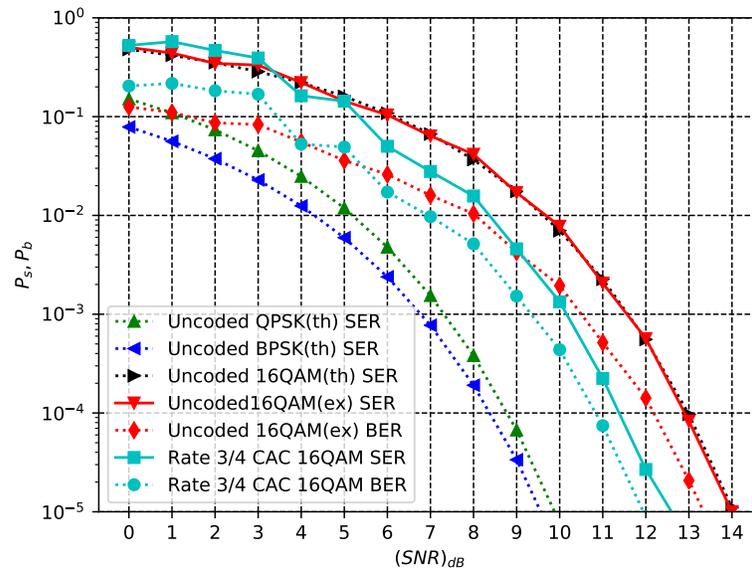
Fig. 3.20: BER and SER curve of rate $\frac{3}{4}$ CAC (SIE) on a 16QAM constellation.
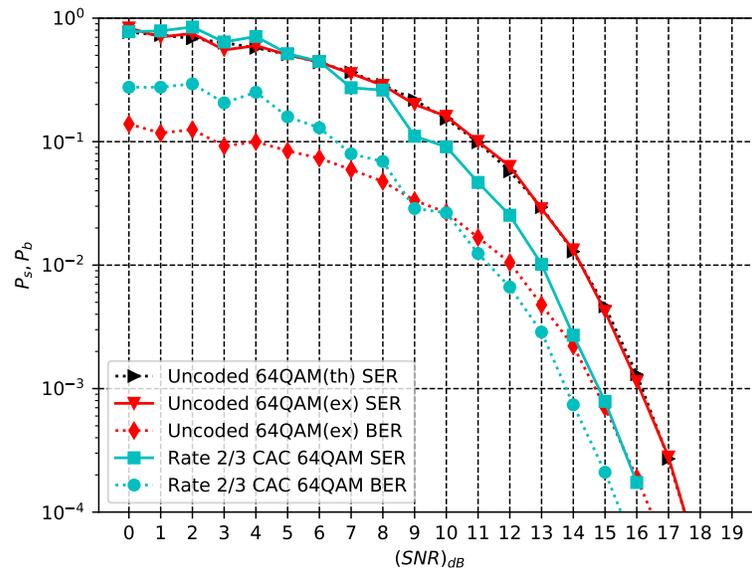
Fig. 3.21: BER and SER curve of rate $\frac{2}{3}$ CAC (SIE) on a 64QAM constellation.

3.4.2 Simulation Results for Rate $2/3$ and Rate $3/4$ Codes Using PPE

The BER and SER curves of rate 2/3 and rate 3/4 codes encoded using PPE and QPSK, 16QAM, and 64QAM constellations are provided in Figs. 3.22, 3.23, 3.24, 3.25 and 3.26. From Figs. 3.22, 3.23 and 3.25 it can be seen that this method fails to provide any coding gain for QPSK constellation and rate 3/4 codes for 16QAM and 64QAM constellation. However, from the results of Figs. 3.24 and 3.26, PPE has a good coding gain for rate 2/3 code on 16QAM and 64QAM constellation compared to the coding gain of SIE rate 2/3 codes for these constellations. In Fig. 3.24, PPE has a 4 dB coding gain at $P_s = 10^{-5}$ compared to the BER and SER of the uncoded 16QAM constellation and in 3.26, PPE has 5.5 dB coding gain at $P_s = 10^{-5}$ compared to the performance of the uncoded 64QAM constellation.
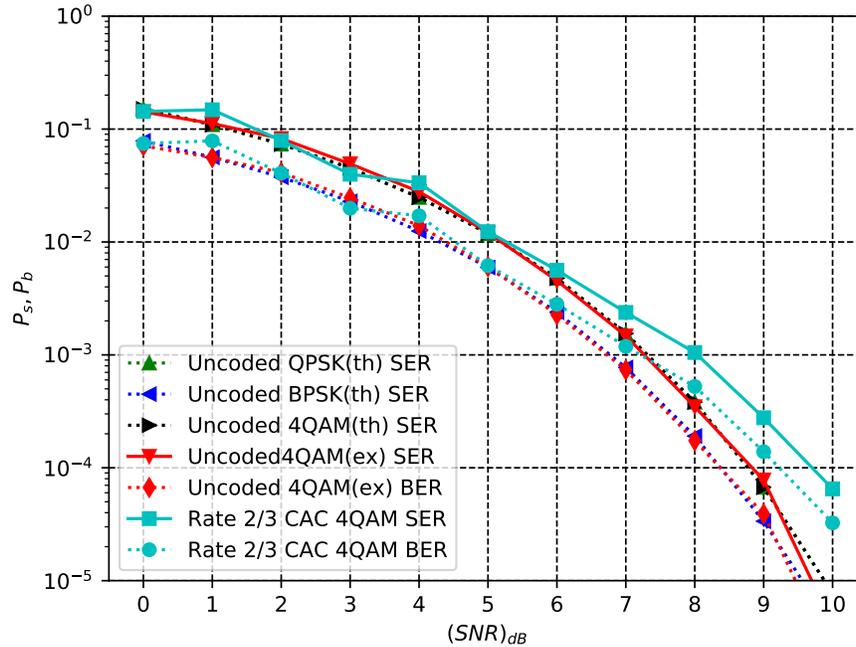
Fig. 3.22: BER and SER curve of rate $\frac{2}{3}$ CAC (PPE) on a QPSK constellation.
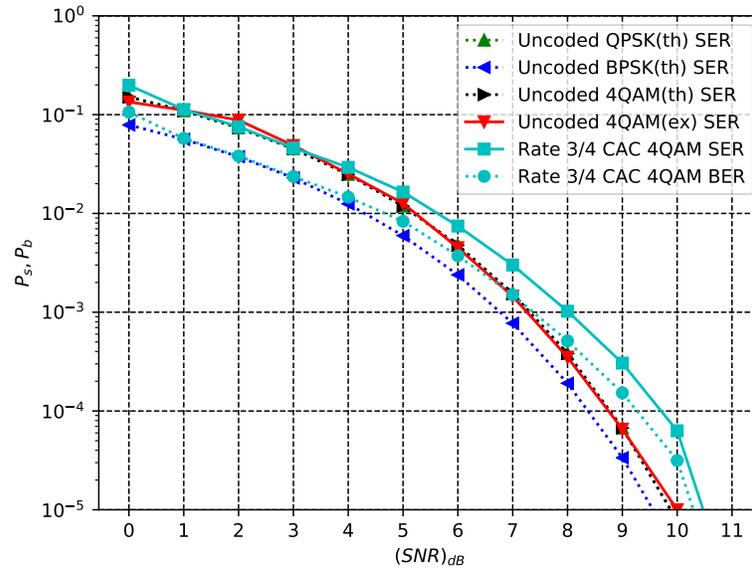
Fig. 3.23: BER and SER curve of rate $\frac{3}{4}$ CAC (PPE) on a QPSK constellation.
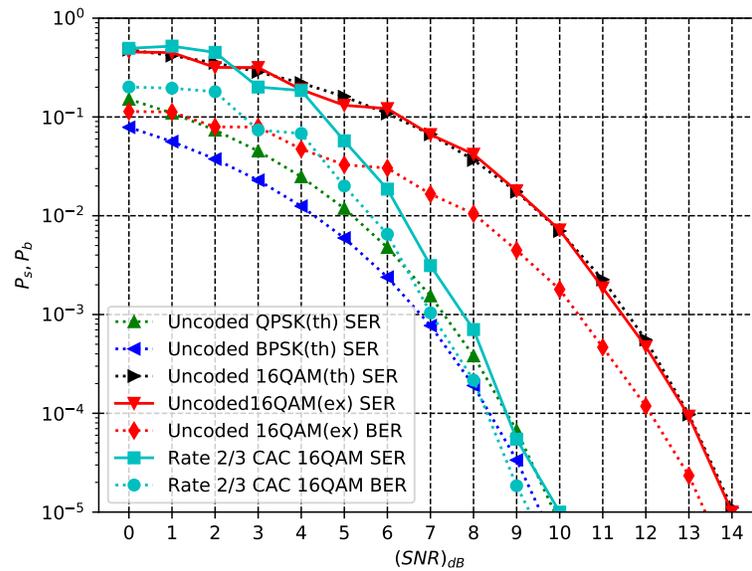
Fig. 3.24: BER and SER curve of rate $\frac{2}{3}$ CAC (PPE) on a 16QAM constellation.
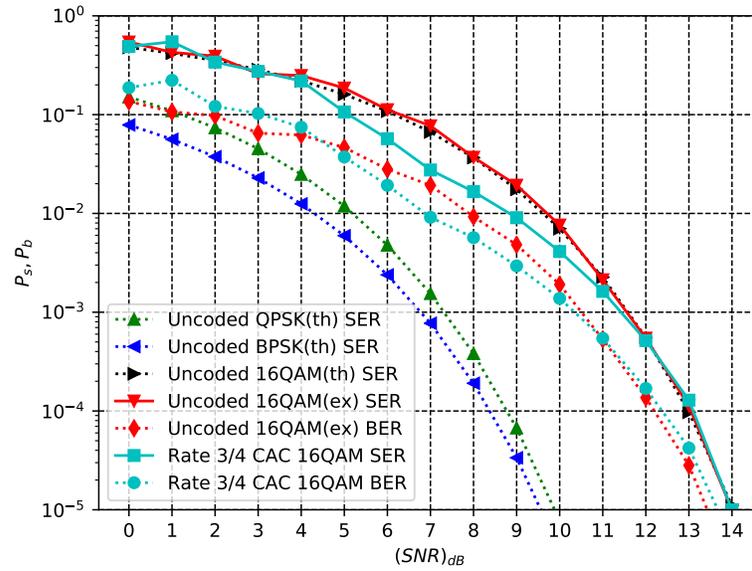
Fig. 3.25: BER and SER curve of rate $\frac{3}{4}$ CAC (PPE) on a 16QAM constellation.
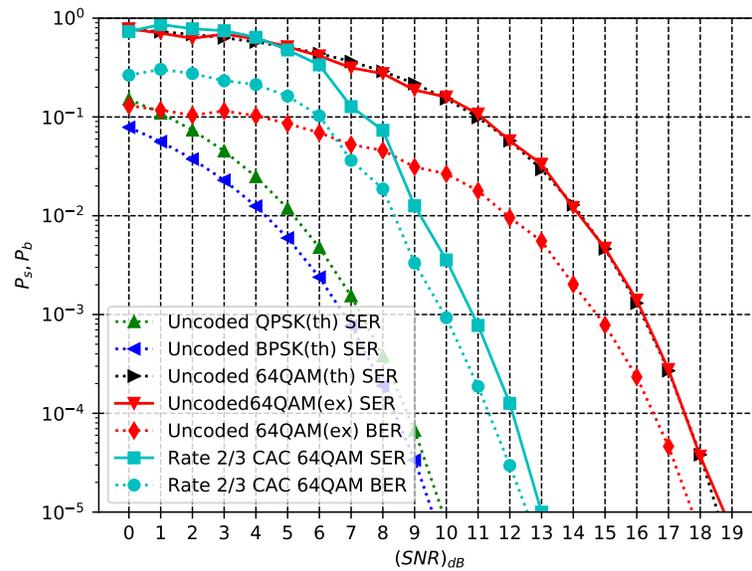
Fig. 3.26: BER and SER curve of rate $\frac{2}{3}$ CAC (PPE) on a 64QAM constellation.

3.4.3 Summarization of Results

Table 3.1 summarizes the performance of SPTC and CAC for different constellations and rates. In this table, for a given constellation coding gain is compared with the SER of the uncoded modulation using that constellation. For rate 2/3 and 3/4 codes and rate 1/2 SPTC using 64QAM, the free Euclidean distance d_{free}^2 is not provided in this table.

Constellation	Encoding Method	Rate	d_{free}^2	Coding Gain
QPSK	SPTC	1/2	$16E_{c,s}$	2.25 dB at $P_s = 10^{-5}$
QPSK	SPTC	1/2	$28E_{c,s}$	3.0 dB at $P_s = 10^{-5}$
QPSK	CAC	1/2	$12E_{c,s}$	2.0 dB at $P_s = 10^{-5}$
QPSK	CAC	1/2	$20E_{c,s}$	3.5 dB at $P_s = 10^{-5}$
QPSK	CAC (SIE)	2/3	—	1.5 dB at $P_s = 10^{-4}$
QPSK	CAC (PPE)	2/3	—	No Coding Gain
QPSK	CAC (SIE)	3/4	—	1.5 dB at $P_s = 10^{-5}$
QPSK	CAC (PPE)	3/4	—	No Coding Gain
16QAM	SPTC	1/2	$16E_{c,s}$	2.75 dB at $P_s = 10^{-4}$
16QAM	CAC	1/2	$12E_{c,s}$	3.0 dB at $P_s = 10^{-5}$
16QAM	CAC	1/2	$20E_{c,s}$	4.0 dB at $P_s = 10^{-5}$
16QAM	CAC	1/2	$28E_{c,s}$	5.0 dB at $P_s = 10^{-5}$
16QAM	CAC (SIE)	2/3	—	1.5 dB at $P_s = 10^{-6}$
16QAM	CAC (PPE)	2/3	—	4.0 dB at $P_s = 10^{-5}$
16QAM	CAC (SIE)	3/4	—	1.5 dB at $P_s = 10^{-5}$
16QAM	CAC (PPE)	3/4	—	No Coding Gain
64QAM	SPTC	1/2	—	2.5 dB at $P_s = 10^{-5}$
64QAM	CAC	1/2	$28E_{c,s}$	5.5 dB at $P_s = 10^{-5}$
64QAM	CAC (SIE)	2/3	—	1.0 dB at $P_s = 10^{-4}$
64QAM	CAC (PPE)	2/3	—	5.5 dB at $P_s = 10^{-5}$
256QAM	CAC	1/2	$28E_{c,s}$	5.0 dB at $P_s = 10^{-4}$

Table 3.1: Summarization of results obtained for SPTC and CAC.

3.5 Comparison Between Different Programming Languages Used for Simulations

In this thesis, three programming languages, Matlab version 9.4, Python version 3.6.4 and Julia version 0.6 have been used to perform various simulations. One of the objectives of this thesis is to compare among those three languages and determine which language is more suitable to study error correction coding theory. In order to compare, all the programs written in those languages has been made as similar to each other as possible in terms of syntax. For simulation, an object containing methods for encoding and decoding has been created and vectorized operations are used. While Matlab and Python are fully object-oriented language, Julia is not fully object-oriented in the conventional structure. In Julia, an object is called a composite type and is defined as Struct. But unlike Python and Matlab, in Julia, the methods are declared outside of Struct and then are linked to that Struct. To compare simulation time between Matlab, Python, and Julia, SPTC using the shape and QPSK constellation shown in Fig. 2.10 has been used as a reference and the simulations are performed using the same hardware. To take into account of randomness while generating bits, the average of 5 simulation time for each of the language has been considered. For SNR 0 to 7, the average simulation time required to generate BER and SER curves while counting 100 errors at each SNR is presented in Table 3.2.

Language	Version	Average Simulation Time (second)
Matlab	9.4	380.55
Python	3.6.4	487.95
Julia	0.6	155.40

Table 3.2: Average Simulation Times for Different programming Languages.

From Table 3.2, it can be seen that Julia is twice faster than Matlab and Python. This high speed of Julia can be useful for calculating the BER by counting a large number of errors at high SNR. Despite Julia being faster than the other two languages it has some

issues. As a fairly new language, the documentation for Julia is not as rich as Matlab or Python. Recent release Julia's stable version 1.1.1 has made some of the functions from version 0.6 deprecated. Another notable issue is while using Julia in Atom IDE, random crashes have been faced during simulations.

Both Python and Matlab have stable IDE, well-defined documentation and similar simulation speed while Julia is a lot faster than the other two. However, Python and Julia are open-source while Matlab can be expensive for students. Julia can be used in the study of error correction coding when simulation speed is an important requirement and Python can be used when stability during simulation is needed. Examining the different aspects of these programming languages, Julia is recommended among Matlab, Python, and Julia for the study of error correction coding because of its high simulation speed.

CHAPTER 4

PERFORMANCE OF CODES WITH LARGER STATES

In this chapter, an approach is presented for increasing the free Euclidean distance for Constellation Arithmetic Code by increasing the number of states during encoding. By exploiting the property of state variable, a modification in the encoding operation has been proposed to increase the number of states. Though this modification increases the d^2_{free}, it fails to provide any coding gain. Therefore this chapter provides a possible foundation for one of the future area for conducting research in Constellation Arithmetic Code.

4.1 Study the Performance of Codes for Different number of States:

For Constellation Arithmetic Code, the number of states is equal to the number of points in the signal space. From (2.2), the modulus operation enforces the state variable to stay between 0 to $M - 1$. By increasing the number of states, it is possible to create more possible paths in the trellis which can. The number of states can be changed by modifying (2.2) to :

$$j_{i+1} = j_i + r_i \ (\text{mod} \ N_s) \quad \text{where } N_s \text{ is number of state} \qquad (4.1)$$

This modification allows for the increment of the possible number of paths in the trellis. But just only by increasing the number of states doesn't increase the free Euclidean distance. Because, if the branches tend to reach the state variables which are in the close vicinity then the distance between the branches will not increase. So a multiplicative factor $\alpha \in \mathbb{Z}_{>0}$ is introduced to calculate the state variable which will enforce the branches to disperse and therefore increasing the free Euclidean distance.

$$j_{i+1} = \alpha(j_i + r_i) \ (\text{mod} \ N_s) \quad \text{where } N_s \text{ is number of state} \qquad (4.2)$$

Using these proposed modifications it is possible to increase the d^2_{free} for codes in a given

constellation by increasing N_s and finding α for that N_s. Table 4.1 contains the d_{free}^2 for the different number of N_s and α. From this table, it can be seen that the asymptotic coding gain can be increased by increasing N_s and using an appropriate α for a code in a given constellation. For Constellation Arithmetic Code on QPSK constellation using bit assignment shown in Fig. 2.16, a trellis of consisting of 13 number of states and $\alpha = 3$ is shown in Fig. 4.1. In this Fig. the input and parity symbols associated with each branch are placed in the left of the initial state of that branch. Starting from left, moving to right each input and parity symbol pair S/R corresponds to the branches selected from top to bottom. For example, located at the left of initial state 0, 2/2 corresponds to the third branch which has started at initial state 0 and ended at state 6. Now for this trellis, the pair of paths that corresponded to d_{free}^2 is shown in Fig. 4.2 by black solid lines. From those paths, the free Euclidean distance

$$d_{\text{free}}^2 = d_e^2(0,1) + d_e^2(0,1) + d_e^2(0,2) + d_e^2(0,1) + d_e^2(0,1) + d_e^2(0,1)$$

$$= 4E_{c,s} + 4E_{c,s} + 8E_{c,s} + 4E_{c,s} + 4E_{c,s} + 4E_{c,s}$$

$$= 28E_{c,s}.$$

Though increasing the N_s increases d_{free}^2, in this thesis no coding gain has been found for larger d_{free}^2 when $N_s > M$ where M is the number of points in the signal constellation. In Fig. 4.3 performances of CAC using 16QAM constellation with bit assignment from Fig. 2.20 for $N_s = 16$ with $d_{\text{free}}^2 = 28E_{c,s}$, $N_s = 64$ with $d_{\text{free}}^2 = 68E_{c,s}$ and $N_s = 128$ with $d_{\text{free}}^2 = 120E_{c,s}$ is shown. From the simulation results, it can be seen that no coding gain has been achieved.

