ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Τμήμα Ηλεκτρονικών Μηχανικών & Μηχανικών Η/Υ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

“ Εφαρμογή του προσαρμοζόμενου χαλαρών αποφάσεων αλγορίθμου (PSP-SSA) σε γρήγορα μεταβαλλόμενους δίαυλους ακραίας δυναμικής “

Γιώργος Σταματάκης

Επιβλέπων : Καθηγητής Μαράς Ανδρέας
Επιτροπή : Καθηγητής Πατεράκης Μιχάλης
Καθηγητής Διγαλάκης Βασίλης

Χανιά , Δεκέμβρης 2000

ΘΕΜΑ

¨Εφαρμογή του προσαρμοζόμενου χαλαρών αποφάσεων αλγόριθμου (PSP-SSA)

σε γρήγορα μεταβαλλόμενους δίαυλους ακραίας δυναμικής¨

ΠΕΡΙΛΗΨΗ

Η αποδοτική μετάδοση πληροφορίας μέσω γρήγορα μεταβαλλόμενων διαύλων ακραίας δυναμικής αποτελεί μια διαρκή πρόκληση στο χώρο των τηλεπικοινωνιών και αντικείμενο συνεχούς έρευνας. Στην εργασία αυτή χρησιμοποιείται ένας αλγόριθμος που προέρχεται από την οικογένεια των MAP αλγορίθμων, ο PSP-SSA, για την αποκωδικοποίηση συμβόλων πληροφορίας που έχουν παραμορφωθεί από διασυμβολική παρεμβολή και λευκό γκαουσσιανό θόρυβο κατά τη διάδοση τους σε δίαυλο με Rayleigh διαλλείψεις. Προκειμένου να εκτιμήσουμε την απόδοση του αλγορίθμου PSP-SSA προσομοιώθηκε ένα τηλεπικοινωνιακό σύστημα που περιλαμβάνει κωδικοποίηση, διαστρωμάτωση και QPSK διαμόρφωση της μεταδιδόμενης πληροφορίας. Η απόδοση του PSP-SSA συγκρίνεται μ'αυτή του ήδη εφαρμοσμένου αλγόριθμου PSP-MLSE, όταν γίνεται χρήση του τελευταίου στο ίδιο τηλεπικοινωνιακό σύστημα.

Γιώργος Σταματάκης

Επιβλέπων : Καθηγητής Μαράς Ανδρέας
Επιτροπή : Καθηγητής Πατεράκης Μιχάλης
Καθηγητής Διγαλάκης Βασίλης

Ημερομηνία : Δευτέρα 18/12/2000
Ώρα : 12:00
Αίθουσα : Ε 33.01

Στους γονείς μου.

Ευχαριστίες

Θα ήθελα να εκφράσω την εκτίμησή μου και τις θερμές μου ευχαριστίες στον επιβλέποντα καθηγητή κ. Ανδρέα Μαρά, που μου ανέθεσε την ενδιαφέρουσα αυτή εργασία και με στήριξε καθ' όλη την διάρκεια της προσπάθειάς μου. Επίσης, θα ήθελα να ευχαριστήσω τα μέλη της εξεταστικής επιτροπής, τους καθηγητές Πατεράκη Μιχάλη και Διγαλάκη Βασίλη οι οποίοι μέσα από τα μαθήματα τους μου κίνησαν το ενδιαφέρον να ασχοληθώ με τον ταχύτατα αναπτυσσόμενο τομέα των τηλεπικοινωνιών καθώς και για την ανάγνωση της διπλωματικής εργασίας και τις παρατηρήσεις τους.

Θα ήθελα επίσης να εκφράσω ένα μεγάλο ευχαριστώ στην οικογένειά μου για την πολύτιμη συμπαράστασή τους και την ηθική και υλική υποστήριξή τους όλα αυτά τα χρόνια σπουδών στο Πολυτεχνείο Κρήτης.