$0/0, 1/1, 2/2, 3/3$ 0

$3/0, 0/1, 1/2, 2/3$ 1

$1/3, 2/0, 3/1, 0/2$ 2

$3/2, 0/3, 1/0, 2/1$ 3

$1/1, 2/2, 3/3, 0/0$ 4

$3/0, 0/1, 1/2, 2/3$ 5

$1/3, 2/0, 3/1, 0/2$ 6

$3/2, 0/3, 1/0, 2/1$ 7

$1/1, 2/2, 3/3, 0/0$ 8

$3/0, 0/1, 1/2, 2/3$ 9

$1/3, 2/0, 3/1, 0/2$ 10

$3/2, 0/3, 1/0, 2/1$ 11

$1/1, 2/2, 3/3, 0/0$ 12
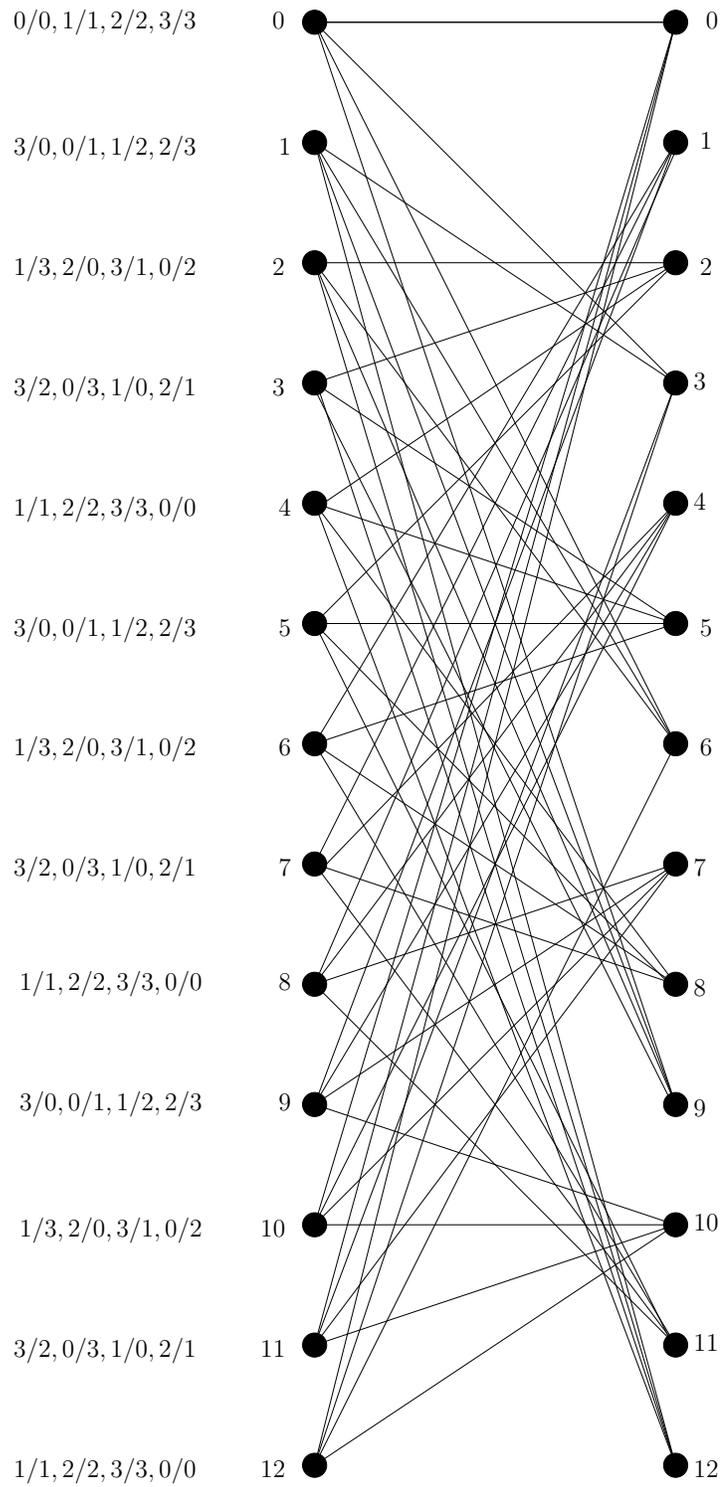
Fig. 4.1: Trellis structure for Constellation Arithmetic Code using QPSK constellation for $\alpha = 3$ and $N_s = 13$.
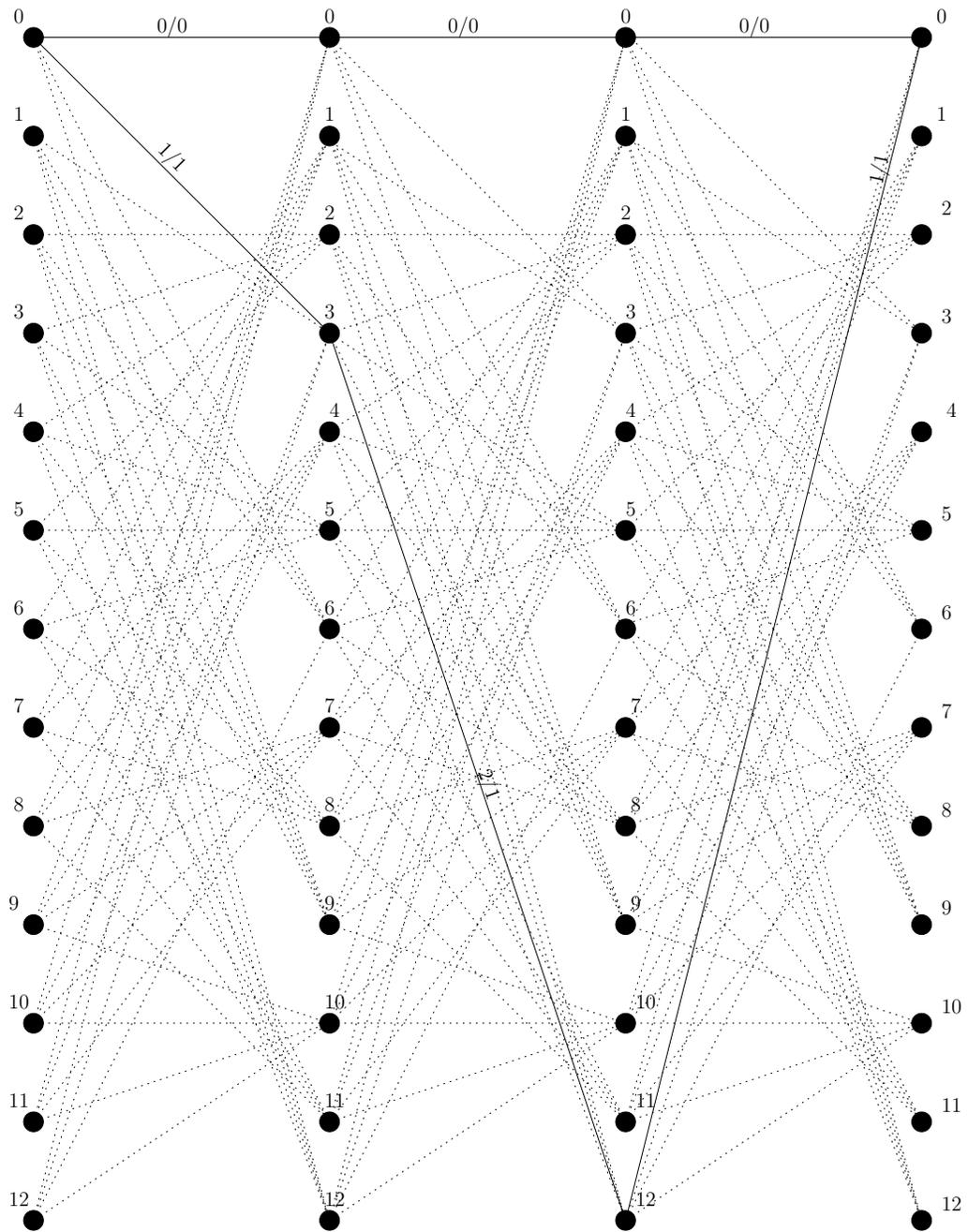
Fig. 4.2: Branches (black solid line) on the trellis that results in d_{free}^2 for CAC on QPSK constellation shown in Fig. 3.6.

constellation	Number of States, N_s	α	d_{free}^2	Asymptotic Coding Gain (dB)
QPSK	4	1	$12E_{c,s}$	1.7609
QPSK	4	1	$16E_{c,s}$	3.0103
QPSK	4	1	$20E_{c,s}$	3.9794
QPSK	11	6	$24E_{c,s}$	4.7712
QPSK	12	1	$32E_{c,s}$	6.0206
QPSK	13	3	$28E_{c,s}$	5.4407
QPSK	15	7	$24E_{c,s}$	4.7712
QPSK	16	7	$24E_{c,s}$	4.7712
QPSK	17	3	$28E_{c,s}$	5.4407
QPSK	17	6	$24E_{c,s}$	4.7712
QPSK	32	7	$32E_{c,s}$	6.0206
QPSK	32	9	$36E_{c,s}$	6.5321
QPSK	32	11	$36E_{c,s}$	6.5321
16QAM	16	1	$28E_{c,s}$	5.4407
16QAM	32	3	$36E_{c,s}$	6.5321
16QAM	32	5	$48E_{c,s}$	7.7815
16QAM	32	11	$36E_{c,s}$	6.5321
16QAM	32	13	$36E_{c,s}$	6.5321
16QAM	32	19	$36E_{c,s}$	6.5321
16QAM	32	27	$36E_{c,s}$	6.5321
16QAM	64	5	$68E_{c,s}$	9.2942
16QAM	64	11	$52E_{c,s}$	8.1291
16QAM	64	19	$52E_{c,s}$	8.1291
16QAM	64	29	$60E_{c,s}$	8.7506
16QAM	64	35	$64E_{c,s}$	9.0309
16QAM	64	37	$52E_{c,s}$	8.1291
16QAM	64	45	$60E_{c,s}$	8.7506
16QAM	128	19	$100E_{c,s}$	10.9691
16QAM	128	23	$88E_{c,s}$	10.4139
16QAM	128	39	$120E_{c,s}$	11.7609
16QAM	128	53	$92E_{c,s}$	10.6070
64QAM	64	1	$28\ E_{c,s}$	5.4407
256QAM	256	1	$28\ E_{c,s}$	5.4407

Table 4.1: Free Euclidean Distance and Asymptotic Coding Gain for Different Number of States.
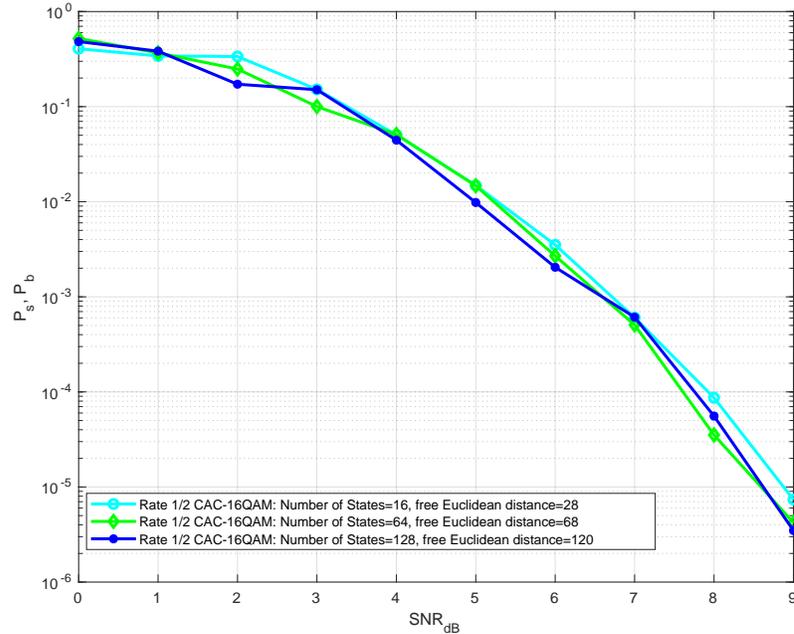
Fig. 4.3: Simulation results for rate 1/2 CAC-16QAM using different number of states.

The proposed method in this section fails to provide coding gain but it can be used as an insight for increasing the coding gain by increasing the number of states. Therefore further research is required in topic to find a viable solution that provides coding gain by increasing the number of states for CAC on any constellation.

4.2   Comparison Between The Performances of Some Well Known TCM and CAC

In TCM a convolutional encoder is used to encode the input bits and then the coded bits are mapped to a larger signal constellation. However, in CAC, each input symbol is encoded to generate a parity symbol. Therefore in this section, the comparison between TCM and CAC is done for the codes with the same *spectral efficiency*. The spectral efficiency is the number of information bits transmitted by each symbol [9]. The Trellis Codes for 8-PSK uses a rate 2/3 convolutional encoder followed by mapping to an 8-PSK signal constellation. So each symbol carries 2 information bits. For the rate 1/2 CAC on 16QAM constellation, each

symbol carries 2 information bit. Therefore the rate 1/2 CAC on 16QAM constellation and the Trellis Codes for 8-PSK have spectral efficiency of 2. Table 4.2 contains the comparison between the rate 1/2 CAC on 16QAM constellation and the Trellis Codes for 8-PSK [9] using asymptotic coding gain for different number of states.

Number of States	Asymptotic Coding Gain (dB) (coded/uncoded)	
	8-PSK/4-PSK	16QAM/QPSK
16	4.1	5.44
32	4.6	7.7
64	5.0	8.12
128	5.2	10.4

Table 4.2: Comparison Between Rate 1/2 CAC on 16QAM Constellation and Trellis Codes for 8-PSK using Asymptotic Coding Gain.

CHAPTER 5

CONCLUSION AND FUTURE WORK

The research presented in this thesis provides a detailed study of Signal Point Target Code. By investigating various aspects of SPTC, the proposed code has been extended for larger signal constellations, different rates, and larger states. The findings from this work provide directions for future research in SPTC.

5.1   Contribution

The contributions of this thesis can be summarized as given below:

- The operation of the SPTC has been described by introducing a notation.

- The relation between the employed shape and the performance of SPTC has been examined using free Euclidean distance. For different shapes, the performance of SPTC has been evaluated on different QAM signal constellations.

- An alternative of SPTC, called Constellation Arithmetic Code has been presented.

- The relation between the bit assignment in the signal constellation and the performance of CAC has been examined using free Euclidean distance. Using different bit assignments and M-ary QAM signal constellations, the performance of CAC has been evaluated.

- A method for bit assignment in the signal constellation has been proposed that provides coding gains for CAC compared to performances of SPTC examined in this thesis.

- Using the CAC, two methods of increasing the rate of codes have been proposed and their performances for different rates have been simulated.

- The free Euclidean distances and asymptotic coding gains of CAC for larger number of states has been calculated. A method for increasing d_{free}^2 using larger number of states has been proposed. Form proposed method, one possible future research avenue for the code has been outlined.

- A suitable language among Matlab, Python, and Julia has been determined for the study of error correction coding by considering simulation speed. Different aspects of these programming languages have been discussed. By calculating average simulation speed, it has been found that Julia is the fastest and most suitable language among those three.

5.2   Conclusion

Both SPTC and CAC provides coding gain compared to the performance of uncoded modulation. The coding gain can be increased for SPTC by employing shapes that provide larger free Euclidean distance and for CAC by using bit assignments which results in larger free Euclidean distance. In this thesis, for QPSK constellation the maximum coding gain found for SPTC is 3 dB and for CAC is 3.5 dB. The shape used for SPTC 16QAM provides a coding gain of 2.75 while for CAC up to of 5 dB coding gain has been achieved. The proposed method for bit assignment in CAC for 64QAM and 256QAM provides good coding gain. From the proposed methods of increasing rates, SIE provides coding gain for all codes. Among Matlab, Python, and Julia, Julia has been found twice faster than the other two languages in terms of simulation speed. Therefore Julia has been selected as a suitable language for studying error correction coding between those three. Though the proposed method for increasing the number of states increases the free Euclidean distance and asymptotic coding gain, it fails to provide any coding gain compared to the performance of CAC using M number of states.

5.3   Future Work

The future work for this thesis includes finding the best shape for SPTC and bit

assignment for CAC on any given constellation. Further research is needed to develop a method similar to set partitioning used in TCM in order to maximize the free Euclidean distance on the trellis. As the PPE provides good coding gain for some selected codes, this method should be investigated further. The proposed method for increasing number of states needs to be studied more to find out why it failed to provide coding gain.

Overall, by providing a detailed study of the Signal Point Target code and by exploring and extending various attributes of the proposed codes, the objectives of this thesis have been investigated.

REFERENCES

[1] C. E. Shannon, "A mathematical theory of communication," *Bell Sys. Tech. Journal*, vol. 27, no. 3, pp. 379–423, July 1948.

[2] R. Gallager, "Low-density parity-check codes," *IRE Tran. Inf. Theory*, vol. 8, no. 1, pp. 21–28, January 1962.

[3] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1261–1271, Oct 1996.

[4] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, July 2009.

[5] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inf. Theory*, vol. 28, no. 1, pp. 55–67, January 1982.

[6] J. Sodha, "Rotating signal point shape code," in *Intelligent Computing*. Springer International Publishing, 2019, pp. 876–880.

[7] J. Sodha and T. Moon, "Signal point target code," in *Proc. IEEE 62nd IMWSCAS (Accepted)*, Aug 2019.

[8] J. Sodha and A. Als, "Shape nature of error-control codes," *Elsevier Sig. Process. Journal*, vol. 83, no. 7, pp. 1457 – 1465, 2003.

[9] T. Moon, *Error correction coding : mathematical methods and algorithms*. Hoboken, N.J: Wiley-Interscience, 2005.

[10] J. Proakis and M. Salehi, *Digital Communications*. McGraw-Hill, 2008.

APPENDICES

APPENDIX A

Q function

$Q(x)$ is the area under the tail of a standard normal distribution. If X is a standard Gaussian random variable then $Q(x)$ is the probability that X is greater than x, so

$$Q(x) = P(X > x)$$

$$= 1 - P(X \leq x)$$

$$= 1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{u^2}{2}} \, du$$

$$= \frac{1}{\sqrt{2\pi}} \int_{x}^{\infty} e^{-\frac{u^2}{2}} \, du$$

$$= \frac{1}{2} \left(\frac{2}{\sqrt{2\pi}} \int_{x}^{\infty} e^{-\frac{u^2}{2}} \, du \right)$$

$$= \frac{1}{2} \left(\frac{2}{\sqrt{\pi}} \int_{\frac{x}{\sqrt{2}}}^{\infty} e^{-t^2} \, dt \right)$$

$$= \frac{1}{2} \text{erfc} \left(\frac{x}{\sqrt{2}} \right).$$

APPENDIX B

Code

B.1   Matlab Code

Main Function

```matlab
1  clc;
2  clear all;
3  close all;
4  tic
5  %Initialize
6  docnostanttarget=0;                % 0 for same target point 1 for ...
       rotating target point
7  M=4;                               %constellation size
8  doMquam=0;                          %Mquam
9  doShapeCode=1;
10 nerrortocount=100;                 %number of error to count at each SNR
11 SNRrange=0:1:8;
12 SNRrangeex=0:1:8;
13 SNRrangetheo=0:1:14;
14 GraphPsrange=10^-6;
15 A=1;
16 %create object
17 approach=2;                        %approach=1 using sum table;
18                                     %approach=2 using mod sum ;
19                                     %approach=3 dr Sodha's approach
20 shapecode_object=shapecode(docnostanttarget,M,A,approach);
21 shapecode_object.setdecoderinfo(1);               % 1=plot trellis ...
       structure of decoder (only for qpsk)
22 shapecode_object.resetshapecoder();
23 shapecode_object.resetshapedecoder();
24 R=1;                               %rate of uncoded symbol
25 Eb=2/3*A^2*(M-1)/(shapecode_object.Nbits*R)                          ...
       %energy per bit
26
27 Es=shapecode_object.Nbits*Eb*R; %Eneregy of uncoded symbol
28
29 Esscale=sqrt(3*Es/(2*(M-1)));
30 Rcoded=.5;                         %rate of coded symbol
31 %Escoded=sqrt(shapecode_object.Nbits*Eb*Rcoded*3/(2*(M-1)));        ...
       %Coded Amplitude
32 Escoded=2/3*(shapecode_object.M-1)*A^2;
33 %ploting thoeritical SNR (Plots are consistant with Figure 5-2-16 Digital ...
       Communication By John G.Proakis)
```

```matlab
34  snrctr=0;
35
36  for SNR=SNRrangetheo
37      %fprintf('Theoritical SNR=%d\n',SNR)
38      snrctr=snrctr+1;
39      N0=Eb*10^(-SNR/10);
40      EbN0=Eb/N0;
41      theorprobQPSK(snrctr) = 1 - (1-qf(sqrt(2*EbN0)))^2;
42      %theorprobBPSK(snrctr) = qf(sqrt(2*EbN0));
43      theoprobMquam(snrctr)=(1-(1-2*(1-sqrt(1/M))....
44                  *qf(sqrt((3/(M-1))*shapecode_object.Nbits*EbN0)))^2);
45  end
46
47  legendstrs={};
48  lct=1;
49  legendstrs{lct} = 'QPSK(th) symbol';
50  %legendstrs{lct+1} = 'BPSK(th) symbol';
51  %legendstrs{lct+2} = '64QAM(th) symbol';
52  %legendstrs{lct+1} = '16QAM(th) symbol';
53  lct=2;
54  if (doMquam)
55      legendstrs{lct} = '16QAM(ex) symbol';
56      lct=lct+1;
57      legendstrs{lct}='16QAM(ex) bit';
58      lct=lct+1;
59  end
60
61  if (doShapeCode)
62      legendstrs{lct} = 'SPTC QPSK(symbol) state 10';
63      lct=lct+1;
64      legendstrs{lct} = 'SPTC QPSK(bit) state 10';
65  end
66  figure;
67  semilogy(SNRrangetheo,theorprobQPSK,'b:','linewidth',2);hold on;
68  %semilogy(SNRrangetheo,theorprobBPSK,'g:','linewidth',2);
69  %semilogy(SNRrangetheo,theoprobMquam,'k:','linewidth',2);
70  xticks(SNRrangetheo)
71  yticks([10^-6 10^-5 10^-4 10^-3 10^-2 10^-1 10^-0])
72  ylim([GraphPsrange 10^-0])
73  set(gca,'fontsize',15)
74  set(gcf, 'Position',  [2357 152 821 624])
75  xlabel('SNR_{db}')
76  ylabel('P_s')
77  grid on
78  %ploting Uncoded SNR (Plots are consistant with Figure 5-2-16 Digital ...
        Communication By John G.Proakis)
79  if (doMquam)
80      snrctr=0;
81      for SNR=SNRrangeex
82          fprintf('Experimental SNR=%d\n',SNR)
83          snrctr=snrctr+1;
84          N0=Eb*10^(-SNR/10);
85          EbN0=Eb/N0;
86          sigma=sqrt(N0/2);
87          errctr(snrctr)=0;
88          errctr_bits(snrctr)=0;
```

```matlab
89              nsyms(snrctr)=0;
90              while(errctr(snrctr)<nerrortocount)
91                  nsyms(snrctr)=nsyms(snrctr)+1;
92
93                  bits=randombits(shapecode_object.Nbits);
94                  symno=Binary2Decimal(bits);
95
96                  %constpts=Esscale*shapecode_object.const_pts(:,symno+1);
97                  constpts=shapecode_object.const_pts(:,symno+1);
98
99
100                 noise=sigma*randn(1,2);
101                 r(1)=constpts(1)+noise(1);
102                 r(2)=constpts(2)+noise(2);
103
104                 decodesymno=mquamdemod(r,shapecode_object.const_pts,....
105                 shapecode_object.const_axis);
106
107                 if(decodesymno≠symno)
108                     errctr(snrctr)=errctr(snrctr)+1;
109                     errctr_bits(snrctr) = errctr_bits(snrctr) + sum(bits ≠ ...
                            fliplr(de2bi(decodesymno,shapecode_object.Nbits)));
110                 end
111             end
112         end
113 end   %if do Mquam
114
115 if (doMquam)
116     semilogy(SNRrangeex,errctr ./ nsyms,'r','linewidth',2);
117     semilogy(SNRrangeex,(errctr_bits./(shapecode_object.Nbits*nsyms)),...
118         'r:','linewidth',2);
119 end
120
121
122 Eb=Escoded/(Rcoded*log2(shapecode_object.M))
123
124 if (doShapeCode)
125     snrctr=0;
126     for SNR=SNRrange
127         fprintf('ShapeCode SNR=%d\n',SNR)
128
129         snrctr=snrctr+1;
130         %N0=Eb*10^(-(SNR-10*log10(Rcoded))/10);
131         N0=Eb*10^(-(SNR)/10);
132         EbN0=Eb/N0;
133         sigma=sqrt(N0/2);                   %calculate sigma for given SNR
134         errctr_coded(snrctr)=0;
135         errctr_bits_coded(snrctr)=0;
136         nsyms_coded(snrctr)=0;
137         while (errctr_coded(snrctr)<nerrortocount)
138             nsyms_coded(snrctr)=nsyms_coded(snrctr)+1;    %calculate ...
                    number of genarted symbols
139
140
141             if(0)
```

```matlab
142                 if (rem(nsyms_coded(snrctr),10000)==0 && ...
                        (nsyms_coded(snrctr)≥ 10000))
143                     fprintf('number of symbol=%d \n',nsyms_coded(snrctr))
144                 end
145                 end
146
147
148                 bits=randombits(shapecode_object.Nbits);        %genarte random ...
                        bits
149                 [sym1,sym2]=shapecode_object.encode(bits);     %encode the bits
150
151                 shapecode_object.bitsqueuein(bits);            %save the bits ...
                        to compare the results
152
153                 %s1=Escoded*shapecode_object.const_pts(:,sym1+1);      ...
                        %genrate signal points to transmit (symbol)
154                 %s2=Escoded*shapecode_object.const_pts(:,sym2+1);      ...
                        %%genrate signal points to transmit (parity)
155                 s1=shapecode_object.const_pts(:,sym1+1);
156                 s2=shapecode_object.const_pts(:,sym2+1);
157
158                 r1=s1+sigma*randn(2,1);                         %add noise
159                 r2=s2+sigma*randn(2,1);                         %add noise
160
161                 %[bitsout,decodeout]=shapecode_object.viterbishapedecode...
162             (r1/Escoded,r2/Escoded);    %decode
163                 [bitsout,decodeout]=shapecode_object.viterbishapedecode(r1,r2);
164
165                 if(decodeout)
166                     inbits=shapecode_object.bitqueueout();
167
168                     if(0)
169                     if (isequal(inbits,bitsout)==0)
170
171                         %save('data_shapecode.mat')      %save workspace to file
172                         fprintf('\nCoded SNR=%d Error ...
                            No=%d\n',SNR,errctr_coded(snrctr)+1)
173 %                       fprintf('InputBits=%d%d%d%d ...
        OutputBits=%d%d%d%d\n',inbits(1),inbits(2),....
174 %             inbits(3),inbits(4),bitsout(1),bitsout(2),bitsout(3),bitsout(4));
175                     end
176                     end
177
178
179
180                     errctr_coded(snrctr)=errctr_coded(snrctr)+any(inbits ≠ ...
                            bitsout);          %calculate number of symbol error
181                     errctr_bits_coded(snrctr) = errctr_bits_coded(snrctr) + ...
                            sum(inbits ≠ bitsout);    %calculate number of bit error
182                 end
183
184 %           if (errctr_coded(snrctr)≠0 && ...
        (errctr_coded(snrctr)/nsyms_coded(snrctr))≤GraphPsrange)
185 %               break
186 %           end
187             % errctr_coded(snrctr)
```

```
188          end %while
189 %          if ((errctr_coded(snrctr)/nsyms_coded(snrctr))≤GraphPsrange)
190 %              break;
191 %          end
192
193 if (doShapeCode)
194
195     semilogy((0:1:(length(errctr_coded)-1)),....
196     (errctr_coded./nsyms_coded),'c','linewidth',2);
197     semilogy((0:1:(length(errctr_coded)-1)),....
198     (errctr_bits_coded./(shapecode_object.Nbits*nsyms_coded)),....
199     'c:','linewidth',2)
200     legend(legendstrs)
201     pause(0.2)
202     %saveas(gcf,'64QAMshapecodeBERrate12approach2','pdf');
203 end
204
205
206     end  %for SNRRange
207 end   %doshapecode
208
209 toc
210 %saveas(gcf,'qpskshapecodeBERrate12approach2state10mult3march20','pdf');
211 %saveas(gcf,'16quamshapecodeBERrate12approach3march17','pdf');
212 %save('data_shapecodeqpskapproach2state10mult3march20.mat')
```

Q function

```
1 function q=qf(x)
2 q=.5*erfc(x/sqrt(2));
```

Function for Binary to Decimal Conversion

```
1 function symbol=Binary2Decimal(k)
2 symbol=0;
3 for ind=1:1:length(k)
4     symbol=symbol+k(ind)*2^(length(k)-ind);
5 end
```

Decoder for Uncoded QAM

```
1 function symbol=mquamdemod(r,const_ptsf,const_axis)
2 [¬,Ix]=min(abs(r(1)-const_axis));
3 [¬,Iy]=min(abs(r(2)-const_axis));
```

```
4  indxx=const_axis(Ix);
5  indyy=const_axis(Iy);
6  d=[indxx indyy];
7  [¬,Locb]=ismember(const_ptsf',d,'rows');
8  symbol=find(Locb==1)-1;
```

Random Bit Generator Function

```
1  function bits=randombits(k)
2  bits=double(rand(1,k)≥.5);
```

Shape Code Object

```
1  classdef shapecode< handle
2      properties (Constant)
3          viterbiwindowwidth=20;
4      end
5      properties
6          M
7          A
8          Nstate
9          Nepochs
10         Nbits
11         cumrotoffset
12         fromtodist
13         const_pts
14         const_axis
15         xcycle
16         sumrotcount                % cumulation of rotations(states)
17         epoch
18         doconstanttarget           % whether target rotation moves
19         trellisoutput
20         inputbits
21         fromtostate
22         metrics
23         nextmetrics
24         epoch_decode
25         prevstatearray
26         front
27         countbranches
28         beststate
29         bitqueue
30         bitqueuefront
31         bitqueueback
32         rotA
33         vec
34         AddRot
35         testbits
```

```matlab
36          approach
37          multipliedfactor
38      end
39
40  %------------------------------------------------------------------
41      methods
42          function obj=shapecode(doconstanttarget,M,A,approach)
43              obj.A=A;
44              obj.doconstanttarget=doconstanttarget;
45              obj.M=M;
46              obj.Nstate=obj.M;
47              obj.Nstate=6;
48              obj.Nepochs=obj.M;
49              obj.Nbits=log2(obj.M);
50              obj.approach=approach;
51              obj.multipliedfactor=1;
52              %obj.multipliedfactor=1;
53              if (obj.approach==3 && obj.M==16)
54              %%% Dr Sodha's constellation point for 16 quam
55
56               obj.const_pts=[-3 -3;-3 -1;-3 3;-3 1;-1 -3;-1 -1;-1 3;-1 1;3 ...
                      -3;3 -1;3 3;3 1;1 -3;1 -1;1 3;1 1]';   %points in the ...
                      constellation
57               obj.const_axis=[-3 -1 3 1];
58
59
60              elseif (obj.approach==3 && obj.M==4)
61              %%% Dr Sodha's constellation point for 16 quam
62
63               obj.const_pts=[-1 -1;1 -1;-1 1;1 1]';   %points in the ...
                      constellation
64               obj.const_axis=[-1 1];
65              else
66
67              %%% My constellation point for square M quam
68
69              obj.const_axis=-(2^(obj.Nbits/2)-1):2:2^(obj.Nbits/2)-1;
70              obj.const_pts=[kron(ones(1,2^(obj.Nbits/2)),obj.const_axis); ...
                      kron(obj.const_axis,ones(1,2^(obj.Nbits/2)))];
71              end
72              if (obj.M==16)
73              obj.rotA=[0 7 15 8 1 6 14 9 3 4 12 11 2 5 13 10];      %from ...
                      dr Sodha's code
74              obj.vec=[2 6 14 10 11 15 7 3 1 5 13 9 8 12 4 0];       %from ...
                      dr Sodha's code
75              end
76              obj.const_pts=obj.const_pts*obj.A;
77              obj.const_axis=obj.const_axis*obj.A;
78
79              if (obj.M==4)
80              obj.rotA=[0 1 3 2];       %from dr Sodha's code
81              obj.vec=[2 3 1 0];        %from dr Sodha's code
82              end
83
84              obj.AddRot=CodeTrellis(obj);                          % sum ...
                      table from dr Sodha's code
```

```matlab
85
86
87              %obj.cumrotoffset=hankel(0:1:obj.M-1,[obj.M-1 0:1:obj.M-2]);
88              obj.cumrotoffset=obj.AddRot;
89              %obj.fromtodist=hankel(0:1:obj.M-1,[obj.M-1 0:1:obj.M-2]);
90              obj.fromtodist=[0  7  15  8  1  6  14  9  3  4  12  11  2  5  13  10 ;
91                              9  0  8  1  10  15  7  2  12  13  5  4  11  14  6  3 ;
92                              1  8  0  9  2  7  15  10  4  5  13  12  3  6  14  11;
93                              8  15  7  0  9  14  6  1  11  12  4  3  10  13  5  2;
94                              15  6  14  7  0  5  13  8  2  3  11  10  1  4  12  9;
95                              10  1  9  2  11  0  8  3  13  14  6  5  12  15  7  4;
96                              2  9  1  10  3  8  0  11  5  6  14  13  4  7  15  12;
97                              7  14  6  15  8  13  5  0  10  11  3  2  9  12  4  1;
98                              13  4  12  5  14  3  11  6  0  1  9  8  15  2  10  7;
99                              12  3  11  4  13  2  10  5  15  0  8  7  14  1  9  6;
100                             4  11  3  12  5  10  2  13  7  8  0  15  6  9  1  14;
101                             5  12  4  13  6  11  3  14  8  9  1  0  7  10  2  15;
102                             14  5  13  6  15  4  12  7  1  2  10  9  0  3  11  8;
103                             11  2  10  4  12  1  9  4  14  15  7  6  13  0  8  5;
104                             3  10  2  11  4  9  1  12  6  7  15  14  5  8  0  13;
105                             6  13  5  14  7  12  4  15  9  0  2  1  8  11  3  0 ];
106
107          if (¬doconstanttarget)
108              if (obj.approach==3)
109              obj.xcycle=obj.rotA;              %set the target point
110              else
111              obj.xcycle=0:1:obj.M-1;
112              end
113          else
114              obj.xcycle=0;
115          end
116
117          obj.epoch=0;                          %number of epoch
118          obj.sumrotcount=0;                    %total number of previous ...
                 rotation
119
120
121      end %end of contructor
122  %---------------------------------------------------------------------
123  % set AddRot Matrix
124
125
126
127  %      function temp=IntializeVec(obj,N)
128  %          for r=0:1:N-1
129  %              for c=0:1:N-1
130  %                  temp(r+1,c+1)=0;
131  %              end
132  %          end
133  %      end
134
135      function temp=CircularShift(obj,A,shift)
136          temp=ones(1,length(obj.vec));
137
138          for r=0:1:length(A)-1
139
```

```
140                    if ((r+shift)<obj.M)
141
142                        temp(r+1)=A(r+1+shift);
143                    else
144
145                        temp(r+1)=A(r+1+shift-obj.M);
146                    end
147                end
148            end
149
150            function AddRot=InsertVector(obj,AddRot,A,index,shift)
151                newvec=obj.CircularShift(A,shift);
152                for c=0:1:length(A)-1
153                    AddRot(index,c+1)=newvec(c+1);
154                end
155            end
156
157
158            function temp=CodeTrellis(obj)
159                temp=ones(obj.M,obj.M);
160                for ind=0:1:length(obj.vec)-1
161                    temp=obj.InsertVector(temp,obj.vec,obj.vec(ind+1)+1,ind);
162                end
163            end
164  %------------------------------------------------------------
165            function resetshapecoder(obj)
166
167                obj.epoch=0;
168                obj.sumrotcount=0;
169            end
170    %------------------------------------------------------------
171        function resetshapedecoder(obj)
172            obj.epoch_decode=0;
173            obj.front=0;                    % index into prevstatearray
174            obj.countbranches=0;
175            obj.bitqueuefront=0;
176            obj.bitqueueback=0;
177        end
178  %------------------------------------------------------------
179        function m=computemetric(obj,r1,r2,output)
180            out11=output(1,1);
181            out12=output(1,2);
182            out21=output(2,1);
183            out22=output(2,2);
184            n1=(r1(1)-out11)^2+(r1(2)-out21)^2;
185            n2=(r2(1)-out12)^2+(r2(2)-out22)^2;
186            m=n1+n2;
187        end
188    %------------------------------------------------------------
189            function [bitsout,decodeout]=viterbishapedecode(obj,r1,r2)
190                beststatemetric=inf;
191                for state=0:obj.Nstate-1
192                    nextmetricbest=inf;
193                    for prevstate=obj.fromtostate(:,state+1,obj.epoch_decode+1)'
194                        m=obj.computemetric(r1,r2,....
195                   obj.trellisoutput(:,:,prevstate+1,state+1,obj.epoch_decode+1));
```

```matlab
196                     nextmetric=obj.metrics(prevstate+1)+m;
197
198                     if (nextmetric<nextmetricbest)
199                         nextmetricbest=nextmetric;
200                         prevstatebest=prevstate;
201                         statebest=state;
202                     end
203                 end %for prevstate
204                 if(nextmetricbest<beststatemetric)
205                     beststatemetric=nextmetricbest;
206                     obj.beststate=state;
207                 end
208                 obj.nextmetrics(state+1)=nextmetricbest;
209                 obj.prevstatearray(state+1,obj.front+1)=prevstatebest;
210             end %for state
211             obj.metrics=obj.nextmetrics;
212             obj.countbranches=obj.countbranches+1;
213             decodeout=0;
214             bitsout=[0;0];
215
216
217         if(obj.countbranches≥obj.viterbiwindowwidth)
218             % start from best state and work back
219             state=obj.beststate;
220             idx=obj.front;
221             epoch1=obj.epoch_decode;
222
223             for i=1:obj.viterbiwindowwidth
224                 prevstate=obj.prevstatearray(state+1,idx+1);   % go back ...
                        to previous state
225                 idx=mod(idx-1,obj.viterbiwindowwidth);        % circular ...
                        indexing
226                 lastepoch=epoch1;
227                 epoch1=mod(epoch1-1,obj.Nepochs);
228
229                 if(i≠obj.viterbiwindowwidth)
230                     state=prevstate;
231                 end
232             end
233           bitsout=obj.inputbits(:,prevstate+1,state+1,lastepoch+1);
234           decodeout=1;
235         end
236         obj.front=mod(obj.front+1,obj.viterbiwindowwidth);
237         obj.epoch_decode=mod(obj.epoch_decode+1,obj.Nepochs);
238       end
239
240   %----------------------------------------------------
241
242       function bitsout=viterbishapedecodeflush(obj)
243           bitsout = [];
244           state = obj.beststate;
245
246           epoch1 = mod(obj.epoch_decode-1,obj.Nepochs);
247           idx = mod(obj.front-1,obj.viterbiwindowwidth);
248
249           for i = 1:obj.viterbiwindowwidth-1
```

```matlab
250                 prevstate = obj.prevstatearray(state +1, idx +1);    % go ...
                        back to previous state
251                 lastepoch = epoch1;
252                 bits1 = obj.inputbits(:,prevstate +1,state +1,lastepoch +1);
253                 idx = mod(idx - 1,obj.viterbiwindowwidth);    % circular ...
                        indexing
254                 epoch1 = mod(epoch1 - 1,obj.Nepochs);
255                 bitsout = [bits1 bitsout];
256               state = prevstate;
257             end
258
259       end
260
261  %-------------------------------------------------------
262       function bitsqueuein(obj,bits)
263          obj.bitqueue(:,obj.bitqueuefront+1)=bits(:);
264          obj.bitqueuefront=mod(obj.bitqueuefront+1,obj.viterbiwindowwidth);
265       end
266  %-------------------------------------------------------
267
268       function bits=bitqueueout(obj)
269          bits=obj.bitqueue(:,obj.bitqueueback+1);
270          obj.bitqueueback=mod(obj.bitqueueback+1,obj.viterbiwindowwidth);
271       end
272
273
274  %-------------------------------------------------------
275        function symbol=Bin2DecClass(obj,k)
276          symbol=0;
277            for ind=1:1:length(k)
278             symbol=symbol+k(ind)*2^(length(k)-ind);
279            end
280          end
281
282  %-------------------------------------------------------
283          function [symnosave,coderot]=encode(obj,bits)
284                  symno=obj.Bin2DecClass(bits);
285                  symnosave=symno;
286
287
288  %                %%%% Encoding using Dr Sodha's approach
289                  if (obj.approach==3)
290                      newpoint=obj.AddRot(symno+1,obj.sumrotcount+1);
291
292                      coderot=obj.rotA(newpoint+1);
293                      %coderot is code symbol signal point
294
295                      %update total number of previous rotation
296  %                    if (obj.sumrotcount+coderot>=obj.M)
297  %                        obj.sumrotcount=obj.sumrotcount+coderot-obj.M;
298  %                    else
299  %                        obj.sumrotcount=obj.sumrotcount+coderot;
300  %                    end
301                    obj.sumrotcount=mod(obj.sumrotcount+coderot,obj.M);
302                      %obj.sumrotcount is the total rotation
303                  end
```

```matlab
304  %                %%%%% end of Dr Shodha's approach
305
306
307                %%%% approach 1
308                if (obj.approach==1)
309                    symno=obj.cumrotoffset(symno+1,obj.sumrotcount+1);
310                end
311
312                %%%% approach 2
313                 if (obj.approach==2)
314                    symno=mod(symno+obj.sumrotcount,obj.Nstate);
315                 end
316
317                %%%%  used in  both approach 1 and 2:
318                if (obj.approach ==1 || obj.approach ==2)
319                   if(¬obj.doconstanttarget)
320                       xcyclecount=obj.epoch;
321                   else
322                       xcyclecount=0;
323                   end
324
325                   tosym=obj.xcycle(xcyclecount+1);
326                 end
327               %approach 1
328               if (obj.approach==1)
329                   coderot=obj.fromtodist(tosym+1,symno+1);
330               end
331
332               %approach 2
333               if (obj.approach==2)
334                   coderot=mod(symno+tosym,obj.Nstate);
335               end
336
337
338               %%%% used in both appraoch 1 and 2:
339               if (obj.approach ==1 || obj.approach ==2)
340                   obj.sumrotcount=mod(obj.multipliedfactor*....
341           (obj.sumrotcount+coderot),obj.Nstate);   %to state
342                   coderot=mod(coderot,obj.M);
343                   obj.epoch=mod(obj.epoch+1,obj.Nepochs);              ...
                      % move to the next epoch
344               end
345        end   %encode
346
347  %----------------------------------------------------------
348        function testbits=CreateTestBitsClass(obj)
349              testbits=zeros(1,obj.M*obj.Nbits);
350              ctrr=0;
351              ind=1;
352              for q=0:1:obj.M-1
353                while(q≠0)
354                    r=rem(q,2);
355                    q=fix(q/2);
356                    testbits(ind*obj.Nbits-ctrr)=r;
357                    ctrr=mod(ctrr+1,obj.Nbits);
358                end
```

```matlab
359                     ctrr=0;
360                     ind=ind+1;
361                 end
362             end
363
364  %---------------------------------------------------------
365          function states=allstateClass(obj)
366              states=ones(obj.Nstate,obj.Nstate);
367              for ind1=1:1:obj.Nstate
368                  for ind2=1:1:obj.Nstate
369                      states(ind1,ind2)=ind1-1;
370                  end
371              end
372          end
373  %---------------------------------------------------------
374
375
376  function obj=setdecoderinfo(obj,doplot)
377    % set trellis information for decode using viterbi;
378    % trellis is computed using each poissble from-state(initial
379    % number of total rotation(sumrotcount)) and for each from-state
380    % using every possible combination of bits;
381    %algorithm
382    if (doplot)
383    figure(doplot)
384    clf;
385    set(gca,'ydir','reverse');
386    yplotsep = 1;
387    xplotsep = 1;
388    markersize = 15;
389    mulist = [0.15 0.45 0.60 0.75];
390    xcyclesym = ['A' 'B' 'C' 'D'];
391    end
392    %initialize trellis parameters
393    allstate=obj.allstateClass();
394    obj.trellisoutput=Inf(2,2,obj.Nstate,obj.Nstate,obj.Nepochs);
395    obj.inputbits=zeros(obj.Nbits,obj.Nstate,obj.Nstate,obj.Nepochs);
396    obj.metrics=zeros(obj.Nstate,1);
397    obj.nextmetrics=zeros(obj.Nstate,1);
398    obj.fromtostate=zeros(obj.Nstate,obj.Nstate,obj.Nepochs);
399    bj.testbits=obj.CreateTestBitsClass();  %genarate bit sequence ...
           [example, for M=4 bit sequence= 0 0; 0 1; 1 0; 1 1]
400    for epoch1=0:obj.Nepochs-1
401     if(doplot)
402     plottime = epoch1; plotx1 = plottime; plotx2 = plottime + xplotsep;
403     text(0.5*(plotx1 + plotx2),-0.2,xcyclesym(epoch1 +1)); hold on;
404     end
405     if (obj.doconstanttarget)
406      xcyclecount=0;
407     else
408      xcyclecount=epoch1;
409     end
410     tosym=obj.xcycle(xcyclecount+1);
411     for state=0:obj.Nstate-1
412      bitctr=0;
413      sumrotcountsave=state;
```

```matlab
414        randcolor=rand(1,3);
415        while(bitctr<(obj.M*obj.Nbits))
416         obj.sumrotcount=sumrotcountsave;
417         symno=obj.Bin2DecClass(obj.testbits(bitctr+1:bitctr+obj.Nbits));
418         symnosave=symno;
419         %%%%% Dr Sodha's approach:
420         if (obj.approach==3)
421          newpoint=obj.AddRot(symno+1,obj.sumrotcount+1);
422          coderot=obj.rotA(newpoint+1);
423          %update total number of previous rotation
424          sumrotcount=mod(obj.sumrotcount+coderot,obj.Nstate);
425 %        if (obj.sumrotcount+coderot>=obj.M)
426 %         sumrotcount=obj.sumrotcount+coderot-obj.M;            %to state
427 %        else
428 %         sumrotcount=obj.sumrotcount+coderot;                 %to state
429 %        end
430        end
431        %%%% End of Dr Sodha's approach:
432        %%% approach 2
433              if (obj.approach==2)
434                symno=mod(symno+obj.sumrotcount,obj.Nstate);
435                coderot=mod(symno+tosym,obj.Nstate);
436              end
437              %%%%% end of approach 2
438              %%%% approach 1
439              if (obj.approach==1)
440              symno=obj.cumrotoffset(symno+1,obj.sumrotcount+1);
441              coderot=obj.fromtodist(tosym+1,symno+1);
442              end
443              %%%%%end of approach 1
444              %%% used in both approach 1 and 2:
445              if (obj.approach ==1 || obj.approach ==2)
446               sumrotcount=mod(obj.multipliedfactor*...
447            (obj.sumrotcount+coderot),obj.Nstate);
448              end
449              coderot=mod(coderot,obj.M);
450              fromstate=state;
451              tostate=sumrotcount;
452              obj.trellisoutput(:,1,fromstate+1,tostate+1,epoch1+1)=...
453                        obj.const_pts(:,symnosave+1);
454              obj.trellisoutput(:,2,fromstate+1,tostate+1,epoch1+1)=...
455                  obj.const_pts(:,coderot+1);
456              obj.inputbits(:,fromstate+1,tostate+1,epoch1+1)=...
457                        obj.testbits(bitctr+1:bitctr+obj.Nbits)';
458              if (doplot)
459               fromy = sumrotcountsave*yplotsep;
460               toy =   sumrotcount*yplotsep;
461               plot([plotx1 plotx2],[fromy,toy],'color',randcolor);
462               plot(plotx1,fromy,'.',...
463             'markersize',markersize,'color','k');
464               plot(plotx2,fromy,'.','markersize',markersize,'color','k');
465               %str = sprintf('%d%d/%d%d',obj.testbits(bitctr+1),...
466            obj.testbits(bitctr+2),symnosave,coderot);
467                 if (obj.Nstate==4)
468                  str = sprintf('%d/%d',symnosave,coderot);
469                  mu = mulist(state +1);
```

```
470            textx = (1-mu)*plotx1 + mu*plotx2;
471            texty = (1-mu)*fromy + mu*toy;
472            htext = text(textx,texty,str);
473            tangle = -(180/pi)*atan((toy - fromy)/(plotx2 - plotx1));
474            % - sign since y coords reversed
475            set(htext,'VerticalAlignment','bottom');
476            set(htext,'rotation',tangle);
477          end
478          %hAxes = gca;      %Axis handle
479          %Changing 'LineStyle' to 'none'
480          yticks([0:1:obj.Nstate-1])
481          xticks([0 1])
482          xlim([0 1])
483          hold on
484          plot([0 0],[0 obj.Nstate-1],'w')
485          %ax1 = gca;
486          %yruler = ax1.YRuler;
487          %yruler.Axle.Visible = 'off';
488          %xruler = ax1.XRuler;
489          %xruler.Axle.Visible = 'off';
490        end % if doplots
491        bitctr=bitctr+obj.Nbits;
492       end %while
493      end% for state
494      obj.fromtostate(:,:,epoch1+1)=allstate;
495    end  % for epoch1
496    obj.prevstatearray=zeros(obj.Nstate,obj.viterbiwindowwidth);
497    obj.epoch_decode=0;
498    obj.bitqueue=zeros(obj.Nbits,obj.viterbiwindowwidth);
499    obj.bitqueuefront=0;
500    obj.bitqueueback=0;
501  end %setdecoderinfo
502  %------------------------------------------------------------
503  end
504 end
```