Σταματάκης Γιώργος

Χανιά,Δεκέμβρης 2000

<u>ΠΕΡΙΕΧΟΜΕΝΑ</u>

ΚΑΤΑΛΟΓΟΣ ΚΥΡΙΟΤΕΡΩΝ ΟΡΩΝ

AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
QPSK	Quadrature Phase Shift Keying
CA-MLSE	Conventional Adaptive Maximum Sequence Likelihood Estimation
DFE	Decision Feedback Equalization
DFSE	Decision Feedback Sequence Estimation
IDM	Inter-Digit Memory
ISI	Inter-Symbol Interference
LMS	Least Mean Square
MLSE	Maximum Sequence Likelihood Estimation
MSE	Mean Square Error
PE	Parameter Estimation
PSP	Per-Survivor Processing
RLS	Recursive Least Squares algorithm
RMI	Residual Metric Information
RSSE	Reduced State Sequence Estimation
SNR	Signal-to-Noise Ratio
SSA	Suboptimum Soft-output Algorithm
VA	Viterbi Algorithm
WMF	Whitened Matched Filter

ΕΙΣΑΓΩΓΗ

Ιστορική Επισκόπηση

Ένα από τα πιο γνωστά προβλήματα στο χώρο των ψηφιακών τηλεπικοινωνιών, που αφορά την σχεδίαση των ψηφιακών δεκτών και έχει απασχολήσει επί δεκαετίες την ακαδημαϊκή κοινότητα είναι το πρόβλημα της ενδοσυμβολικής παρεμβολής (Intersymbol Interference ή ISI).

Όταν ένα σήμα μεταδίδεται μέσα από ένα δίαυλο συχνά υφίσταται αντανακλάσεις. Αν μια κυματομορφή ανακλάται πάνω σε κάποιο αντικείμενο και μετά φτάνει στον δέκτη, τότε έχει διανύσει ένα μακρύτερο μονοπάτι από το απ' ευθείας μονοπάτι από τον πομπό στο δέκτη. Εξ' αιτίας αυτού του φαινομένου το ανακλώμενο σήμα έχει υποστεί μια καθυστέρηση σε σχέση με το απ' ευθείας μεταδιδόμενο σήμα. Το αποτέλεσμα είναι η λαμβανόμενη κυματομορφή να ισούται με το άθροισμα του μεταδιδόμενου σήματος και διάφορων χρονικών μετατοπίσεων αυτού του σήματος, φαινομένου γνωστού ως multipath ή echo. Ειδικά στις κινητές ψηφιακές επικοινωνίες τα φαινόμενα διάδοσης εξαρτώνται σημαντικά από το λόγο της διαρκείας ενός συμβόλου προς την διάρκεια καθυστέρησης (delay spread) του διαύλου. Η διάρκεια καθυστέρησης μπορεί να θεωρηθεί σαν το μήκος του λαμβανόμενου παλμού όταν μεταδώσουμε ένα κρουστικό παλμό (impulse). Άν μεταδίδουμε δεδομένα σε χαμηλούς ρυθμούς τα δεδομένα μπορούν εύκολα να αναγνωριστούν από τον δέκτη. Αυτό γίνεται γιατί η διάδοση του παλμού δεδομενών λόγω των multipaths έχει ολοκληρωθεί πριν μεταδοθεί ο επόμενος κρουστικός παλμός. Όμως αν αυξήσουμε το ρυθμό μετάδοσης, για καλύτερη εκμετάλλευση του διαύλου, θα φτάσουμε σε ένα σημείο όπου η μετάδοση κάθε συμβόλου δεδομένων επηρεάζει σημαντικά τα διπλανά του σύμβολα δηλαδή οδηγεί στην ενδοσυμβολική παρεμβολή. Χωρίς την χρήση κατάλληλων μεθόδων ισοστάθμισης και εξάλειψης της ISI ο ρυθμός σφαλμάτων στα bits (bit error rate, BER) μπορεί να φτάσει σε μη αποδεκτά επίπεδα.