B.2   Julia Code

Julia Code for Signal Point Target Code

```
1  tic()
2  using SpecialFunctions
3  using PyPlot
4  using ToeplitzMatrices
5  using LaTeXStrings
6
7  function qf(x)
8      return .5*erfc(x/sqrt(2))
9  end
10
```

```julia
11  function mquamdemod(r,const_ptsf,const_axis)
12    indxx=const_axis[indmin(abs.(r[1]-const_axis))]
13    indyy=const_axis[indmin(abs.(r[2]-const_axis))]
14    d=[indxx,indyy]
15    lia=Int[ d == const_ptsf[:,i] for i=1:size(const_ptsf,2) ]
16    symbol=(findin(lia,1))[1]
17      return symbol-1
18    end
19
20  function bin2dec(k)
21    symbol=0
22    for i=1:1:length(k)
23      symbol=symbol+k[i]*2^(length(k)-i)
24    end
25    return symbol
26  end
27
28  function randombits(k)
29    return bits=Int.(bitrand(k))
30  end #randombit
31  function createtestbits(M,Nbits)
32    testbits=zeros(Int64,1,M*Nbits)
33    ctrr=0
34    i=1
35    for q in collect(0:1:M-1)
36      while (q!=0)
37        q,r=divrem(q,2)
38        testbits[i*Nbits-ctrr]=r
39        ctrr=mod(ctrr+1,Nbits)
40      end #while
41        ctrr=0
42        i=i+1
43    end #for
44    return testbits
45  end #createtestbits
46
47  doconstanttarget=false
48  doplot=false
49  M=64
50  Viterbiwindowwidth=10
51  doMquam=false
52  doshapecode=true
53  SNRrange=0:10
54  SNRrangetheo=0:15
55  GraphPsrange=10.0^-5
56  Nstate=M
57  Nepochs=M
58  Nbits=Int64(log2(M))
59  allstates=ones(Int64,Nstate,Nstate)
60  for i=1:1:Nstate
61    for j=1:1:Nstate
62    allstates[i,j]=i-1
63    end
64  end
65  testbits=createtestbits(M,Nbits)
66  const_axis=collect(-(2^(Nbits/2)-1):2:(2^(Nbits/2)-1))
```

```
67
68  const_pts=Int.([kron(Int.(ones(2^(Nbits/2))),const_axis) ...
        kron(const_axis,Int.(ones(2^(Nbits/2))))]')
69
70  println("------------------------------------------------------------------")
71  if (doshapecode)
72    #SNRrangeShapecode=0:10
73    if ¬(doconstanttarget)
74      xcycle=collect(0:1:(M-1))
75    else
76      xcycle=0
77    end
78
79  #const_pts
80
81    cumrotoffset=Int64.(Hankel(float.(collect(0:1:M-1)),float.([M-1; ...
          collect(0:1:M-2)])))
82    fromtodist=Int64.(Hankel(float.(collect(0:1:M-1)),float.([M-1; ...
          collect(0:1:M-2)])))
83    xcyclesym=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P']
84    trellisoutput=zeros(Int64,2,2,Nstate,Nstate,Nepochs)
85    inputbits=zeros(Int64,Nbits,Nstate,Nstate,Nepochs)
86    metrics=zeros(Nstate,1)
87    nextmetrics=zeros(Nstate,1)
88    prevstatearray=zeros(Int64,Nstate,Viterbiwindowwidth)
89    fromtostate=zeros(Int64,Nstate,Nstate,Nepochs)
90    bitqueue=zeros(Int64,Nbits,Viterbiwindowwidth)
91    bitqueuefront=0
92    bitqueueback=0
93
94  #------------------------------------------------------------------
95    mutable struct ShapeCodeFun
96    M::Int64
97    Viterbiwindowwidth::Int64
98    Nstate::Int64
99    Nepochs::Int64
100   Nbits::Int64
101   fromtodist
102   cumrotoffset
103   const_pts
104   xcycle
105   sumrotcount
106   epoch
107   doconstanttarget::Bool
108   trellisoutput
109   inputbits
110   fromtostate
111   metrics
112   nextmetrics
113   epoch_decode
114   prevstatearray
115   front
116   countbranches
117   beststate
118   bitqueue
119   bitqueuefront
```

```
120    bitqueueback
121    end
122  #------------------------------------------------------------------------
123
124    function setdecoderinfo(self::ShapeCodeFun,doplot,allstates,testbitsM)
125    if (doplot)
126      xcyclesym=['A' 'B' 'C' 'D' ];
127      yplotsep=1
128      xplotsep=1
129      markersize=15
130      mulist=[0.15; 0.45; 0.60; 0.75]
131      fig,ax=subplots()
132      ax[:axis]("off")
133      ax[:invert_yaxis]()
134    end
135    testbits=testbitsM
136
137    for epoch1=0:self.Nepochs-1
138      if (doplot)
139        plottime=epoch1
140        plotx1=plottime
141        plotx2=plottime+xplotsep
142
143
144        ax[:text](plotx2,-.3,"$(xcyclesym[epoch1+1])")
145      end
146      if (self.doconstanttarget)
147        xcyclecount=0
148      else
149        xcyclecount=epoch1
150      end
151
152      tosym=self.xcycle[xcyclecount+1]
153
154      for state=0:self.M-1
155        bitctr=0
156        sumrotcountsave=state
157        while (bitctr<(self.M*self.Nbits))
158          self.sumrotcount=sumrotcountsave
159          symno=bin2dec(testbits[bitctr+1:bitctr+self.Nbits])
160          #println(symno) function [symnosave,coderot]=encode(obj,bits)
161          symnosave=symno
162          symno=mod(symno+self.sumrotcount,self.M)
163          coderot=mod(symno+tosym,self.M)
164          sumrotcount=mod(self.sumrotcount+coderot,self.M)
165          fromstate=state
166          tostate=sumrotcount
167          self.trellisoutput[:,1,fromstate+1,tostate+1,epoch1+1]=...
168              self.const_pts[:,symnosave+1]
169          self.trellisoutput[:,2,fromstate+1,tostate+1,epoch1+1]=...
170              self.const_pts[:,coderot+1]
171          self.inputbits[:,fromstate+1,tostate+1,epoch1+1]=...
172              testbits[bitctr+1:bitctr+self.Nbits]
173          if (doplot)
174                     fromy=sumrotcountsave*yplotsep
175                     toy=sumrotcount*xplotsep
```

```
176
177                    ax[:plot]([plotx1; plotx2],[fromy; toy],"k")
178                    ax[:scatter](plotx1,fromy,c=:black)
179                    ax[:scatter](plotx2,fromy,c=:black)
180
181                    mu=mulist[state+1]
182
183                    textx=(1-mu)*plotx1+mu*plotx2
184                    texty=(1-mu)*fromy+mu*toy
185                    tangle=(-180/pi)*atan((toy - fromy)/(plotx2 - plotx1))
186                    ax[:text](textx,texty,"$(testbits[bitctr+1])...
187              $(testbits[bitctr+2])/$(symnosave)$(coderot)",...
188              rotation=tangle,horizontalalignment="left",...
189              verticalalignment="bottom", rotation_mode="anchor")
190          end   #doplot
191          #println(bitctr)
192          bitctr=bitctr+self.Nbits
193        end #while
194      end #end state
195      self.fromtostate[:,:,epoch1+1]=allstates
196    end #end epoch1
197    end #end setdecoderinfo
198  #-----------------------------------------------------------------------
199
200    function resetshapecoder(self::ShapeCodeFun)
201    self.epoch=0
202    self.sumrotcount=0
203    end #end resetshapecoder
204  #-----------------------------------------------------------------------
205    function resetshapedecoder(self::ShapeCodeFun)
206    self.epoch_decode=0
207    self.front=0
208    self.countbranches=0
209    self.bitqueuefront=0
210    self.bitqueueback=0
211    end  #end resetshapedecoder
212  #-----------------------------------------------------------------------
213
214    function encode(self::ShapeCodeFun,bits)
215    symno=bin2dec(bits)
216    symnosave=symno
217    symno=mod(symno+self.sumrotcount,self.M)
218    if (¬self.doconstanttarget)
219      xcyclecount=self.epoch
220    else
221      xcyclecount=0
222    end
223    tosym=self.xcycle[xcyclecount+1]
224    coderot=mod(symno+tosym,self.M)
225    self.sumrotcount=mod(self.sumrotcount+coderot,self.M)
226    self.epoch=mod(self.epoch+1,self.Nepochs)
227    return symnosave,coderot
228    end #end encode
229
230    function computemetric(r1,r2,output)
231    out11=output[1,1]
```

```
232    out12=output[1,2]
233    out21=output[2,1]
234    out22=output[2,2]
235    n1=(r1[1]-out11)^2+(r1[2]-out21)^2
236    n2=(r2[1]-out12)^2+(r2[2]-out22)^2
237    m=n1+n2
238    return m
239    end #end computemetric
240
241    function viterbishapedecode(self::ShapeCodeFun,r1,r2)
242    beststatemetric=Inf
243    for state=0:self.Nstate-1
244      nextmetricbest=Inf
245      for prevstate=self.fromtostate[:,state+1,self.epoch_decode+1]'
246        m=computemetric(r1,r2,...
247         self.trellisoutput[:,:,prevstate+1,state+1,self.epoch_decode+1])
248        nextmetric=self.metrics[prevstate+1]+m
249        if (nextmetric<nextmetricbest)
250          nextmetricbest=nextmetric
251          global prevstatebest=prevstate
252          statebest=state
253        end
254      end #for prevstate
255      if (nextmetricbest < beststatemetric)
256        beststatemetric = nextmetricbest
257        self.beststate = state
258      end
259      self.nextmetrics[state+1]=nextmetricbest
260      self.prevstatearray[state+1,self.front+1]=prevstatebest
261    end #for state
262    self.metrics=copy(self.nextmetrics)
263    self.countbranches=self.countbranches+1
264    decodeout=0
265    bitsout=[0;0]
266    if (self.countbranches≥self.Viterbiwindowwidth)
267      state=self.beststate
268      idx=self.front
269      epoch1=self.epoch_decode
270      prevstate=[]
271      lastepoch=[]
272      for i=1:self.Viterbiwindowwidth
273        prevstate=self.prevstatearray[state+1,idx+1]
274        idx=mod(idx-1,self.Viterbiwindowwidth)
275        lastepoch=epoch1
276        epoch1=mod(epoch1-1,self.Nepochs)
277        if (i!=self.Viterbiwindowwidth)
278          state=prevstate
279        end
280      end # i
281        bitsout=self.inputbits[:,prevstate+1,state+1,lastepoch+1]
282        decodeout=1
283    end #if
284    self.front=mod(self.front+1,self.Viterbiwindowwidth)
285    self.epoch_decode=mod(self.epoch_decode+1,self.Nepochs)
286    return decodeout,bitsout
287    end #shape decoder viterbi
```

```
288
289   # -----------------------------------------------------------------------
290
291     function viterbishapecodeflush(self::ShapeCodeFun)
292     bitsout=zeros(Int64,self.Nbits,self.Viterbiwindowwidth-1)
293     state=self.beststate
294     epoch1=mod(self.epoch_decode-1,self.Nepochs)
295     idx=mod(self.front-1,self.Viterbiwindowwidth)
296     for i=1:self.Viterbiwindowwidth-1
297       prevstate=self.prevstatearray[state+1,idx+1]
298       lastepoch=epoch1
299       bits1=self.inputbits[:,prevstate+1,state+1,lastepoch+1]
300       idx=mod(idx-1,self.Viterbiwindowwidth)
301       epoch1=mod(epoch1-1,self.Nepochs)
302       bitsout[:,self.Viterbiwindowwidth-1-i]=bits1
303       state=prevstate
304     end #for
305     end #viterbishapecodeflush
306   #-----------------------------------------------------------------------------
307
308     function bitqueuein(self::ShapeCodeFun,bits)
309     self.bitqueue[:,self.bitqueuefront+1]=bits
310     self.bitqueuefront = mod(self.bitqueuefront + 1, self.Viterbiwindowwidth)
311     end
312
313     function bitqueueout(self::ShapeCodeFun)
314     bits=self.bitqueue[:,self.bitqueueback+1]
315     self.bitqueueback = mod(self.bitqueueback + 1, self.Viterbiwindowwidth)
316     return bits
317     end
318
319   #create object of ...
          ShapeCodeFun-------------------------------------------------
320
321
322     a=ShapeCodeFun(M,Viterbiwindowwidth,Nstate,Nepochs,Nbits,
323                  fromtodist,cumrotoffset,const_pts,xcycle,0,
324                  0,doconstanttarget,trellisoutput,inputbits,fromtostate,
325           metrics,nextmetrics,0,prevstatearray,0,0,0,bitqueue,0,0)
326
327
328     setdecoderinfo(a,doplot,allstates,testbits)
329     resetshapecoder(a)
330     resetshapedecoder(a)
331
332   end #doshapecode
333
334   nerrortocount=100
335
336   Eb=1
337   R=1
338   Es=Nbits*Eb*R
339   Esscale=sqrt(3*Es/(2*(M-1)))
340   Rcoded=.5
341   Escoded=sqrt(Nbits*Eb*Rcoded*3/(2*(M-1)))
342   theorprobQPSK=[]
```

```julia
343  theorprobBPSK=[]
344  theoprobMquam=[]
345  for SNR in SNRrangetheo
346    println("snr=$(SNR)")
347    N0=Eb*10^(-SNR/10)
348    EbN0=Eb/N0
349    push!(theorprobQPSK,(1 - (1-qf(sqrt(2*EbN0)))^2))
350    push!(theorprobBPSK,(qf(sqrt(2*EbN0))))
351    push!(theoprobMquam,...
352      (1-(1-2*(1-sqrt(1/M))*qf(sqrt((3/(M-1))*Nbits*EbN0)))^2))
353  end
354
355  r=[0.0 0.0]
356  if (doMquam)
357    snrctr=0
358    errctr=[]
359    nsyms=[]
360    for SNR in SNRrange
361      println("snr=$(SNR)")
362      snrctr=snrctr+1
363      N0=Eb*10^(-SNR/10)
364      EbN0=Eb/N0
365      #println("N0=$(N0) EbN0=$(EbN0)")
366      sigma2=N0/2
367      sigma=sqrt(sigma2)
368      push!(errctr,0)
369      push!(nsyms,0)
370
371      while(errctr[snrctr]<nerrortocount)
372        nsyms[snrctr]=nsyms[snrctr]+1
373        bits=randombits(Nbits)
374        symno=bin2dec(bits)
375        constpts=Esscale*const_pts[:,symno+1]
376        noise=sigma*randn(1,2)
377        r[1]=constpts[1]+noise[1]
378        r[2]=constpts[2]+noise[2]
379        decodesymno=mquamdemod(r/Esscale,const_pts,const_axis)
380        #println("decodesymno=$(decodesymno) symno=$(symno)")
381
382        if (decodesymno != symno)
383          errctr[snrctr]=errctr[snrctr]+1
384          println("SNR=$(SNR):uncoded error no=$(errctr[snrctr])")
385          #println("decodesymno=$(decodesymno) symno=$(symno)")
386        end
387       if (errctr[snrctr]!=0 && errctr[snrctr]/nsyms[snrctr]≤GraphPsrange)
388          break
389        end   #end break while
390      end #while errctr
391      if (errctr[snrctr]/nsyms[snrctr]≤GraphPsrange)
392        break
393      end #end brek for
394    end #for snr
395  end #if(doqpsk)
396
397  if (doshapecode)
398      errctr_coded=[]
```

```
399      errctr_bits_coded=[]
400      nsyms_coded=[]
401      snrctr=0
402      for SNR in SNRrange
403        println("SNRShapeCode=$(SNR)")
404        snrctr=snrctr+1
405        N0=Eb*10^(-SNR/10)
406        EbN0=Eb/N0
407        sigma2=N0/2
408        sigma=sqrt(sigma2)
409        push!(errctr_coded,0)
410        push!(errctr_bits_coded,0)
411        push!(nsyms_coded,0)
412        while(errctr_coded[snrctr]<nerrortocount)
413          nsyms_coded[snrctr]=nsyms_coded[snrctr]+1
414          bits1=randombits(Nbits)
415          sym1,sym2=encode(a,bits1)
416          bitqueuein(a,bits1)
417          s1=Escoded*a.const_pts[:,sym1+1]
418          s2=Escoded*a.const_pts[:,sym2+1]
419          noise1=sigma*randn(2,1)
420          noise2=sigma*randn(2,1)
421          r1=s1+noise1
422          r2=s2+noise2
423          decodeout, bitsout=viterbishapedecode(a,r1/Escoded,r2/Escoded)
424          if (decodeout==1)
425            inbits=bitqueueout(a)
426            if (inbits!=bitsout)
427              println("SNR=$(SNR):coded error no=$(errctr_coded[snrctr])")
428            end
429            println("inbits=$(inbits) bitsout=$(bitsout)")
430            errctr_coded[snrctr]=errctr_coded[snrctr]....
431          +Int64(any(inbits!=bitsout))
432            errctr_bits_coded[snrctr] = errctr_bits_coded[snrctr] + ...
                   sum(Int64.(inbits .!= bitsout))
433          end #decodeout
434          if (errctr_coded[snrctr]!=0 && ...
                  errctr_coded[snrctr]/nsyms_coded[snrctr]≤GraphPsrange)
435            break
436          end   #end break while
437      end #while error count
438      if (errctr_coded[snrctr]/nsyms_coded[snrctr]≤GraphPsrange)
439        break
440      end #end break for
441
442    end #for SNRrangeShapecode
443  end #doshapecode
444  legendstr=[]
445  push!(legendstr,"QPSK(th)")
446  push!(legendstr,"BPSK(th)")
447  push!(legendstr,"$(M)QUAM(th)")
448
449  if (doMquam)
450    push!(legendstr,"$(M)Quam(ex)")
451  end
452  if (doshapecode)
```

```
453    push!(legendstr,"Shapecoded$(M)Quam(sym)")
454    push!(legendstr,"Shapecoded$(M)Quam(bit)")
455  end
456  len=length(SNRrange)
457
458  fig1,ax1=subplots()
459  ax1[:semilogy](collect(0:1:length(theorprobQPSK)-1),theorprobQPSK,"g:")
460  ax1[:semilogy](collect(0:1:length(theorprobBPSK)-1),theorprobBPSK,"b:")
461  ax1[:semilogy](collect(0:1:length(theoprobMquam)-1),theoprobMquam,"k:")
462
463  if (doMquam)
464    ax1[:semilogy](collect(0:1:length(errctr)-1),errctr./(nsyms),"r:")
465  end
466
467  if (doshapecode)
468    ax1[:semilogy](collect(0:1:length(errctr_coded)-1),...
469      errctr_coded./nsyms_coded,"c")
470    ax1[:semilogy](collect(0:1:length(errctr_coded)-1),...
471      errctr_bits_coded./(Nbits*nsyms_coded),"c:")
472  end
473
474  ax1[:legend](legendstr)
475  ax1[:set_ylim]([GraphPsrange,10.0^-0])
476  ax1[:grid](color="k", linestyle="--")
477  xlabel(L"(SNR)_{db}")
478  ylabel(L"P_s" )
479  toc()
480  Ps=-1*Int(log10(GraphPsrange))
481  savefig("BER$(M)QUAMSCEbSNR$(SNRrange[end])Ps$(Ps)rate12.pdf",dpi=1500)
```

Code for Rate $2/3$ (SIE)

```
1     trellisoutput=zeros(Int64,2,3,M,Nstate,Nstate,Nepochs)
2     inputbits=zeros(Int64,Nbits,2,M,Nstate,Nstate,Nepochs)
3     metrics=zeros(Nstate,1)
4     nextmetrics=zeros(Nstate,1)
5     prevstatearray=zeros(Int64,Nstate,Viterbiwindowwidth)
6     fromtostate=zeros(Int64,Nstate,Nstate,Nepochs)
7     bitqueue=zeros(Int64,Nbits,2,Viterbiwindowwidth)
8     bitqueuefront=0
9     bitqueueback=0
10
11    bestsymbolarray=zeros(Int64,Nstate,Viterbiwindowwidth)
12
13
14
15    function setdecoderinfo(self::ShapeCodeFun,doplot,allstates,testbitsM)
16    if (doplot)
17      xcyclesym=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' ...
             'O' 'P']
18      yplotsep=1
19      xplotsep=1
```

```
20      markersize=15
21      mulist=[.062 .12 .18 .25 .31 .37 .43 .5 .562 .6245 .687 .75 .81 .87 ...
            .93 .98]
22      fig,ax=subplots()
23      ax[:axis]("off")
24      ax[:invert_yaxis]()
25    end
26    testbits=testbitsM
27
28    for epoch1=0:self.Nepochs-1
29      if (doplot)
30        plottime=epoch1
31        plotx1=plottime
32        plotx2=plottime+xplotsep
33
34
35        ax[:text](plotx2,-.3,"$(xcyclesym[epoch1+1])")
36      end
37      if (self.doconstanttarget)
38        xcyclecount=0
39      else
40        xcyclecount=epoch1
41      end
42
43      tosym=self.xcycle[xcyclecount+1]
44
45      for state=0:self.M-1
46        bitctr=0
47        sumrotcountsave=state
48        while (bitctr<(self.M*self.Nbits))
49          bitssymbol1=testbits[bitctr+1:bitctr+self.Nbits]
50          symno=bin2dec(bitssymbol1)
51          symnosave=symno
52          self.sumrotcount=sumrotcountsave
53          bitctr1=0
54          while(bitctr1<(self.M*self.Nbits))
55            bitssymbol2=testbits[bitctr1+1:bitctr1+self.Nbits]
56
57            symno2=bin2dec(bitssymbol2)      #conver secend symbol to decimel
58
59            symnosave2=symno2                                      #save ...
                to a memory
60
61            state_int1=mod(symno+self.sumrotcount,self.M) ...
                                 #intermediate state after taking ...
                previosu number of rotation into account
62            #coderot1=mod(state_int1+tosym,self.M)
63            #state_final1=mod(self.sumrotcount+state_int1,self.M)
64
65            #tosym=self.xcycle[xcyclecount+1]
66            state_int2=mod(state_int1+symno2,self.M)
67            #coderot2=mod(state_int2+tosym,self.M)
68            #state_final2=mod(state_final1+state_int2,self.M)
69
70
71
```

```
72              #tosym=self.xcycle[xcyclecount+1]
73              coderot=mod(state_int2+tosym,self.M)
74
75              #state_int3=mod(state_final2+coderot,self.M)
76              #state_interfinal=mod(state_int3+tosym,self.M)
77              sumrotcount=mod(self.sumrotcount+coderot,self.M)
78
79
80              fromstate=state
81              tostate=sumrotcount
82              #println("symn1=$(symnosave)  sym2=$(symnosave2)  ...
                    fromstate=$(fromstate) tostate=$(sumrotcount) ...
                    epoch=$(epoch1) ")
83              self.trellisoutput[:,1,symnosave+1,fromstate+1,tostate+1, ...
                    epoch1+1]=self.const_pts[:,symnosave+1]
84
85              self.trellisoutput[:,2,symnosave+1,fromstate+1,tostate+1, ...
                    epoch1+1]=self.const_pts[:,symnosave2+1]
86              self.trellisoutput[:,3,symnosave+1,fromstate+1,tostate+1, ...
                    epoch1+1]=self.const_pts[:,coderot+1]
87
88              #println(bitssymbol1)
89              self.inputbits[:,1,symnosave+1,fromstate+1,tostate+1,epoch1+1] ...
                    =bitssymbol1
90              self.inputbits[:,2,symnosave+1,fromstate+1,tostate+1,epoch1+1 ...
                    ]=bitssymbol2
91              if (doplot)
92                      fromy=sumrotcountsave*yplotsep
93                      toy=sumrotcount*xplotsep
94
95                      ax[:plot]([plotx1; plotx2],[fromy; toy],"k")
96                      ax[:scatter](plotx1,fromy,c=:black)
97                      ax[:scatter](plotx2,fromy,c=:black)
98
99                      mu=mulist[state+1]
100                     textx=(1-mu)*plotx1+mu*plotx2
101                     texty=(641-mu)*fromy+mu*toy
102                     tangle=(-180/pi)*atan((toy - fromy)/(plotx2 - plotx1))
103                   ax[:text](textx,texty,"$(testbits[bitctr+1]) ...
                          $(testbits[bitctr+2])$(testbits[bitctr+3]) ...
                          $(testbits[bitctr+4])/$ (symnosave)$(coderot)", ...
                          rotation=tangle,horizontalalignment="left", ...
                          verticalalignment="bottom", rotation_mode="anchor")
104             end   #doplot
105                  #println(bitctr)
106             bitctr1=bitctr1+self.Nbits
107         end #while bitctr1
108         bitctr=bitctr+self.Nbits
109     end #while
110   end #end state
111   self.fromtostate[:,:,epoch1+1]=allstates
112 end #end epoch1
113 end #end setdecoderinfo
114 #-----------------------------------------------------------
115
116 function encode(self::ShapeCodeFun,bits1,bits2)
```

```
117    symno1=bin2dec(bits1)
118    symnosave1=symno1
119    symno2=bin2dec(bits2)
120    symnosave2=symno2
121    if (¬self.doconstanttarget)
122      xcyclecount=self.epoch
123    else
124      xcyclecount=0
125    end
126
127
128    tosym=self.xcycle[xcyclecount+1]
129
130    state_int1=mod(symno1+self.sumrotcount,self.M)
131    #coderot1=mod(state_int1+tosym,self.M)
132    #state_final1=mod(state_int1+self.sumrotcount,self.M)
133
134    state_int2=mod(symno2+state_int1,self.M)
135    #coderot2=mod(state_int2+tosym,self.M)
136    #state_final2=mod(state_int2+state_final1,self.M)
137
138    coderot=mod(state_int2+tosym,self.M)
139    #state_int3=mod(state_final2+coderot,self.M)
140    #state_interfinal=mod(state_int3+tosym,self.M)
141    self.sumrotcount=mod(coderot+self.sumrotcount,self.M)
142
143    self.epoch=mod(self.epoch+1,self.Nepochs)
144    return symnosave1,symnosave2,coderot
145    end #end encode
146
147    function computemetric(r1,r2,r3,output)
148    out11=output[1,1]
149    out12=output[1,2]
150    out13=output[1,3]
151    out21=output[2,1]
152    out22=output[2,2]
153    out23=output[2,3]
154    n1=(r1[1]-out11)^2+(r1[2]-out21)^2
155    n2=(r2[1]-out12)^2+(r2[2]-out22)^2
156    n3=(r3[1]-out13)^2+(r3[2]-out23)^2
157    m=n1+n2+n3
158    return m
159    end #end computemetric
160
161    function viterbishapedecode(self::ShapeCodeFun,r1,r2,r3)
162    beststatemetric=Inf
163    for state=0:self.Nstate-1
164      nextmetricbest=Inf
165      for prevstate=self.fromtostate[:,state+1,self.epoch_decode+1]'
166        for symbol=0:self.Nstate-1
167          m=computemetric(r1,r2,r3,self.trellisoutput ...
                  [:,:,symbol+1,prevstate+1,state+1,self.epoch_decode+1])
168          nextmetric=self.metrics[prevstate+1]+m
169          if (nextmetric<nextmetricbest)
170            nextmetricbest=nextmetric
171            global prevstatebest=prevstate
```

```
172           global symbolbest=symbol
173           statebest=state
174         end
175       end #symbol
176     end #for prevstate
177     if (nextmetricbest < beststatemetric)
178         beststatemetric = nextmetricbest
179         self.beststate = state
180     end
181     self.nextmetrics[state+1]=nextmetricbest
182     self.prevstatearray[state+1,self.front+1]=prevstatebest
183     self.bestsymbolarray[state+1,self.front+1]=symbolbest
184   end #for state
185   self.metrics=copy(self.nextmetrics)
186   self.countbranches=self.countbranches+1
187   decodeout=0
188   bitsout1=[0;0]
189   bitsout2=[0;0]
190   if (self.countbranches≥self.Viterbiwindowwidth)
191     state=self.beststate
192     idx=self.front
193     epoch1=self.epoch_decode
194     prevstate=[]
195     lastepoch=[]
196     prevstatbestsymbol=[]
197     for i=1:self.Viterbiwindowwidth
198       prevstate=self.prevstatearray[state+1,idx+1]
199
200       prevstatbestsymbol=self.bestsymbolarray[state+1,idx+1]
201       idx=mod(idx-1,self.Viterbiwindowwidth)
202       lastepoch=epoch1
203       epoch1=mod(epoch1-1,self.Nepochs)
204       if (i!=self.Viterbiwindowwidth)
205         state=prevstate
206       end
207     end # i
208       bitsout1=self.inputbits[:,1,prevstatbestsymbol+1, ...
209             prevstate+1,state+1,lastepoch+1]
208       bitsout2=self.inputbits[:,2,prevstatbestsymbol+1, ...
             prevstate+1,state+1,lastepoch+1]
210       decodeout=1
211   end #if
212   self.front=mod(self.front+1,self.Viterbiwindowwidth)
213   self.epoch_decode=mod(self.epoch_decode+1,self.Nepochs)
214   return decodeout,bitsout1,bitsout2
215   end #shape decoder viterbi
216
217 # ----------------------------------------------------------------
```

Code for Rate $3/4$ (SIE)

```
1
```

```
2    trellisoutput=zeros(Int64,2,n,M,M,Nstate,Nstate,Nepochs)
3    inputbits=zeros(Int64,Nbits,k,M,M,Nstate,Nstate,Nepochs)
4    metrics=zeros(Nstate,1)
5    nextmetrics=zeros(Nstate,1)
6    prevstatearray=zeros(Int64,Nstate,Viterbiwindowwidth)
7    fromtostate=zeros(Int64,Nstate,Nstate,Nepochs)
8    bitqueue=zeros(Int64,Nbits,k,Viterbiwindowwidth)
9    bitqueuefront=0
10   bitqueueback=0
11
12   bestsymbol1array=zeros(Int64,Nstate,Viterbiwindowwidth)
13   bestsymbol2array=zeros(Int64,Nstate,Viterbiwindowwidth)
14
15
16
17   function setdecoderinfo(self::ShapeCodeFun,doplot,allstates,testbitsM)
18   if (doplot)
19
20   end
21   testbits=testbitsM
22
23   for epoch1=0:self.Nepochs-1
24     if (doplot)
25
26     end
27     if (self.doconstanttarget)
28       xcyclecount=0
29     else
30       xcyclecount=epoch1
31     end
32
33     tosym=self.xcycle[xcyclecount+1]
34
35     for state=0:self.Nstate-1
36       bitctr=0
37       sumrotcountsave=state
38       while (bitctr<(self.M*self.Nbits))
39         bitssymbol1=testbits[bitctr+1:bitctr+self.Nbits]
40         symno=bin2dec(bitssymbol1)
41         symnosave=symno
42         self.sumrotcount=sumrotcountsave
43         bitctr1=0
44         while (bitctr1<(self.M*self.Nbits))
45           bitssymbol2=testbits[bitctr1+1:bitctr1+self.Nbits]
46
47           symno2=bin2dec(bitssymbol2)       #conver secend symbol to decimel
48
49           symnosave2=symno2                                    #save ...
                 to a memory
50
51
52           bitctr2=0
53           while (bitctr2<(self.M*self.Nbits))
54             bitssymbol3=testbits[bitctr2+1:bitctr2+self.Nbits]
55             symno3=bin2dec(bitssymbol3)
56             symnosave3=symno3
```

```julia
57                #println(symno3)
58
59                An=mod(symno+self.sumrotcount,self.M)                          ...
                      #intermediate state after taking previosu number of ...
                      rotation into account
60                An1=mod(An+symno2,self.M)
61                An2=mod(An1+symno3,self.M)
62                coderot=(An2+tosym)
63               sumrotcount=mod(self.sumrotcount+coderot,self.Nstate)
64
65
66                fromstate=state
67                tostate=sumrotcount
68                #println("symn1=$(symnosave)  sym2=$(symnosave2) ...
                      sym3=$(symnosave3) parity=$(coderot) ...
                      fromstate=$(fromstate) tostate=$(sumrotcount) ...
                      epoch=$(epoch1) ")
69
70                self.trellisoutput[:,1,symnosave+1,symnosave2+1,fromstate+1, ...
                      tostate+1,epoch1+1]=self.const_pts[:,symnosave+1]
71                self.trellisoutput[:,2,symnosave+1,symnosave2+1,fromstate+1, ...
                      tostate+1,epoch1+1]=self.const_pts[:,symnosave2+1]
72                self.trellisoutput[:,3,symnosave+1,symnosave2+1,fromstate+1, ...
                      tostate+1,epoch1+1]=self.const_pts[:,symnosave3+1]
73                self.trellisoutput[:,4,symnosave+1,symnosave2+1,fromstate+1, ...
                      tostate+1,epoch1+1]=self.const_pts[:,coderot+1]
74
75                #println(bitssymbol1)
76                self.inputbits[:,1,symnosave+1,symnosave2+1,fromstate+1, ...
                      tostate+1,epoch1+1]=bitssymbol1
77                self.inputbits[:,2,symnosave+1,symnosave2+1,fromstate+1, ...
                      tostate+1,epoch1+1]=bitssymbol2
78                self.inputbits[:,3,symnosave+1,symnosave2+1,fromstate+1, ...
                      tostate+1,epoch1+1]=bitssymbol3
79
80
81                    #println(bitctr)
82            bitctr2=bitctr2+self.Nbits
83            end #while bitctr2
84              bitctr1=bitctr1+self.Nbits
85            end #while bitctr1
86            bitctr=bitctr+self.Nbits
87        end #while
88      end #end state
89      self.fromtostate[:,:,epoch1+1]=allstates
90    end #end epoch1
91    end #end setdecoderinfo
92
93  #-------------------------------------------------------------
94
95    function encode(self::ShapeCodeFun,bits1,bits2,bits3)
96    symno1=bin2dec(bits1)
97    symnosave1=symno1
98    symno2=bin2dec(bits2)
99    symnosave2=symno2
100   symno3=bin2dec(bits3)
```

```
101    symnosave3=symno3
102    if (¬self.doconstanttarget)
103      xcyclecount=self.epoch
104    else
105      xcyclecount=0
106    end
107
108
109    tosym=self.xcycle[xcyclecount+1]
110
111    An=mod(symno1+self.sumrotcount,self.M)
112
113    An1=mod(An+symno2,self.M)
114
115    An2=mod(symno3+An1,self.M)
116
117
118
119
120    coderot=mod(An2+tosym,self.M)
121
122    self.sumrotcount=mod(coderot+self.sumrotcount,self.Nstate)
123
124    self.epoch=mod(self.epoch+1,self.Nepochs)
125    return symnosave1,symnosave2,symnosave3,coderot
126    end #end encode
127
128    function computemetric(r1,r2,r3,r4,output)
129    out11=output[1,1]
130    out12=output[1,2]
131    out13=output[1,3]
132    out14=output[1,4]
133    out21=output[2,1]
134    out22=output[2,2]
135    out23=output[2,3]
136    out24=output[2,4]
137    n1=(r1[1]-out11)^2+(r1[2]-out21)^2
138    n2=(r2[1]-out12)^2+(r2[2]-out22)^2
139    n3=(r3[1]-out13)^2+(r3[2]-out23)^2
140    n4=(r4[1]-out14)^2+(r4[2]-out24)^2
141    m=n1+n2+n3+n4
142    return m
143    end #end computemetric
144
145    function viterbishapedecode(self::ShapeCodeFun,r1,r2,r3,r4)
146    beststatemetric=Inf
147    for state=0:self.Nstate-1
148      nextmetricbest=Inf
149      for prevstate=self.fromtostate[:,state+1,self.epoch_decode+1]'
150        for symbol1=0:self.M-1
151          for symbol2=0:self.M-1
152            m=computemetric(r1,r2,r3,r4,self.trellisoutput[:,:,symbol1+1, ...
                   symbol2+1,prevstate+1,state+1, self.epoch_decode+1])
153            nextmetric=self.metrics[prevstate+1]+m
154            if (nextmetric<nextmetricbest)
155              nextmetricbest=nextmetric
```

```
156            global prevstatebest=prevstate
157            global symbolbest1=symbol1
158            global symbolbest2=symbol2
159            statebest=state
160          end
161        end #symbol2
162      end #symbol1
163    end #for prevstate
164    if (nextmetricbest < beststatemetric)
165        beststatemetric = nextmetricbest
166        self.beststate = state
167    end
168    self.nextmetrics[state+1]=nextmetricbest
169    self.prevstatearray[state+1,self.front+1]=prevstatebest
170    self.bestsymbol1array[state+1,self.front+1]=symbolbest1
171    self.bestsymbol2array[state+1,self.front+1]=symbolbest2
172  end #for state
173  self.metrics=copy(self.nextmetrics)
174  self.countbranches=self.countbranches+1
175  decodeout=0
176  bitsout1=[0;0]
177  bitsout2=[0;0]
178  bitsout3=[0;0]
179  if (self.countbranches≥self.Viterbiwindowwidth)
180    state=self.beststate
181    idx=self.front
182    epoch1=self.epoch_decode
183    prevstate=[]
184    lastepoch=[]
185    prevstatbestsymbol1=[]
186    prevstatbestsymbol2=[]
187    for i=1:self.Viterbiwindowwidth
188      prevstate=self.prevstatearray[state+1,idx+1]
189
190      prevstatbestsymbol1=self.bestsymbol1array[state+1,idx+1]
191      prevstatbestsymbol2=self.bestsymbol2array[state+1,idx+1]
192      idx=mod(idx-1,self.Viterbiwindowwidth)
193      lastepoch=epoch1
194      epoch1=mod(epoch1-1,self.Nepochs)
195      if (i!=self.Viterbiwindowwidth)
196        state=prevstate
197      end
198    end # i
199      bitsout1=self.inputbits[:,1,prevstatbestsymbol1+1, ...
            prevstatbestsymbol2+1,prevstate+1, state+1,lastepoch+1]
200      bitsout2=self.inputbits[:,2,prevstatbestsymbol1+1, ...
            prevstatbestsymbol2+1,prevstate+1, state+1,lastepoch+1]
201      bitsout3=self.inputbits[:,3,prevstatbestsymbol1+1, ...
            prevstatbestsymbol2+1,prevstate+1, state+1,lastepoch+1]
202      decodeout=1
203  end #if
204  self.front=mod(self.front+1,self.Viterbiwindowwidth)
205  self.epoch_decode=mod(self.epoch_decode+1,self.Nepochs)
206  return decodeout,bitsout1,bitsout2,bitsout3
207  end #shape decoder viterbi
```