Ανάμεσα στις διάφορες μεθόδους που αναπτύχθηκαν και που έχουν στόχο την εξάλειψη της ISI βασικότερη είναι αυτή που στηρίζεται στον αλγόριθμο του Viterbi και η οποία έδωσε μια μεγάλη ώθηση στην έρευνα πάνω σε βέλτιστες τεχνικές αναζήτησης με trellis διαγράμματα για μια ποικιλία προβλημάτων [1, 9, 10]. Οι παραπάνω τεχνικές χρησιμοποιούν το κριτήριο μέγιστης πιθανοφάνειας (MLSE : maximum likelihood sequence estimators) και αποδεικνύονται βέλτιστες για την περίπτωση που έχουμε ελεγχόμενη ενδοσυμβολική παρεμβολή. Όμως η ακριβής γνώση ενός διαύλου δεν είναι πάντα δυνατή, και η πιο ευθύς διαφοροποίηση που προτάθηκε είναι η χρήση της αρχιτεκτονικής Viterbi με την προσθήκη ενός εξωτερικού εκτιμητή των άγνωστων παραμέτρων του διαύλου. Έτσι γεννήθηκε ο συμβατικός προσαρμοζόμενος εκτιμητής μέγιστης πιθανοφάνειας (CA-MLSE). Όμως υπάρχουν περιπτώσεις που ο CA-MLSE δεν μπορεί να εφαρμοστεί (π.χ. πολύ μεγάλης έκτασης ενδοσυμβολική παρεμβολή). Μεταξύ πολλών λύσεων που προτάθηκαν για την εξουδετέρωση της ενδοσυμβολικής παρεμβολής ήταν και η λύση επεξεργασίας ανά επιβιώνουσα μετάβαση (Per-Survivor Processing, PSP) που αρχικά προτάθηκε στο [13]. Μια γενίκευση της ιδέας του PSP προτάθηκε ανεξάρτητα αργότερα στο [14] μαζί με ένα ευρύτερο ορισμό των μειωμένης μνήμης καταστάσεων στα διαγράμματα trellis. Ανεξάρτητα επίσης προτάθηκε [30, 31] και η ελαττωμένων καταστάσεων εκτίμηση ακολουθίας (RSSE), που αφορούσε τεχνικές για την μείωση του χώρου των καταστάσεων και PSP υπολογισμό της δημιουργούμενης έλλειψης από την πλήρη σε καταστάσεις περιγραφή. Ωστόσο όλη η παραπάνω δουλειά αφορούσε ένα γνωστό κανάλι το οποίο ήταν κάπως δύσκολο να περιλάβουμε σε μια απ' ευθείας έκδοση του κριτηρίου MLSE.

Στις αρχές της δεκαετίας του '90 είχαμε την εισαγωγή καινούργιων ιδεών για άγνωστα και πιθανώς χρονικά μεταβαλλόμενων καναλιών. Αυτές οι μέθοδοι μπορούν να κατηγοριοποιηθούν με βάση τρία συσχετιζόμενα προβλήματα :

(α)Άγνωστο καθορισμένο κανάλι που αντιμετωπίζεται με μέγιστης πιθανοφάνειας από κοινού εκτίμηση των δεδομένων και του καναλιού [15, 16, 32, 33],

(β)Χρονικά μεταβαλλόμενο αλλά στατιστικά γνωστό κανάλι, τυπικά της γκαουσιανής οικογένειας [17, 34, 35],

(γ)Χρονικά μεταβαλλόμενο πλήρως άγνωστο κανάλι, για το οποίο το προσαρμοζόμενο PSP αρχικά προτάθηκε είτε σαν ειδική τεχνική πάνω σε ένα συγκεκριμένο πρόβλημα

[18, 19, 20, 36, 37, 38, 39], είτε σαν μια γενική αρχή, εφαρμόσιμη σε οποιοδήποτε τέτοιο περιβάλλον [15, 21]. Από τότε η τεχνική PSP έχει επεκταθεί σε διάφορες κατευθύνσεις, έχοντας πρόσφατα βρει μια σταθερή θεωρητική θεμελίωση [23, 24, 40].