Code for Rate $2/3$ (PPE)

```
1
2    trellisoutput=zeros(Int64,2,3,M,Nstate,Nstate,Nepochs)
3    inputbits=zeros(Int64,Nbits,2,M,Nstate,Nstate,Nepochs)
4    metrics=zeros(Nstate,1)
5    nextmetrics=zeros(Nstate,1)
6    prevstatearray=zeros(Int64,Nstate,Viterbiwindowwidth)
7    fromtostate=zeros(Int64,Nstate,Nstate,Nepochs)
8    bitqueue=zeros(Int64,Nbits,2,Viterbiwindowwidth)
9    bitqueuefront=0
10   bitqueueback=0
11
12   bestsymbolarray=zeros(Int64,Nstate,Viterbiwindowwidth)
13
14
15   function setdecoderinfo(self::ShapeCodeFun,doplot,allstates,testbitsM)
16   if (doplot)
17     xcyclesym=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' ...
            'O' 'P']
18     yplotsep=1
19     xplotsep=1
20     markersize=15
21     mulist=[.062 .12 .18 .25 .31 .37 .43 .5 .562 .6245 .687 .75 .81 .87 ...
            .93 .98]
22     fig,ax=subplots()
23     ax[:axis]("off")
24     ax[:invert_yaxis]()
25   end
26   testbits=testbitsM
27
28   for epoch1=0:self.Nepochs-1
29     if (doplot)
30       plottime=epoch1
31       plotx1=plottime
32       plotx2=plottime+xplotsep
33
34
35       ax[:text](plotx2,-.3,"$(xcyclesym[epoch1+1])")
36     end
37     if (self.doconstanttarget)
38       xcyclecount=0
39     else
40       xcyclecount=epoch1
41     end
42
43     tosym=self.xcycle[xcyclecount+1]
44
45     for state=0:self.M-1
46       bitctr=0
47       sumrotcountsave=state
48       while (bitctr<(self.M*self.Nbits))
49         bitssymbol1=testbits[bitctr+1:bitctr+self.Nbits]
50         symno=bin2dec(bitssymbol1)
51         symnosave=symno
```

```
52              self.sumrotcount=sumrotcountsave
53              bitctr1=0
54              while(bitctr1<(self.M*self.Nbits))
55                bitssymbol2=testbits[bitctr1+1:bitctr1+self.Nbits]
56
57                symno2=bin2dec(bitssymbol2)       #conver secend symbol to decimel
58
59                symnosave2=symno2                                    #save ...
                      to a memory
60
61                state_int1=mod(symno+self.sumrotcount,self.M) ...
                                       #intermediate state after taking ...
                      previosu number of rotation into account
62                coderot1=mod(state_int1+tosym,self.M)
63                state_final1=mod(self.sumrotcount+coderot1,self.M)
64
65                #tosym=self.xcycle[xcyclecount+1]
66                state_int2=mod(state_final1+symno2,self.M)
67                coderot2=mod(state_int2+tosym,self.M)
68                state_final2=mod(state_final1+coderot2,self.M)
69
70
71
72                #tosym=self.xcycle[xcyclecount+1]
73                coderot=mod(coderot1+coderot2,self.M)
74
75                state_int3=mod(state_final2+coderot,self.M)
76                state_interfinal=mod(state_int3+tosym,self.M)
77                sumrotcount=mod(state_final2+state_interfinal,self.M)
78
79
80                fromstate=state
81                tostate=sumrotcount
82                #println("symn1=$(symnosave)  sym2=$(symnosave2)  ...
                      fromstate=$(fromstate) tostate=$(sumrotcount) ...
                      epoch=$(epoch1) ")
83                self.trellisoutput[:,1,symnosave+1,fromstate+1,tostate+1 ...
                      ,epoch1+1]=self.const_pts[:,symnosave+1]
84
85                self.trellisoutput[:,2,symnosave+1,fromstate+1,tostate+1, ...
                      epoch1+1]=self.const_pts[:,symnosave2+1]
86                self.trellisoutput[:,3,symnosave+1,fromstate+1,tostate+1, ...
                      epoch1+1]=self.const_pts[:,coderot+1]
87
88                #println(bitssymbol1)
89                self.inputbits[:,1,symnosave+1,fromstate+1,tostate+1, ...
                      epoch1+1]=bitssymbol1
90                self.inputbits[:,2,symnosave+1,fromstate+1,tostate+1, ...
                      epoch1+1]=bitssymbol2
91                if (doplot)
92                          fromy=sumrotcountsave*yplotsep
93                          toy=sumrotcount*xplotsep
94
95                          ax[:plot]([plotx1; plotx2],[fromy; toy],"k")
96                          ax[:scatter](plotx1,fromy,c=:black)
97                          ax[:scatter](plotx2,fromy,c=:black)
```

```
98
99                          mu=mulist[state+1]
100                         textx=(1-mu)*plotx1+mu*plotx2
101                         texty=(641-mu)*fromy+mu*toy
102                         tangle=(-180/pi)*atan((toy - fromy)/(plotx2 - plotx1))
103                       ax[:text](textx,texty,"$(testbits[bitctr+1]) ...
                              $(testbits[bitctr+2])$ (testbits[bitctr+3])$ ...
                              (testbits[bitctr+4])/$(symnosave)$(coderot)", ...
                              rotation=tangle,horizontalalignment="left", ...
                              verticalalignment="bottom",  rotation_mode="anchor")
104               end   #doplot
105                   #println(bitctr)
106               bitctr1=bitctr1+self.Nbits
107           end #while bitctr1
108           bitctr=bitctr+self.Nbits
109       end #while
110     end #end state
111     self.fromtostate[:,:,epoch1+1]=allstates
112   end #end epoch1
113   end #end setdecoderinfo
114
115   function encode(self::ShapeCodeFun,bits1,bits2)
116   symno1=bin2dec(bits1)
117   symnosave1=symno1
118   symno2=bin2dec(bits2)
119   symnosave2=symno2
120   if (¬self.doconstanttarget)
121     xcyclecount=self.epoch
122   else
123     xcyclecount=0
124   end
125
126
127   tosym=self.xcycle[xcyclecount+1]
128
129   state_int1=mod(symno1+self.sumrotcount,self.M)
130   coderot1=mod(state_int1+tosym,self.M)
131   state_final1=mod(coderot1+self.sumrotcount,self.M)
132
133   state_int2=mod(symno2+state_final1,self.M)
134   coderot2=mod(state_int2+tosym,self.M)
135   state_final2=mod(coderot2+state_final1,self.M)
136
137   coderot=mod(coderot1+coderot2,self.M)
138   state_int3=mod(state_final2+coderot,self.M)
139   state_interfinal=mod(state_int3+tosym,self.M)
140   self.sumrotcount=mod(state_interfinal+state_final2,self.M)
141
142   self.epoch=mod(self.epoch+1,self.Nepochs)
143   return symnosave1,symnosave2,coderot
144   end #end encode
145
146   function computemetric(r1,r2,r3,output)
147   out11=output[1,1]
148   out12=output[1,2]
149   out13=output[1,3]
```

```
150    out21=output[2,1]
151    out22=output[2,2]
152    out23=output[2,3]
153    n1=(r1[1]-out11)^2+(r1[2]-out21)^2
154    n2=(r2[1]-out12)^2+(r2[2]-out22)^2
155    n3=(r3[1]-out13)^2+(r3[2]-out23)^2
156    m=n1+n2+n3
157    return m
158    end #end computemetric
159
160    function viterbishapedecode(self::ShapeCodeFun,r1,r2,r3)
161    beststatemetric=Inf
162    for state=0:self.Nstate-1
163      nextmetricbest=Inf
164      for prevstate=self.fromtostate[:,state+1,self.epoch_decode+1]'
165        for symbol=0:self.Nstate-1
166          m=computemetric(r1,r2,r3,self.trellisoutput[:,:,symbol+1, ...
                 prevstate+1,state+1,self.epoch_decode+1])
167          nextmetric=self.metrics[prevstate+1]+m
168          if (nextmetric<nextmetricbest)
169            nextmetricbest=nextmetric
170            global prevstatebest=prevstate
171            global symbolbest=symbol
172            statebest=state
173          end
174        end #symbol
175      end #for prevstate
176      if (nextmetricbest < beststatemetric)
177        beststatemetric = nextmetricbest
178        self.beststate = state
179      end
180      self.nextmetrics[state+1]=nextmetricbest
181      self.prevstatearray[state+1,self.front+1]=prevstatebest
182      self.bestsymbolarray[state+1,self.front+1]=symbolbest
183    end #for state
184    self.metrics=copy(self.nextmetrics)
185    self.countbranches=self.countbranches+1
186    decodeout=0
187    bitsout1=[0;0]
188    bitsout2=[0;0]
189    if (self.countbranches≥self.Viterbiwindowwidth)
190      state=self.beststate
191      idx=self.front
192      epoch1=self.epoch_decode
193      prevstate=[]
194      lastepoch=[]
195      prevstatbestsymbol=[]
196      for i=1:self.Viterbiwindowwidth
197        prevstate=self.prevstatearray[state+1,idx+1]
198
199        prevstatbestsymbol=self.bestsymbolarray[state+1,idx+1]
200        idx=mod(idx-1,self.Viterbiwindowwidth)
201        lastepoch=epoch1
202        epoch1=mod(epoch1-1,self.Nepochs)
203        if (i!=self.Viterbiwindowwidth)
204          state=prevstate
```

```
205         end
206       end # i
207         bitsout1=self.inputbits[:,1,prevstatbestsymbol+1, ...
                prevstate+1,state+1,lastepoch+1]
208         bitsout2=self.inputbits[:,2,prevstatbestsymbol+1, ...
                prevstate+1,state+1,lastepoch+1]
209         decodeout=1
210     end #if
211     self.front=mod(self.front+1,self.Viterbiwindowwidth)
212     self.epoch_decode=mod(self.epoch_decode+1,self.Nepochs)
213     return decodeout,bitsout1,bitsout2
214     end #shape decoder viterbi
```

Code for Rate 3/4 (PPE)

```
1
2
3
4     trellisoutput=zeros(Int64,2,n,M,M,Nstate,Nstate,Nepochs)
5     inputbits=zeros(Int64,Nbits,k,M,M,Nstate,Nstate,Nepochs)
6     metrics=zeros(Nstate,1)
7     nextmetrics=zeros(Nstate,1)
8     prevstatearray=zeros(Int64,Nstate,Viterbiwindowwidth)
9     fromtostate=zeros(Int64,Nstate,Nstate,Nepochs)
10    bitqueue=zeros(Int64,Nbits,k,Viterbiwindowwidth)
11    bitqueuefront=0
12    bitqueueback=0
13
14    bestsymbol1array=zeros(Int64,Nstate,Viterbiwindowwidth)
15    bestsymbol2array=zeros(Int64,Nstate,Viterbiwindowwidth)
16
17
18    function setdecoderinfo(self::ShapeCodeFun,doplot,allstates,testbitsM)
19    if (doplot)
20
21    end
22    testbits=testbitsM
23
24    for epoch1=0:self.Nepochs-1
25      if (doplot)
26
27      end
28      if (self.doconstanttarget)
29        xcyclecount=0
30      else
31        xcyclecount=epoch1
32      end
33
34      tosym=self.xcycle[xcyclecount+1]
35
36      for state=0:self.Nstate-1
37        bitctr=0
```

```
38        sumrotcountsave=state
39        while (bitctr<(self.M*self.Nbits))
40          bitssymbol1=testbits[bitctr+1:bitctr+self.Nbits]
41          symno=bin2dec(bitssymbol1)
42          symnosave=symno
43          self.sumrotcount=sumrotcountsave
44          bitctr1=0
45          while (bitctr1<(self.M*self.Nbits))
46            bitssymbol2=testbits[bitctr1+1:bitctr1+self.Nbits]
47
48            symno2=bin2dec(bitssymbol2)       #conver secend symbol to decimel
49
50            symnosave2=symno2                                        #save ...
                  to a memory
51
52
53            bitctr2=0
54            while (bitctr2<(self.M*self.Nbits))
55              bitssymbol3=testbits[bitctr2+1:bitctr2+self.Nbits]
56              symno3=bin2dec(bitssymbol3)
57              symnosave3=symno3
58              #println(symno3)
59
60              An=mod(symno+self.sumrotcount,self.M)                  ...
                    #intermediate state after taking previosu number of ...
                    rotation into account
61              coderot1=mod(An+tosym,self.M)
62              An1=mod(self.sumrotcount+coderot1,self.M)
63
64              #tosym=self.xcycle[xcyclecount+1]
65              An2=mod(An1+symno2,self.M)
66              coderot2=mod(An2+tosym,self.M)
67              An3=mod(An1+coderot2,self.M)
68
69              An4=mod(An3+symno3,self.M)
70              coderot3=mod(An4+tosym,self.M)
71              An5=mod(coderot3+An3,self.M)
72
73              #tosym=self.xcycle[xcyclecount+1]
74              coderot=mod(coderot1+coderot2+coderot3,self.M)
75
76              An6=mod(An5+coderot,self.M)
77              An7=mod(An6+tosym,self.M)
78              sumrotcount=mod(An5+An7,self.Nstate)
79
80
81              fromstate=state
82              tostate=sumrotcount
83              #println("symn1=$(symnosave)  sym2=$(symnosave2) ...
                    sym3=$(symnosave3) parity=$(coderot) ...
                    fromstate=$(fromstate) tostate=$(sumrotcount) ...
                    epoch=$(epoch1) ")
84
85              self.trellisoutput[:,1,symnosave+1,symnosave2+1,fromstate+1, ...
                    tostate+1,epoch1+1]=self.const_pts[:,symnosave+1]
```

```
86              self.trellisoutput[:,2,symnosave+1,symnosave2+1,fromstate+1, ...
                  tostate+1,epoch1+1]=self.const_pts[:,symnosave2+1]
87              self.trellisoutput[:,3,symnosave+1,symnosave2+1,fromstate+1, ...
                  tostate+1,epoch1+1]=self.const_pts[:,symnosave3+1]
88              self.trellisoutput[:,4,symnosave+1,symnosave2+1,fromstate+1, ...
                  tostate+1,epoch1+1]=self.const_pts[:,coderot+1]
89
90              #println(bitssymbol1)
91              self.inputbits[:,1,symnosave+1,symnosave2+1,fromstate+1, ...
                  tostate+1,epoch1+1]=bitssymbol1
92              self.inputbits[:,2,symnosave+1,symnosave2+1,fromstate+1, ...
                  tostate+1,epoch1+1]=bitssymbol2
93              self.inputbits[:,3,symnosave+1,symnosave2+1,fromstate+1, ...
                  tostate+1,epoch1+1]=bitssymbol3
94
95
96                    #println(bitctr)
97            bitctr2=bitctr2+self.Nbits
98            end #while bitctr2
99              bitctr1=bitctr1+self.Nbits
100           end #while bitctr1
101           bitctr=bitctr+self.Nbits
102        end #while
103      end #end state
104      self.fromtostate[:,:,epoch1+1]=allstates
105    end #end epoch1
106    end #end setdecoderinfo
107
108
109    function encode(self::ShapeCodeFun,bits1,bits2,bits3)
110    symno1=bin2dec(bits1)
111    symnosave1=symno1
112    symno2=bin2dec(bits2)
113    symnosave2=symno2
114    symno3=bin2dec(bits3)
115    symnosave3=symno3
116    if (¬self.doconstanttarget)
117      xcyclecount=self.epoch
118    else
119      xcyclecount=0
120    end
121
122
123    tosym=self.xcycle[xcyclecount+1]
124
125    An=mod(symno1+self.sumrotcount,self.M)
126    coderot1=mod(An+tosym,self.M)
127    An1=mod(coderot1+self.sumrotcount,self.M)
128
129    An2=mod(symno2+An1,self.M)
130    coderot2=mod(An2+tosym,self.M)
131    An3=mod(coderot2+An1,self.M)
132
133    An4=mod(symno3+An3,self.M)
134    coderot3=mod(An4+tosym,self.M)
135    An5=mod(coderot3+An3,self.M)
```

```
136
137    coderot=mod(coderot1+coderot2+coderot3,self.M)
138
139
140    An6=mod(An5+coderot,self.M)
141    An7=mod(An6+tosym,self.M)
142    self.sumrotcount=mod(An7+An5,self.Nstate)
143
144    self.epoch=mod(self.epoch+1,self.Nepochs)
145    return symnosave1,symnosave2,symnosave3,coderot
146    end #end encode
147
148    function computemetric(r1,r2,r3,r4,output)
149    out11=output[1,1]
150    out12=output[1,2]
151    out13=output[1,3]
152    out14=output[1,4]
153    out21=output[2,1]
154    out22=output[2,2]
155    out23=output[2,3]
156    out24=output[2,4]
157    n1=(r1[1]-out11)^2+(r1[2]-out21)^2
158    n2=(r2[1]-out12)^2+(r2[2]-out22)^2
159    n3=(r3[1]-out13)^2+(r3[2]-out23)^2
160    n4=(r4[1]-out14)^2+(r4[2]-out24)^2
161    m=n1+n2+n3+n4
162    return m
163    end #end computemetric
164
165    function viterbishapedecode(self::ShapeCodeFun,r1,r2,r3,r4)
166    beststatemetric=Inf
167    for state=0:self.Nstate-1
168      nextmetricbest=Inf
169      for prevstate=self.fromtostate[:,state+1,self.epoch_decode+1]'
170        for symbol1=0:self.M-1
171          for symbol2=0:self.M-1
172            m=computemetric(r1,r2,r3,r4,self.trellisoutput[:,:,symbol1+1, ...
                   symbol2+1,prevstate+1,state+1, self.epoch_decode+1])
173            nextmetric=self.metrics[prevstate+1]+m
174            if (nextmetric<nextmetricbest)
175              nextmetricbest=nextmetric
176              global prevstatebest=prevstate
177              global symbolbest1=symbol1
178              global symbolbest2=symbol2
179              statebest=state
180            end
181          end #symbol2
182        end #symbol1
183      end #for prevstate
184      if (nextmetricbest < beststatemetric)
185        beststatemetric = nextmetricbest
186        self.beststate = state
187      end
188      self.nextmetrics[state+1]=nextmetricbest
189      self.prevstatearray[state+1,self.front+1]=prevstatebest
190      self.bestsymbol1array[state+1,self.front+1]=symbolbest1
```

```
191     self.bestsymbol2array[state+1,self.front+1]=symbolbest2
192   end #for state
193   self.metrics=copy(self.nextmetrics)
194   self.countbranches=self.countbranches+1
195   decodeout=0
196   bitsout1=[0;0]
197   bitsout2=[0;0]
198   bitsout3=[0;0]
199   if (self.countbranches≥self.Viterbiwindowwidth)
200     state=self.beststate
201     idx=self.front
202     epoch1=self.epoch_decode
203     prevstate=[]
204     lastepoch=[]
205     prevstatbestsymbol1=[]
206     prevstatbestsymbol2=[]
207     for i=1:self.Viterbiwindowwidth
208       prevstate=self.prevstatearray[state+1,idx+1]
209
210       prevstatbestsymbol1=self.bestsymbol1array[state+1,idx+1]
211       prevstatbestsymbol2=self.bestsymbol2array[state+1,idx+1]
212       idx=mod(idx-1,self.Viterbiwindowwidth)
213       lastepoch=epoch1
214       epoch1=mod(epoch1-1,self.Nepochs)
215       if (i!=self.Viterbiwindowwidth)
216         state=prevstate
217       end
218     end # i
219       bitsout1=self.inputbits[:,1,prevstatbestsymbol1+1, ...
             prevstatbestsymbol2+1,prevstate+1, state+1,lastepoch+1]
220       bitsout2=self.inputbits[:,2,prevstatbestsymbol1+1, ...
             prevstatbestsymbol2+1,prevstate+1, state+1,lastepoch+1]
221       bitsout3=self.inputbits[:,3,prevstatbestsymbol1+1 ...
             ,prevstatbestsymbol2+1,prevstate+1, state+1,lastepoch+1]
222       decodeout=1
223   end #if
224   self.front=mod(self.front+1,self.Viterbiwindowwidth)
225   self.epoch_decode=mod(self.epoch_decode+1,self.Nepochs)
226   return decodeout,bitsout1,bitsout2,bitsout3
227   end #shape decoder viterbi
228
229 # ------------------------------------------------------------
230
231   function viterbishapecodeflush(self::ShapeCodeFun)
232   bitsout=zeros(Int64,self.Nbits,self.Viterbiwindowwidth-1)
233   state=self.beststate
234   epoch1=mod(self.epoch_decode-1,self.Nepochs)
235   idx=mod(self.front-1,self.Viterbiwindowwidth)
236   for i=1:self.Viterbiwindowwidth-1
237     prevstate=self.prevstatearray[state+1,idx+1]
238     lastepoch=epoch1
239     bits1=self.inputbits[:,prevstate+1,state+1,lastepoch+1]
240     idx=mod(idx-1,self.Viterbiwindowwidth)
241     epoch1=mod(epoch1-1,self.Nepochs)
242     bitsout[:,self.Viterbiwindowwidth-1-i]=bits1
243     state=prevstate
```

```
244    end #for
245    end #viterbishapecodeflush
246  #---------------------------------------------------------------
247
248    function bitqueuein(self::ShapeCodeFun,bits1,bits2,bits3)
249    self.bitqueue[:,1,self.bitqueuefront+1]=bits1
250    self.bitqueue[:,2,self.bitqueuefront+1]=bits2
251    self.bitqueue[:,3,self.bitqueuefront+1]=bits3
252    self.bitqueuefront = mod(self.bitqueuefront + 1, self.Viterbiwindowwidth)
253    end
254
255    function bitqueueout(self::ShapeCodeFun)
256    bits1=self.bitqueue[:,1,self.bitqueueback+1]
257    bits2=self.bitqueue[:,2,self.bitqueueback+1]
258    bits3=self.bitqueue[:,3,self.bitqueueback+1]
259    self.bitqueueback = mod(self.bitqueueback + 1, self.Viterbiwindowwidth)
260    return bits1,bits2 ,bits3
261    end
262
263  #create object of ShapeCodeFun--------------------------------------
264
265
266    a=ShapeCodeFun(M,Viterbiwindowwidth,Nstate,Nepochs,Nbits,
267                   fromtodist,cumrotoffset,const_pts,xcycle,0,
268                   0,doconstanttarget,trellisoutput,inputbits, ...
269                       fromtostate,metrics,
270                   nextmetrics,0,prevstatearray,0,0,0,bitqueue,0,0, ...
                       bestsymbol1array,bestsymbol2array)
270
271
272    setdecoderinfo(a,doplot,allstates,testbits)
273    resetshapecoder(a)
274    resetshapedecoder(a)
275
276  end #doshapecode
277
278  nerrortocount=100
279
280  Eb=1
281  R=1
282  Es=Nbits*Eb*R
283  Esscale=sqrt(3*Es/(2*(M-1)))
284  Rcoded=k/n
285  Escoded=sqrt(Nbits*Eb*Rcoded*3/(2*(M-1)))
286  theorprobQPSK=[]
287  theorprobBPSK=[]
288  theoprobMquam=[]
289  for SNR in SNRrangetheo
290    println("snr=$(SNR)")
291
292    N0=Eb*10^(-SNR/10)
293    EbN0=Eb/N0
294
295    push!(theorprobQPSK,(1 - (1-qf(sqrt(2*EbN0)))^2))
296    push!(theorprobBPSK,(qf(sqrt(2*EbN0))))
297
```

```
298    push!(theoprobMquam, ...
             (1-(1-2*(1-sqrt(1/M))*qf(sqrt((3/(M-1))*Nbits*EbN0)))^2))
299  end
300
301
302  r=[0.0 0.0]
303  if (doMquam)
304    snrctr=0
305
306    errctr=[]
307    errctr_bits=[]
308    nsyms=[]
309    for SNR in SNRrangeex
310      println("snr=$(SNR)")
311      snrctr=snrctr+1
312      N0=Eb*10^(-SNR/10)
313      EbN0=Eb/N0
314      #println("N0=$(N0) EbN0=$(EbN0)")
315
316      sigma2=N0/2
317      sigma=sqrt(sigma2)
318
319      push!(errctr,0)
320      push!(errctr_bits,0)
321
322      push!(nsyms,0)
323
324
325      while(errctr[snrctr]<nerrortocount)
326        nsyms[snrctr]=nsyms[snrctr]+1
327        bits=randombits(Nbits)
328        symno=bin2dec(bits)
329        constpts=Esscale*const_pts[:,symno+1]
330
331        noise=sigma*randn(1,2)
332
333        r[1]=constpts[1]+noise[1]
334        r[2]=constpts[2]+noise[2]
335
336        decodesymno=mquamdemod(r/Esscale,const_pts,const_axis)
337
338        #println("decodesymno=$(decodesymno) symno=$(symno)")
339
340        if (decodesymno != symno)
341
342          errctr[snrctr]=errctr[snrctr]+1
343          errctr_bits[snrctr] = errctr_bits[snrctr] + ...
                   sum(Int64.(decodesymno .!= symno))
344
345          println("SNR=$(SNR):uncoded error no=$(errctr[snrctr])")
346          #println("decodesymno=$(decodesymno) symno=$(symno)")
347        end
348       if (errctr[snrctr]!=0 && errctr[snrctr]/nsyms[snrctr]≤GraphPsrange)
349          break
350        end   #end break while
351      end #while errctr
```

```
352      if (errctr[snrctr]/nsyms[snrctr]≤GraphPsrange)
353        break
354      end #end brek for
355    end #for snr
356  end #if(doqpsk)
357
358  if (doshapecode)
359      errctr_coded=[]
360      errctr_bits_coded=[]
361      nsyms_coded=[]
362      snrctr=0
363      for SNR in SNRrange
364        println("SNRShapeCode=$(SNR)")
365        snrctr=snrctr+1
366        N0=Eb*10^(-SNR/10)
367        EbN0=Eb/N0
368        sigma2=N0/2
369        sigma=sqrt(sigma2)
370        push!(errctr_coded,0)
371        push!(errctr_bits_coded,0)
372        push!(nsyms_coded,0)
373        while(errctr_coded[snrctr]<nerrortocount)
374          nsyms_coded[snrctr]=nsyms_coded[snrctr]+3
375
376          if (rem(nsyms_coded[snrctr],10000)==0 && (nsyms_coded[snrctr]≥ ...
                 10000))
377            println("number of symbol=$(nsyms_coded[snrctr])")
378          end
379
380
381
382
383
384          bits1=randombits(Nbits)
385          bits2=randombits(Nbits)
386          bits3=randombits(Nbits)
387          sym1,sym2,sym3,parity=encode(a,bits1,bits2,bits3)
388          bitqueuein(a,bits1,bits2,bits3)
389          s1=Escoded*a.const_pts[:,sym1+1]
390          s2=Escoded*a.const_pts[:,sym2+1]
391          s3=Escoded*a.const_pts[:,sym3+1]
392          s4=Escoded*a.const_pts[:,parity+1]
393          noise1=sigma*randn(2,1)
394          noise2=sigma*randn(2,1)
395          noise3=sigma*randn(2,1)
396          noise4=sigma*randn(2,1)
397          r1=s1+noise1
398          r2=s2+noise2
399          r3=s3+noise3
400          r4=s4+noise4
401
402          decodeout,bitsout1,bitsout2,bitsout3=viterbishapedecode ...
                 (a,r1/Escoded,r2/Escoded,r3/Escoded,r4/Escoded)
403
404          if (decodeout==1)
405            inbits1,inbits2,inbits3=bitqueueout(a)
```

```
406          #println("inbits 1=$(inbits1) bitsout1=$(bitsout1) inbits ...
                 2=$(inbits2) bitsout2=$(bitsout2) inbits 3=$(inbits3)   ...
                 bitsout3=$(bitsout3)")
407          if (inbits1!=bitsout1 || inbits2!=bitsout2 || inbits3!=bitsout3)
408            println("SNR=$(SNR):coded error no=$(errctr_coded[snrctr])")
409          end
410          #println("inbit+sum(Int64.(inbits2 .!= bitsout2))s=$(inbits) ...
                 bitsout=$(bitsout)")
411          errctr_coded[snrctr]=errctr_coded[snrctr]+ ...
                 Int64(any(inbits1!=bitsout1))+ ...
                 Int64(any(inbits2!=bitsout2))+Int64(any(inbits3!=bitsout3))
412          errctr_bits_coded[snrctr] = errctr_bits_coded[snrctr] + ...
                 sum(Int64.(inbits1 .!= bitsout1))+ sum(Int64.(inbits2 .!= ...
                 bitsout2))+ sum(Int64.(inbits3 .!= bitsout3))
413        end #decodeout
414        if (errctr_coded[snrctr]!=0 && ...
               errctr_coded[snrctr]/nsyms_coded[snrctr]≤GraphPsrange)
415          break
416        end   #end break while
417      end #while error count
418      if (errctr_coded[snrctr]/nsyms_coded[snrctr]≤GraphPsrange)
419        break
420      end #end break for
421      @save "$(M)QAMrate34method1$(datetoday).jld"
422    end #for SNRrangeShapecode
423  end #doshapecode
424  legendstr=[]
425  push!(legendstr,"Uncoded QPSK(th) SER")
426  push!(legendstr,"Uncoded BPSK(th) SER")
427  push!(legendstr,"Uncoded $(M)QAM(th) SER")
428
429  if (doMquam)
430    push!(legendstr,"Uncoded $(M)QAM(ex) SER")
431    push!(legendstr,"Uncoded $(M)QAM(ex) BER")
432  end
433  if (doshapecode)
434    push!(legendstr,"Rate 3/4 CAC $(M)QAM SER")
435    push!(legendstr,"Rate 3/4 CAC $(M)QAM BER")
436  end
437  len=length(SNRrange)
438
439  fig1,ax1=subplots()
440  ax1[:semilogy](collect(0:1:length(theorprobQPSK)-1), ...
        theorprobQPSK,"g:",marker="^")
441  ax1[:semilogy](collect(0:1:length(theorprobBPSK)-1), ...
        theorprobBPSK,"b:",marker="<")
442  ax1[:semilogy](collect(0:1:length(theoprobMquam)-1), ...
        theoprobMquam,"k:",marker=">")
443
444  if (doMquam)
445    ax1[:semilogy](collect(0:1:length(errctr)-1), ...
          errctr./(nsyms),"r",marker="v")
446    ax1[:semilogy](collect(0:1:length(errctr_bits)-1), ...
          errctr_bits./(nsyms*Nbits),"r:",marker="d")
447  end
448
```

```
449  if (doshapecode)
450    ax1[:semilogy](collect(0:1:length(errctr_coded)-1), ...
            errctr_coded./nsyms_coded,"c",marker="s")
451    ax1[:semilogy](collect(0:1:length(errctr_coded)-1), ...
            errctr_bits_coded./(Nbits*nsyms_coded),"c:",marker="o")
452  end
453
454
455  ax1[:legend](legendstr)
456  ax1[:set_ylim]([GraphPsrange,10.0^-0])
457  ax1[:grid](color="k", linestyle="--")
458  xlabel(L"(SNR)_{dB}")
459  ylabel(L"P_s, P_b" )
460  xticks(SNRrangetheo)
461  toc()
462  Ps=-1*Int(log10(GraphPsrange))
463  savefig("BER$(M)QUAMSCSNR$(SNRrange[end])Ps$(Ps) ...
          Rate34method2$(datetoday).pdf",dpi=1500)
464  @save "$(M)QAMrate34method2$(datetoday).jld"
```

B.3 Python Code

Python Code for Signal Point Target Code Using QPSK Constellation

```
1
2  #!/usr/bin/env python3
3  # -*- coding: utf-8 -*-
4  """
5  Created on Wed Jun 27 08:58:24 2018
6
7  @author: munibun
8  """
9
10  SNRrange=8  # It will count SNRrange-1
11  import time
12  start_time = time.time()
13
14  import numpy as np
15  import math
16  import matplotlib.pyplot as plt
17
18  def qf(x):
19      return .5*math.erfc(x/math.sqrt(2))
20
21  def qpskdemod(r):
22      if (r[1]≥0):
23          bit1=1
24      else:
25          bit1=0
26      if (r[0]≥0):
```

```python
27          bit2=1
28      else:
29          bit2=0
30
31      return 2*bit1+bit2
32
33  def randombits():
34      genrand=np.random.rand(2,1)
35      bits=np.zeros(shape=(2,1),dtype=int).reshape(2,)
36      if genrand[0]>.5:
37          bits[0]=1
38      else:
39          bits[0]=0
40      if genrand[1]>.5:
41          bits[1]=1
42      else:
43          bits[1]=0
44
45      return bits
46
47  class ShapeCodeFunctions1:
48      M=int(4)
49      viterbiwindowwidth=int(10)
50
51      def __init__(self,doconstanttarget):       #basic constructor
52
53          self.doconstanttarget=doconstanttarget
54          self.Nstate= self.M
55          self.Nepoch=self.M
56
57          self.Nbits=int(math.log2(self.M))
58
59
60          self.const_pts=np.transpose(np.array([[-1,-1],       #D(0)
61                                                 [1,-1],        #C(1)
62                                                 [-1,1],        #A(2)
63                                                 [1,1]]).astype(int))      #B(3))
64
65
66          self.fromtodist=np.array([[0,3,1,2],
67                                    [1,0,2,3],
68                                    [3,2,0,1],
69                                    [2,1,3,0]]).astype(int)
70
71
72          self.cumrotoffset=np.array([[0,2,3,1],
73                                      [1,0,2,3],
74                                      [2,3,1,0],
75                                      [3,1,0,2]]).astype(int)
76
77
78          if (not(doconstanttarget)):
79              self.xcycle=np.array([2,3,1,0]).astype(int)
80          else:
81              self.xcycle=np.array([2]).astype(int)
82
```

```python
83          self.epoch=0
84          self.sumrotcount=0
85
86          #end of constructor-----------------------------
87
88      def resetshapecoder(self):
89          self.epoch=0
90          self.sumrotcount=0
91
92          #end of resetshapecoder-------------------------
93
94      def resetshapedecoder(self):
95          self.epoch_decode=0
96          self.front=0
97          self.countbranches=0
98          self.bitqueuefront=0
99          self.bitqueueback=0
100
101          #end of resershapedeocder-----------------------
102
103      def encode(self,bits):
104              symno=2*bits[0]+bits[1]
105              symnosave=symno
106              symno=self.cumrotoffset[symno,self.sumrotcount]
107              if(not(self.doconstanttarget)):
108                  xcyclecount=self.epoch
109              else:
110                  xcyclecount=0
111              tosym=self.xcycle[xcyclecount]
112              coderot=self.fromtodist[symno,tosym]
113              self.sumrotcount=(self.sumrotcount+coderot)%self.M
114              self.epoch=(self.epoch+1)%4
115              return symnosave,coderot
116
117              #end of encode ------------------------------
118
119      def setdecoderinfo(self,doplot):
120          if(doplot):
121              fig, ax = plt.subplots(1,1)
122
123              plt.clf()
124
125
126              for spine in plt.gca().spines.values():
127                  spine.set_visible(False)
128
129
130              ax.set_xlim(-.8, 4.2)
131              ax.set_ylim(-.8, 3.2)
132
133              plt.xticks(range(0, 5, 1), fontsize=14)
134              plt.yticks(range(0, 4, 1), fontsize=14)
135
136              #plt.tick_params(
137              #        axis='x',          # changes apply to the x-axis
```

```python
138             #        which='both',      # both major and minor ticks are ...
                    affected
139             #        bottom=False,      # ticks along the bottom edge are off
140             #        top=False,         # ticks along the top edge are off
141             #        labelbottom=False) # labels along the bottom edge ...
                    are off
142             plt.gca().invert_yaxis()
143
144             xplotsep=1
145             yplotsep=1
146             #markersize = 15
147             mulist = [0.15, 0.45, 0.60, 0.75]
148             xcyclesym=['A','B','C','D']
149
150         self.trellisoutput=np.ndarray(
151         shape=(2,2,self.Nstate,self.Nstate,self.Nepoch),dtype=int)
152         self.inputbits=np.ndarray(
153         shape=(self.Nbits,self.Nstate,self.Nstate,self.Nepoch),dtype=int)
154         self.metrics=np.zeros(shape=(self.Nstate,1),dtype=int)
155         self.nextmetrics=np.zeros(shape=(self.Nstate,1),dtype=int)
156
157         self.fromtostate=np.ndarray(
158         shape=(self.Nstate,self.Nstate,self.Nepoch),dtype=int)
159
160         testbits=[0,0,0,1,1,0,1,1]
161
162         for epoch1 in range(self.Nepoch):
163             if(doplot):
164                     plottime=epoch1
165
166                     plotx1=plottime
167                     plotx2=plottime+xplotsep
168                     plt.text( plotx2,-.3,xcyclesym[plottime])
169
170             if (self.doconstanttarget):
171                 xcyclecount=0
172             else:
173                 xcyclecount=epoch1
174
175             targetsymbol=self.xcycle[xcyclecount]
176
177             for init_total_rot in range(self.M):
178                 bitcount=0
179                 init_total_rot_save=init_total_rot
180                 #print("\n initial sum rot count(state)={0}  target ...
                        symbol={1}".format(init_total_rot,self.xcycle[epoch1]))
181
182                 while (bitcount< len(testbits)):
183                     self.sumrotcount=init_total_rot_save
184                     sumno=2*testbits[bitcount]+testbits[bitcount+1]
185                     currentposition=self.cumrotoffset[sumno,self.sumrotcount]
186                     codeoutput=self.fromtodist[currentposition,targetsymbol]
187                     self.sumrotcount=(self.sumrotcount+codeoutput)%self.M
188                     # print("bits={0}{1}  out=({2},{3})   ...
                            sumrotcount={4}".format(testbits[bitcount] ...
```

```
                              ,testbits[bitcount+1],sumno, ...
                                  codeoutput,self.sumrotcount))
189                        fromstate=init_total_rot
190                        tostate=self.sumrotcount
191                        self.trellisoutput[:,0,fromstate,tostate,epoch1]= ...
                                  self.const_pts[:,sumno]
192                        self.trellisoutput[:,1,fromstate,tostate,epoch1]= ...
                                  self.const_pts[:,codeoutput]
193                        self.inputbits[:,fromstate,tostate,epoch1]= ...
                                  np.array([testbits[bitcount], ...
                                  testbits[bitcount+1]]).astype(int)
194
195                        if(doplot):
196                            from_pos=init_total_rot_save*yplotsep
197                            to_pos=self.sumrotcount*yplotsep
198
199                            plt.plot([plotx1,plotx2],[from_pos,to_pos],c='k')
200                            plt.plot(plotx1,from_pos,'ko')
201                            plt.plot(plotx2,from_pos,'ko')
202
203                            strng="{0}{1}/{2}{3}".format(
204                    testbits[bitcount],testbits[bitcount+1],sumno,codeoutput)
205
206                            mu=mulist[init_total_rot]
207                            textx = (1-mu)*plotx1 + mu*plotx2
208                            texty = (1-mu)*from_pos + mu*to_pos
209                            tangle = (180/math.pi)*math.atan((to_pos - ...
                                  from_pos)/(plotx2 - plotx1));
210                            trans_angle = ...
                                  plt.gca().transData.transform_angles( ...
                                  np.array((tangle,)), ...
                                  np.array([textx,texty]).reshape((1, 2)))[0]
211                            plt.text(textx, texty, strng,color='r', ...
                                  fontsize=8,rotation=trans_angle, ...
                                  horizontalalignment='left', ...
                                  verticalalignment='bottom', ...
                                  rotation_mode='anchor')
212                    bitcount=bitcount+2
213            self.fromtostate[:,:,epoch1]=np.array([[0,0,0,0],
214                                                    [1,1,1,1],
215                                                    [2,2,2,2],
216                                                    [3,3,3,3]]).astype(int)
217        if(doplot):
218
219            plt.savefig('trellisfromshapecodefunction.png',dpi=1500)
220            plt.show()
221
222        self.epoch_decode=0
223        self.bitqueuefront=0
224        self.bitqueueback=0
225        self.front=0
226        self.countbranches=0
227        self.prevstatearray=np.zeros((self.Nstate, ...
                self.viterbiwindowwidth)).astype(int)
228        self.bitqueue=np.zeros((self.Nbits, ...
                self.viterbiwindowwidth)).astype(int)
```

```python
229
230             #end of setdecoderinfo---------------------------------
231
232     def computemetric(self,r1,r2,output):
233         m=(r1[0]-output[0,0])**2+(r1[1]-output[1,0])**2 ...
                +(r2[0]-output[0,1])**2 +(r2[1]-output[1,1])**2
234         return m
235
236     #end of compute metric--------------------
237
238     def viterbishapedecoder(self,r1,r2):
239         beststatemetric=math.inf
240         for state in range(self.Nstate):
241             bestnextmetric=math.inf
242             for prevstate in self.fromtostate[:,state,self.epoch_decode]:
243                 m=self.computemetric(r1,r2, ...
                        self.trellisoutput[:,:,prevstate,state,self.epoch_decode])
244                 nextmetric=self.metrics[prevstate]+m
245
246                 if(nextmetric<bestnextmetric):
247                     bestnextmetric=nextmetric
248                     bestprevstate=prevstate
249                     statebest=state
250             if(bestnextmetric<beststatemetric):
251                 beststatemetric=bestnextmetric
252                 self.beststate=state
253
254             self.nextmetrics[state]=bestnextmetric
255             self.prevstatearray[state,self.front]=bestprevstate
256             #print('state: {0} bestprevstate={1} beststate={2} ...
                    bestmetric={3}\n'.format(state,bestprevstate,statebest, ...
                    int(bestnextmetric)))
257         self.metrics[:]=self.nextmetrics[:]
258         self.countbranches=self.countbranches+1
259         decodeout=0
260         bitsout=[0,0]
261
262         if(self.countbranches>=self.viterbiwindowwidth):
263             state=self.beststate
264             idx=self.front
265             epoch1=self.epoch_decode
266             for i in range(self.viterbiwindowwidth):
267
268                 prevstate=self.prevstatearray[state,idx]
269
270                 lastepoch=epoch1
271                 epoch1=(epoch1-1)%self.Nepoch
272                 idx=(idx-1)%self.viterbiwindowwidth
273
274                 if(i!=self.viterbiwindowwidth-1):
275                     state=prevstate
276             decodeout=1
277             bitsout=self.inputbits[:,prevstate,state,lastepoch]
278
279         self.front=(self.front+1)%self.viterbiwindowwidth
280         self.epoch_decode=(self.epoch_decode+1)%self.Nepoch
```

```python
281          return decodeout,bitsout
282
283  #end of viterbishapedecoder--------------------------------------
284
285      def viterbishapedecoderflush(self):
286          bitsout=np.zeros((self.Nbits,self.viterbiwindowwidth-1)).astype(int)
287          state=self.beststate
288          epoch1=(self.epoch_decode-1)%self.Nepoch
289          idx=(self.front-1)%self.viterbiwindowwidth
290          for i in range(self.viterbiwindowwidth-2):
291              prevstate=self.prevstatearray[state,idx]
292              lastepoch=epoch1
293              bits=self.inputbits[:,prevstate,state,lastepoch]
294              idx=(idx-1)%self.viterbiwindowwidth
295              epoch1=(epoch1-1)%self.Nepoch
296              bitsout[:,self.viterbiwindowwidth-2-i]=bits
297              state=prevstate
298
299  #end of viterbishapedecoderflush----------------------------------
300
301  #end of viterbishapedecoderflush----------------------------------
302
303      def bitqueuein(self,bits):
304          self.bitqueue[:,self.bitqueuefront]=bits[:]
305          self.bitqueuefront=(self.bitqueuefront+1)%self.viterbiwindowwidth
306  #end of bitqueuein------------------------------------------------
307      def bitqueueout(self):
308          bits=self.bitqueue[:,self.bitqueueback]
309          self.bitqueueback=(self.bitqueueback+1)%self.viterbiwindowwidth
310          return bits
311  #end of bitqueueout ----------------------------------------------
312
313  shapecodeclass1=ShapeCodeFunctions1(1)
314  shapecodeclass1.resetshapecoder()
315  shapecodeclass1.resetshapedecoder()
316  shapecodeclass1.setdecoderinfo(0)
317
318  nerrortocount=100
319  Eb=1
320  R=1
321  Es=2*Eb*R
322  Esscale=math.sqrt(Es/2)
323  Rcoded=.5
324  Escoded=math.sqrt(2*Eb*Rcoded/2)
325  doqpsk=1
326  doshapecode=1
327  snrctr=0
328  theorprobQPSK=[0]*SNRrange
329  theorprobBPSK=[0]*SNRrange
330  errctr=[0]*SNRrange
331  nsyms=[0]*SNRrange
332  errctr_coded=[0]*SNRrange
333  errctr_bits_coded=[0]*SNRrange
334  nsyms_coded=[0]*SNRrange
335
336  r=[0,0]
```

```
337
338  for SNR in range(SNRrange):
339      print('snr={0}\n'.format(SNR))
340      N0=Eb*10**(-SNR/10)
341      EbN0=Eb/N0
342      theorprobQPSK[snrctr]=(1-(1-qf(math.sqrt(2*EbN0)))**2)
343      theorprobBPSK[snrctr]=(qf(math.sqrt(2*EbN0)))
344      sigma=math.sqrt(N0/2)
345
346      if(doqpsk):
347          #errctr.append(0)
348          #nsyms.append(0)
349          while (errctr[snrctr]<nerrortocount):
350              nsyms[snrctr]=nsyms[snrctr]+1
351              bits1=np.random.rand(2)>.5
352              bits2=bits1.astype(int)
353              symno=2*bits2[0]+bits2[1]
354              constpt=Esscale*shapecodeclass1.const_pts[:,symno]
355              noise1=sigma*np.random.randn(2)
356              r[0]=constpt[0]+noise1[0]
357              r[1]=constpt[1]+noise1[1]
358              decodesymno=qpskdemod(r)
359              if (decodesymno != symno):
360                  errctr[snrctr]=errctr[snrctr]+1
361
362      if(doshapecode):
363          #errctr_coded.append(0)
364          #errctr_bits_coded.append(0)
365          #nsyms_coded.append(0)
366          while(errctr_coded[snrctr]<nerrortocount):
367              nsyms_coded[snrctr]=nsyms_coded[snrctr]+1
368              bits3=np.random.rand(2)>.5
369              bits4=bits3.astype(int)
370              bits4=bits4.reshape((2,))
371              sym1,sym2=shapecodeclass1.encode(bits4)
372              shapecodeclass1.bitqueuein(bits4)
373              s1 = Escoded*shapecodeclass1.const_pts[:,sym1]
374              s2 = Escoded*shapecodeclass1.const_pts[:,sym2]
375
376              noise2=sigma*np.random.randn(2)
377              noise3=sigma*np.random.randn(2)
378              r1=s1+noise2
379              r2=s2+noise3
380              decodeout,bitsout= shapecodeclass1.viterbishapedecoder(r1,r2)
381              if(decodeout):
382                  inbits = shapecodeclass1.bitqueueout()
383                  errctr_coded[snrctr] = errctr_coded[snrctr] + ...
384                      np.any((inbits) != (bitsout)).astype(int)
                     errctr_bits_coded[snrctr]=errctr_bits_coded[snrctr] ...
                         +sum(((inbits) != (bitsout)).astype(int))
385
386      snrctr=snrctr+1
387
388  lenn=len(range(SNRrange))
389  legendstrs=[]
390
```

```
391  plt.semilogy(list(range(SNRrange)),theorprobQPSK,'g:')
392  plt.semilogy(list(range(SNRrange)),theorprobBPSK,'b:')
393
394  legendstrs.append('BPSK(th)')
395  legendstrs.append( 'QPSK(th)')
396
397  if(doqpsk):
398      plt.semilogy(list(range(SNRrange)),np.array(errctr)/np.array(nsyms),'r:')
399      legendstrs.append( 'QPSK(ex)')
400
401  if(doshapecode):
402      plt.semilogy(list(range(SNRrange)), ...
               np.array(errctr_coded)/np.array(nsyms_coded),'c')
403      legendstrs.append( 'Shapecode(sym)')
404      plt.semilogy(list(range(SNRrange)), ...
               np.array(errctr_bits_coded)/(2*np.array(nsyms_coded)),'c:')
405      legendstrs.append( 'Shapecode(bit)')
406  plt.legend(legendstrs)
407  print("--- %s seconds ---" % (time.time() - start_time))
408  if (shapecodeclass1.doconstanttarget):
409      plt.savefig('BERcurveconstatnttarget.png',dpi=1500)
410  else:
411      plt.savefig('BERcurve.png',dpi=1500)
412  plt.show()
```