Σκοπός της διατριβής

Όπως έγινε φανερό από την παραπάνω ιστορική αναδρομή η ιδέα της τεχνικής PSP έχει επαναπροταθεί ανεξάρτητα από πολλούς ερευνητές μέσα στα προηγούμενα 35 χρόνια με διαφορετικές μορφές κάθε φορά. Όταν ο λόγος σήματος προς θόρυβο (signal to noise ratio, SNR) δεν είναι πολύ μεγάλος ή το κανάλι αλλάζει πολύ γρήγορα όπως στην περίπτωση της κινητής τηλεφωνίας οι συμβατικές μέθοδοι για την ανίχνευση των δεδομένων δεν έχουν πολύ καλή απόδοση. Έτσι δημιουργείται η ανάγκη πιο "έξυπνων" μεθόδων, οι οποίες στόχο έχουν να καλύψουν τις περισσότερες απώλειες των παλιότερων μεθόδων.

Ωστόσο το κέρδος που προσφέρουν οι νέες μέθοδοι δεν έρχεται χωρίς το ανάλογο κόστος. Οι λύσεις αυτές όπως η τεχνική PSP έχουν σαφώς υψηλότερη πολυπλοκότητα και απαιτούν αρκετά μεγάλη υπολογιστική ισχύ τόσο σε ταχύτητα επεξεργασίας όσο και σε μνήμη, πράγμα που καθιστούσε την πρακτική υλοποίηση τους αδύνατη μέχρι πριν από λίγα χρόνια. Το ερώτημα που καλούμαστε να απαντήσουμε είναι ποιά είναι η επίδοση της επεξεργασίας ανά επιβιώνουσα μετάβαση στην περίπτωση που έχουμε πεπερασμένη ISI και ο δίαυλος επικοινωνίας μεταβάλεται με το χρόνο, όταν αυτή συνδυάζεται με τή συμβατική μέθοδο προσαρμοζόμενης MLSE αρχικά και με τον υποβέλτιστο αλγόριθμο χαλαρών αποφάσεων (SSA) στή συνέχεια. Τέλος καταλήγουμε σε συμπεράσματα σχετικά με τη δυνατότητα που προσφέρει ο αλγόριθμος PSP-SSA να επιτύχουμε ικανοποιητική απόδοση με ταυτόχρονη μείωση της πολυπλοκότητας .

Ο σκοπός της παρούσας διατριβής είναι να αναλύσει την τεχνική PSP-SSA όταν εφαρμόζεται σε δίαυλο μεταβλητών παραμέτρων και να τη συγκρίνει με τις ήδη υπάρχουσες και ευρέως διαδεδομένες τεχνικές. Μετά την κατανόηση της γενικής ιδέας της τεχνικής του PSP την οποία παρουσιάζουμε στο κεφάλαιο 1, αναλύεται το διακριτού χρόνου μοντέλο ενός μεταβλητού διαύλου επικοινωνίας. Στην συνέχεια εξετάζουμε το πρόβλημα της εκτίμησης ακολουθίας με βάση τις κλασικές μεθόδους MLSE και του υποβέλτιστου προσαρμοζόμενου αλγορίθμου χαλαρών αποφάσεων (SSA). Παράλληλα δείχνουμε πως η μέθοδος PSP μπορεί να χρησιμοποιηθεί γενικά για τους διάφορους διαύλους με ISI και τονίζουμε τη διαφορά της από τις προηγούμενες μεθόδους.

Στο κεφάλαιο 2 περιγράφουμε πως η μέθοδος PSP εφαρμόζεται για το συγκεκριμένο πρόβλημα δηλαδή ένα δίαυλο μεταβλητών παραμέτρων στο χρόνο. Στο ίδιο κεφάλαιο παρουσιάζουμε ένα ενδεικτικό παράδειγμα εφαρμογής της τεχνικής PSP στο συγκεκριμένο δίαυλο. Ωστόσο λόγω της πολυπλοκότητας του προβλήματος είναι πάρα πολύ δύσκολη η μαθηματική ανάλυση της επίδοσης των παραπάνω αλγόριθμων. Για αυτό το λόγο καταφεύγουμε σε μεθόδους προσομοίωσης της τεχνικής PSP-SSA και PSP-MLSE. Τα αποτελέσματα των προσομοιώσεων για τους αλγόριθμους PSP-SSA και PSP-MLSE καθώς και τα συγκριτικά μεταξύ τους αποτελέσματα παρουσιάζονται στο κεφάλαιο 3.