Python Code for finding Free Euclidean Distance

```
1
2
3  from collections import defaultdict
4  import numpy as np
5  import matplotlib.pyplot as plt
6  import math
7  class CreatePath:
8
9      def __init__(self,totalstates):
10          self.vertices= totalstates
11          self.statenode = defaultdict(list)
12          self.branchlist=defaultdict(list)
13          self.index=0
14
15      def AddBranch(self,initstate,finalstate):
16          self.statenode[initstate].append(finalstate)
17
18      def CheckVisitedBranchs(self, initstate, targetstate, branchvisited, ...
               branch):
19
20
21          branchvisited[initstate]= True
22          branch.append(initstate)
23          if (initstate ==targetstate):
24
```

```python
25              branchh=list(branch)
26              self.branchlist[self.index]=branchh
27              self.index=self.index+1
28          else:
29
30              for ind in self.statenode[initstate]:
31                  if branchvisited[ind]==False:
32                      self.CheckVisitedBranchs(ind, targetstate, ...
                            branchvisited, branch)
33          branch.pop()
34          branchvisited[initstate]= False
35
36      def MakePaths(self,startingstate, targetstate):
37          branchvisited =[False]*(self.vertices)
38          branch = []
39          self.CheckVisitedBranchs(startingstate, ...
                targetstate,branchvisited, branch)
40
41
42
43
44  def distancecal(r1,r2):
45      return (r1[0]-r2[0])**2+(r1[1]-r2[1])**2
46
47  doplot=1
48  if doplot:
49
50      for spine in plt.gca().spines.values():
51              spine.set_visible(False)
52
53      plt.gca().invert_yaxis()
54
55
56  loopcontrol=1
57  Nepochinit=0
58  M=16
59  Nstate=128
60  Nbits=math.log2(M)
61  target=0
62  mulipliedfactor=19
63  const_axis=np.arange(-(2**(Nbits/2)-1),(2**(Nbits/2)+1),2,dtype=int)
64  axis1=np.kron(np.ones(shape=(1,int(2**(Nbits/2))),dtype=int),const_axis)
65  axis2=np.kron(const_axis,np.ones(shape=(1,int(2**(Nbits/2))),dtype=int))
66  const_pts=np.concatenate((axis1, axis2), axis=0)
67  print(const_pts)
68  #const_pts=np.array([[ -3, -3, -3, -3, -1, -1, -1, -1, 3, 3, 3, 3, 1, 1, ...
        1, 1],[-3, -1, 3, 1, -3, -1, 3, 1, -3, -1, 3, 1, -3, -1, 3, 1]],np.int32)
69  #---------------------0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
70  #const_pts=const_pts[:,[1,0,15,5,14,2,8,6,9,10,4,13,7,11,12,3]]
71  print(const_pts)
72
73
74  if M==4:
75      const_pts=np.array([[-1,1,1,-1],[-1,-1,1,1]],np.int32)
76  #bleow is dr sodha const for 16 QAM
77  #if M==16:
```

```
78  #      const_pts=np.array([[ -3, -3, -3, -3, -1, -1, -1, -1, 3, 3, 3, 3, 1, ...
        1, 1, 1],[-3, -1, 3, 1, -3, -1, 3, 1, -3, -1, 3, 1, -3, -1, 3, ...
        1]],np.int32)
79  symbollist=list(range(M))
80  bestmindistance=0
81  #const_pts=np.transpose(const_pts)
82  #np.random.shuffle(const_pts)
83  #const_pts=np.transpose(const_pts)
84  #while(loopcontrol==1):
85
86  for s in list(range(Nstate)):
87
88      for offset in list(range(0-s,Nstate-s,1)):
89
90          while(loopcontrol<3):
91
92              Nepochinit=Nepochinit+1
93
94              Nepoch=Nepochinit+1
95
96              trellisoutput=np.zeros(shape=(2,2,Nstate,Nstate,Nepoch), ...
                    dtype=int)*10000
97              trellisbranch=np.zeros(shape=(2,Nstate,Nstate,Nepoch),dtype=int)
98
99
100             g = CreatePath(Nstate*(Nepoch+1))
101             initnode=0
102             if doplot:
103                 plt.xticks(range(0, Nepoch+1, 1), fontsize=14)
104                 plt.yticks(range(0, Nstate+1, 1), fontsize=14)
105
106             for epoch in list(range(Nepoch)):
107
108                 for state in list(range(Nstate)):
109                     fromstate=state
110                     symbolnum=0
111                     #print(fromstate)
112
113                     for symbol in symbollist:
114                         #print(initnode)
115
116                         newpoint=(symbol+state)%M
117                         parity=(newpoint+target)%M
118                         tostate=(mulipliedfactor*(parity+state)) %Nstate
119
120                         fromy=fromstate
121                         toy=tostate
122                         trellisoutput[:,0,fromstate,tostate,epoch]= ...
                                const_pts[:,symbol]
123                         trellisoutput[:,1,fromstate,tostate,epoch]= ...
                                const_pts[:,parity]
124                         trellisbranch[0,fromstate,tostate,epoch]=symbol
125                         trellisbranch[1,fromstate,tostate,epoch]=parity
126                         g.AddBranch(fromy+initnode,initnode+Nstate+toy)
127                     initnode=initnode+Nstate
128
```

```python
129
130
131              d = Nepoch*Nstate+s+offset
132
133              g.MakePaths(s, d)
134
135              branchlist=defaultdict(list)
136              revisedbranchlist=defaultdict(list)
137              index=0
138
139
140              for row in g.branchlist:
141
142                  for element in g.branchlist[row]:
143                      branchlist[index].append(element% Nstate)
144
145                  index=index+1
146
147
148              index=0
149              for keyy in branchlist:
150                  checkk=branchlist[keyy]
151                  if checkk.count(s)==2 or checkk.count(s)==Nepoch+1:
152                      for element in checkk:
153                          revisedbranchlist[index].append(element)
154                      index=index+1
155
156
157              indexsave=index
158
159              if doplot:
160                  for epoch in list(range(Nepoch)):
161                      plotx1=epoch
162                      plotx2=epoch+1
163                      for key in revisedbranchlist:
164                          fromy=revisedbranchlist[key][plotx1]
165                          toy=revisedbranchlist[key][plotx2]
166                          plt.plot([plotx1,plotx2],[fromy,toy],c='k')
167                          plt.plot(plotx1,fromy,'ko')
168                          plt.plot(plotx2,fromy,'ko')
169
170
171
172              #phase 2
173
174              if(Nepoch>2):
175                  Nepoch=Nepoch-1
176                  g = CreatePath(Nstate*(Nepoch+1))
177                  initnode=0
178
179
180
181
182                  for epoch in list(range(Nepoch)):
183
184                      for state in list(range(Nstate)):
```

```python
185                         fromstate=state
186                         symbolnum=0
187                         #print(fromstate)
188
189                         for symbol in symbollist:
190                             #print(initnode)
191
192                             newpoint=(symbol+state)%M
193                             parity=(newpoint+target)%M
194                             tostate=(mulipliedfactor*(parity+state)) %Nstate
195
196                             fromy=fromstate
197                             toy=tostate
198                             g.AddBranch(fromy+initnode,initnode+Nstate+toy)
199                     initnode=initnode+Nstate
200
201
202
203             d = Nepoch*Nstate+s+offset
204             branchlist=defaultdict(list)
205             g.MakePaths(s, d)
206
207             index1=0
208
209
210             for row in g.branchlist:
211
212                 for element in g.branchlist[row]:
213                     branchlist[index1].append(element% Nstate)
214                 index1=index1+1
215
216             for keyy in branchlist:
217                 checkk=branchlist[keyy]
218                 if checkk.count(s)==2 or checkk.count(s)==Nepoch+1:
219                     for element in checkk:
220                         revisedbranchlist[index].append(element)
221                     index=index+1
222
223
224
225             if doplot:
226                 for epoch in list(range(Nepoch)):
227                     plotx1=epoch
228                     plotx2=epoch+1
229                     for key in ...
                            list(range(indexsave,len(revisedbranchlist))):
230                         fromy=revisedbranchlist[key][plotx1]
231                         toy=revisedbranchlist[key][plotx2]
232                         plt.plot([plotx1,plotx2],[fromy,toy],c='k')
233                         plt.plot(plotx1,fromy,'ko')
234                         plt.plot(plotx2,fromy,'ko')
235
236         distancelist=[]
237         for key in revisedbranchlist:
238             temp_list=revisedbranchlist[key]
```

```
239         if temp_list.count(s) != Nepoch+1 and temp_list.count(s) ...
                != Nepoch+2:
240             distance=0
241             for ind in list(range(len(temp_list)-1)):
242                 distance=distance+ distancecal(trellisoutput[:,0, ...
                        temp_list[ind], ...
                        temp_list[ind+1],0],trellisoutput[:,0,s,s,0])+ ...
                        distancecal(trellisoutput[:,1, ...
                        temp_list[ind],temp_list[ind+1],0], ...
                        trellisoutput[:,1,s,s,0])
243             distancelist.append(distance)
244         else:
245             distancelist.append(1000)
246
247
248     minimumpath=revisedbranchlist[distancelist. ...
            index(min(distancelist))]
249
250     if doplot:
251         for epoch in list(range(len(minimumpath)-1)):
252             plotx1=epoch
253             plotx2=epoch+1
254
255             fromy=minimumpath[plotx1]
256             toy=minimumpath[plotx2]
257             plt.plot([plotx1,plotx2],[fromy,toy],c='g')
258             plt.plot(plotx1,fromy,'g')
259             plt.plot(plotx2,fromy,'g')
260
261     alldistancelist=[]
262     bestmindistance=1000
263     for key1 in revisedbranchlist:
264         ref_path=revisedbranchlist[key1]
265         for key2 in revisedbranchlist:
266             current_path=revisedbranchlist[key2]
267             if (current_path != ref_path and len(current_path)== ...
                    len(ref_path)):
268                 distance=0
269                 for ind in list(range(len(current_path)-1)):
270                     distance=distance+distancecal( ...
                            trellisoutput[:,0, current_path[ind], ...
                            current_path[ind+1],0], ...
                            trellisoutput[:,0, ...
                            ref_path[ind],ref_path[ind+1],0]) \
271                     +distancecal( trellisoutput[:,1, ...
                            current_path[ind], ...
                            current_path[ind+1],0], ...
                            trellisoutput[:,1, ref_path[ind], ...
                            ref_path[ind+1],0])
272                 if (distance<bestmindistance):
273                     bestmindistance=distance
274                     final_reference_path=ref_path
275                     final_current_path=current_path
276
277
278
```

```
279
280              loopcontrol=len(revisedbranchlist)
281
282  if doplot:
283      for epoch in list(range(len(final_reference_path)-1)):
284                  plotx1=epoch
285                  plotx2=epoch+1
286
287                  fromy=final_reference_path[plotx1]
288                  toy=final_reference_path[plotx2]
289                  plt.plot([plotx1,plotx2],[fromy,toy],c='b')
290                  plt.plot(plotx1,fromy,'b')
291                  plt.plot(plotx2,fromy,'b')
292                  fromy=final_current_path[plotx1]
293                  toy=final_current_path[plotx2]
294                  plt.plot([plotx1,plotx2],[fromy,toy],c='m')
295                  plt.plot(plotx1,fromy,'m')
296                  plt.plot(plotx2,fromy,'m')
297
298  print(min(distancelist))
299  print(bestmindistance)
300  print(final_reference_path)
301  print(final_current_path)
302  plt.figure()
303
304  plt.plot(const_pts[0,:],const_pts[1,:],'ko')
305  for ind in list(range(len(final_reference_path)-1)):
306      initstate_ref=final_reference_path[ind]
307      finalstate_ref=final_reference_path[ind+1]
308      initstate_cur=final_current_path[ind]
309      finalstate_cur=final_current_path[ind+1]
310      print("epoch={}: Blue {}/{} magenta ...
            {}/{}".format(ind,trellisbranch[0,initstate_ref, ...
            finalstate_ref,0],trellisbranch[1,initstate_ref,finalstate_ref,0], ...
            trellisbranch[0,initstate_cur,finalstate_cur,0],trellisbranch[1, ...
            initstate_cur,finalstate_cur,0]))
311
312      print(trellisbranch[0,initstate_ref,finalstate_ref,0],
313          const_pts[:,trellisbranch[0,initstate_ref,finalstate_ref,0]])
314      print(trellisbranch[1,initstate_ref,finalstate_ref,0],
315          const_pts[:,trellisbranch[1,initstate_ref,finalstate_ref,0]])
316      print(trellisbranch[0,initstate_cur,finalstate_cur,0],
317          const_pts[:,trellisbranch[0,initstate_cur,finalstate_cur,0]])
318      print(trellisbranch[1,initstate_cur,finalstate_cur,0], ...
            const_pts[:,trellisbranch[1,initstate_cur,finalstate_cur,0]])
319      x=[const_pts[0,trellisbranch[0,initstate_ref,finalstate_ref,0]], ...
            const_pts[0,trellisbranch[0,initstate_cur,finalstate_cur,0]]]
320      y=[const_pts[1,trellisbranch[0,initstate_ref,finalstate_ref,0]], ...
            const_pts[1,trellisbranch[0,initstate_cur,finalstate_cur,0]]]
321      plt.plot(x,y, 'r')
322      x=[const_pts[0,trellisbranch[1,initstate_ref,finalstate_ref,0]], ...
            const_pts[0,trellisbranch[1,initstate_cur,finalstate_cur,0]]]
323      y=[const_pts[1,trellisbranch[1,initstate_ref,finalstate_ref,0]], ...
            const_pts[1,trellisbranch[1,initstate_cur,finalstate_cur,0]]]
324      plt.plot(x,y, 'g--')
325
```

```
326  for i in list(range(len(symbollist))):
327      plt.annotate(i, (const_pts[0,i]+.07,const_pts[1,i]+.07))
```