Στο κεφάλαιο 4 εξάγονται τα τελικά συμπεράσματα που προκύπτουν από την εφαρμογή της μεθόδου. Επίσης γίνεται αναφορά σε μελλοντικές επεκτάσεις και σε περιοχές έρευνας που πρόκειται να συγκεντρώσουν το ενδιαφέρον στο άμεσο μέλλον.

ΚΕΦΑΛΑΙΟ 1^{ο}

ΕΠΕΞΕΡΓΑΣΙΑ ΑΝΑ ΕΠΙΒΙΩΝΟΥΣΑ ΜΕΤΑΒΑΣΗ (PSP)

1.1 ΓΕΝΙΚΑ ΓΙΑ ΤΗΝ ΤΕΧΝΙΚΗ PSP

Είναι γνωστό ότι η μέθοδος MLSE αποτελεί τη βέλτιστη στρατηγική αποκωδικοποίησης μιας ακολουθίας δεδομένων η οποία έχει κωδικοποιηθεί και μεταδοθεί μέσα από ένα δίαυλο με διάχυση (dispersive channel) και θόρυβο, κάτω βέβαια από την υπόθεση ότι ο δέκτης γνωρίζει ακριβώς τις παραμέτρους που χαρακτηρίζουν το κανάλι αυτό. Η διαδοχή του κωδικοποιητή και του καναλιού μπορούν να περιγραφούν από μια μηχανή πεπερασμένων καταστάσεων με αντίστοιχες καταστάσεις και αντίστοιχα διαγράμματα trellis. Σαν αποτέλεσμα ο δέκτης πραγματοποιεί εκτίμηση μέγιστης πιθανοφάνειας πάνω στην ακολουθία των δεδομένων με την βοήθεια του αλγόριθμου Viterbi ο οποίος αναζητά το μονοπάτι ελαχίστου κόστους στο συνδυαστικό διάγραμμα trellis της ISI και του χρησιμοποιούμενου κώδικα.

Εφόσον όμως ο αριθμός των καταστάσεων αυξάνει εκθετικά με το μήκος της κρουστικής απόκρισης του καναλιού, υψηλής πολυπλοκότητας αποκωδικοποιητές διαγραμμάτων trellis θα απαιτούνταν για την υλοποίηση του βέλτιστου αλγόριθμου. Γι' αυτό το λόγο αυτό μειωμένης πολυπλοκότητας αποκωδικοποιητές trellis έχουν προταθεί οι οποίοι βασίζονται στην ισοστάθμιση με ανάδραση απόφασης (DFE ή Decision Feedback Equalization). Η τεχνική αυτή περιλαμβάνει το περιορισμό της κρουστικής απόκρισης του διαύλου όπως θεωρήθηκε στο συνδυαστικό διάγραμμα trellis και στην ακύρωση της υπολειπόμενης ISI με την χρήση δοκιμαστικών αποφάσεων.

Μια εναλλακτική τεχνική σε σχέση μ' αυτή την κλασσική μπορεί να προκύψει με την ενσωμάτωση του μηχανισμού ανάδρασης απόφασης μέσα στον ίδιο αποκωδικοποιητή Viterbi. Αυτό πρωτοεμφανίστηκε στην προσπάθεια να επεκτείνουμε την κλασσική δομή των MLSE αποκωδικοποιητών σε περιπτώσεις καναλιών άπειρης ISI, το οποίο φανερά δεν μπορεί να υποστηριχτεί από ένα πεπερασμένο διάγραμμα trellis. Η βασική ιδέα είναι απλή: προσπάθησε να ακυρώσεις τα αποτελέσματα της υπολειπόμενης ISI απ' ευθείας, στον υπολογισμό κάθε μετρικής των μεταβάσεων μέσα στον VA, βασιζόμενος στην ακολουθία δεδομένων που σχετίζεται με την επιβιώνουσα μετάβαση που οδηγεί σ' αυτή την μετάβαση, δηλαδή ουσιαστικά σε επεξεργασία ανά επιβιώνουσα μετάβαση.

Οι τεχνικές αυτές είναι γνωστές ως εκτίμηση ακολουθίας με ανάδραση απόφασης (DFSE) και εκτίμηση ακολουθίας με μειωμένες καταστάσεις (RSSE). Η δεύτερη τεχνική είναι πιο γενική από την άποψη ότι η μείωση της πολυπλοκότητας μπορεί να επιτευχθεί όχι μόνο με την περικοπή της κρουστικής απόκρισης του διαύλου, αλλά και με την μερική απεικόνιση της υπολειπόμενης ISI. Σαν γενικό συμπέρασμα μπορούμε να πούμε ότι η αποτελεσματικότητα της ανά επιβιώνουσα DFE οφείλεται στην μείωση της διάδοσης του σφάλματος σε σχέση με τις συμβατικές μεθόδους, οι οποίες στηρίζονται σε δοκιμαστικές αποφάσεις.

Η ιδέα της ακύρωσης της υπολειπόμενης ISI με βάση μια συγκεκριμένη ακολουθία επιβιωνουσών δεν περιορίζεται μόνο στην μείωση της πολυπλοκότητας αποκωδικοποίησης αλλά μπορεί να επεκταθεί σε πολλές άλλες περιπτώσεις. Είναι δυνατό να δούμε αυτήν την ιδέα σαν μια πιο γενική αρχή η οποία μπορεί να εφαρμοστεί όταν οι μετρικές μετάβασης στον αλγόριθμο Viterbi επηρεάζονται από κάποιο βαθμό αβεβαιότητας και η οποία μπορεί να εξαλειφθεί ή να μειωθεί με τεχνικές εκτίμησης οδηγούμενες από τα δεδομένα. Τυπικά αυτή η αβεβαιότητα οφείλεται σε ατελή γνώση κάποιων παραμέτρων του διαύλου, όπως για παράδειγμα η φάση του φέροντος ή η ίδια η κρουστική απόκριση του καναλιού. Όλες οι παραπάνω περιπτώσεις μπορούν να αντιμετωπιστούν με μια ενιαία στρατηγική γνωστή με τον όρο PSP. Όπως και στην περίπτωση της μειωμένης πολυπλοκότητας αποκωδικοποιητών, η PSP είναι μια αποτελεσματική εναλλακτική λύση στην χρήση δοκιμαστικών αποφάσεων για την εκτίμηση των άγνωστων ποσοτήτων λόγω του γεγονότος ότι οι συνέπειες που έχει η διάδοση του σφάλματος μειώνονται σημαντικά.

Επιπρόσθετες δελεαστικές όψεις της κλάσης των PSP αλγορίθμων στην περίπτωση που απαιτείται εκτίμηση των παραμέτρων του διαύλου είναι :

(α) Η εκτίμηση στην τεχνική PSP σχετίζεται με την καλύτερη επιβιώνουσα που προκύπτει από την γνώση των δεδομένων, την οποία μπορούμε να δεχτούμε σαν υψηλής ποιότητας, μηδενικής καθυστέρησης απόφαση, κάνοντας έτσι την τεχνική PSP κατάλληλη για πολύ γρήγορα μεταβαλλόμενα κανάλια.

(β) Εφόσον πολλές υποθετικές ακολουθίες δεδομένων εξετάζονται ταυτόχρονα στην διαδικασία εκτίμησης παραμέτρων, η διαδικασία της τυφλής απόκτησης των άγνωστων παραμέτρων διευκολύνεται ουσιαστικά σε σχέση με τις συμβατικές τεχνικές. (Όταν λέμε τυφλή απόκτηση παραμέτρων εννοούμε ότι δεν χρησιμοποιούμε ακολουθία δεδομένων για την εκπαίδευση του δέκτη.)

Οι τεχνικές PSP μπορούν να εφαρμοστούν σαν διαφορετικές γενικεύσεις του αλγόριθμου VA, όπου περισσότερες της μιας επιβιώνουσας διατηρούνται σε κάθε κατάσταση και μια λίστα από καλύτερα μονοπάτια είναι διαθέσιμη σε κάθε βαθμίδα της αποκωδικοποίησης. Επιπλέον τεχνικές PSP είναι εφαρμόσιμες σε MLSE αλγορίθμους μειωμένης πολυπλοκότητας.

Συμπερασματικά η επεξεργασία ανά επιβιώνουσα μετάβαση (Per Survivor Processing, PSP) παρέχει ένα γενικό πλαίσιο για την προσέγγιση αλγορίθμων εκτίμησης ακολουθίας με βάση το κριτήριο μέγιστης πιθανότητας (MLSE) όταν η παρουσία άγνωστων ποσοτήτων εμποδίζει την εφαρμογή του κλασσικού αλγόριθμου του Viterbi (VA). Η αρχή της PSP προκύπτει από την ιδέα ότι η οδηγούμενη από τα δεδομένα εκτίμηση των άγνωστων παραμέτρων μπορεί να εμφυτευτεί στην δομή του ίδιου του αλγόριθμου VA. Ανάμεσα στις πολυάριθμες εφαρμογές της ιδέας της PSP αναφέρουμε τις παρακάτω :

(α) Προσαρμοζόμενη αποκωδικοποίηση MLSE με την παρουσία άγνωστων παραμέτρων.

(β) Ταυτόχρονη αποκωδικοποίηση διαμόρφωσης κωδικοποιημένης με TCM (Trellis Coded Modulation) με συγχρονισμό φάσης.

(γ) Προσαρμοζόμενη εκτίμηση ακολουθίας ελαττωμένων καταστάσεων (RSSE).

Ουσιαστικά η PSP μπορεί να ερμηνευθεί σαν μια γενίκευση τεχνικών απόφασης με ανάδραση της κατηγορίας RSSE παρουσία άγνωστων παραμέτρων.

1.2 ΤΙ ΕΙΝΑΙ Η ΤΕΧΝΙΚΗ PSP

Την τεχνική PSP μπορούμε να την αντιληφθούμε σαν την κοινή αρχή που διέπει φαινομενικά ανόμοιες διεργασίες όπως βέλτιστη αναγνώριση δεδομένων σε τυχαία διαταραγμένα κανάλια, ψηφιακή αποδιαμόρφωση αναλογικών (FM) σημάτων, μείωση της πολυπλοκότητας στην αποδιαμόρφωση και αποκωδικοποίηση δεδομένων για δίαυλο με μεγάλο βαθμό διάχυσης, ταυτόχρονη αναγνώριση δεδομένων και προσαρμοζόμενο φιλτράρισμα παραμέτρων άγνωστων και ίσως χρονικά μεταβαλλόμενων καναλιών, κτλ. Η χρησιμοποιούμενη διαμόρφωση μπορεί να είναι οποιαδήποτε από τις ακόλουθες: με μνήμη ή χωρίς μνήμη, σταθερής περιβάλλουσας ή όχι, με κωδικοποίηση ή χωρίς κωδικοποίηση, στενού ή διασκορπισμένου φάσματος, ή χωρίς καθόλου διαμόρφωση. Ο δίαυλος μπορεί να διαθέτει μνήμη ή να είναι χωρίς μνήμη , μπορεί να περιλαμβάνει κάποια άλλη παρεμβολή από χρήστες εκτός από τον πανταχού παρών γκαουσιανό λευκό θερμικό θόρυβο ή να υπόκειται μόνο στον τελευταίο και γενικά μπορεί να έχει ή να μην έχει κάποιο από τα τυπικά παραμετρικά μειονεκτήματα που περιγράφονται στην βιβλιογραφία (βλέπε [2]). Ο δέκτης μπορεί να μην ενδιαφέρεται καν για τα ίδια τα δεδομένα ,αλλά ίσως για κάποια άλλη παράμετρο π.χ. για την γωνία άφιξης. Δύο όμως πολύ σημαντικά χαρακτηριστικά πρέπει να είναι παρόντα για την εφαρμογή της μεθόδου PSP :

(α) Να υπάρχει μνήμη ανάμεσα στα κωδικοποιημένα σύμβολα των δεδομένων (Inter-Digit Memory, IDM) δηλαδή ο συνδυασμός των συμβόλων είναι τέτοιος ώστε να υπάρχει παραμετρική μνήμη ανάμεσα στα γειτονικά σύμβολα στην λαμβανόμενη κυματομορφή στο δέκτη (η οποία κάνει αναγκαία μια αναζήτηση μονοπατιού για βέλτιστες ή ημιβέλτιστες αποφάσεις).

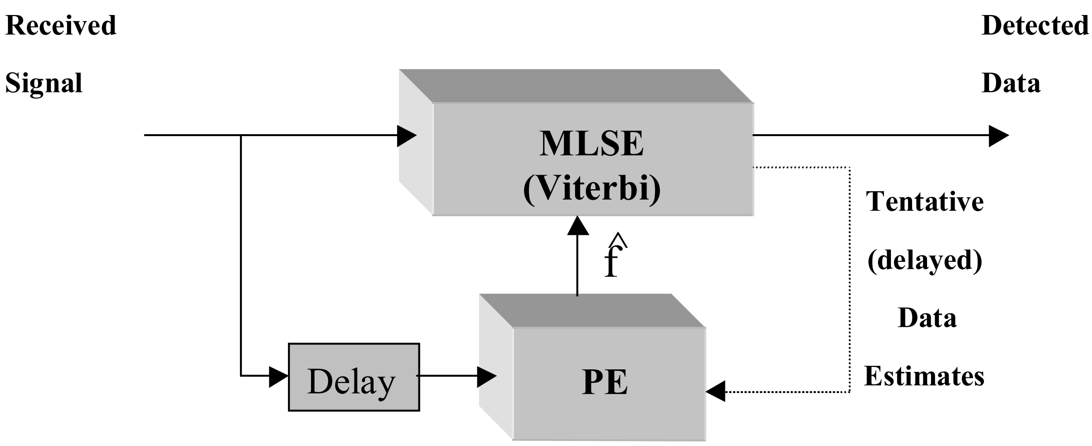
(β) Να υπάρχει μια υπολειπόμενη μετρική πληροφορία (Residual Metric Information RMI) η οποία πρέπει να υπολογιστεί και να τροφοδοτηθεί στον υπολογισμό των μετρικών των κλάδων ενός κλασσικού αλγόριθμου αναζήτησης μονοπατιού. Η πληροφορία αυτή αναφέρεται στην επίδραση των παρελθόντων μεταβάσεων στην παρούσα μετάβαση κατάστασης.

Για να δώσουμε ένα εισαγωγικό παράδειγμα θεωρήστε ένα απλό συνελικτικά κωδικοποιημένο σύστημα. Για το σύστημα αυτό υπάρχει ένα καλά ορισμένο διάγραμμα καταστάσεων και μια trellis περιγραφή (ικανοποιώντας έτσι την απαίτηση IDM που περιγράψαμε παραπάνω) και ένας πεπερασμένης πολυπλοκότητας αλγόριθμος, ο αλγόριθμος Viterbi (VA), ο οποίος παράγει βέλτιστες αποφάσεις πάνω στα μεταδιδόμενα δεδομένα παρουσία γκαουσιανού προσθετικού λευκού θορύβου (AWGN). Αν ξαφνικά συμβεί μια τυχαία μετατόπιση φάσης στο κανάλι, που είναι μια διαδικασία που αφορά την μετατόπιση του στο χρόνο, τότε οι μετρικές των κλάδων στον αλγόριθμο VA πρέπει να ενημερωθούν για την εκτιμώμενη τιμή αυτής της μετατόπισης, δηλαδή πρέπει να τους παραχθεί RMI, διαφορετικά ο υπολογισμός των μετρικών καθίσταται αδύνατος. Δηλαδή και τα δύο χαρακτηριστικά που απαιτεί η PSP είναι παρόντα, γεγονός που επιτρέπει την εφαρμογή της μεθόδου PSP σ' αυτή την περίπτωση.

Είναι άμεσα εφικτό να διευρύνουμε την παραπάνω εφαρμογή και το σύνολο των παραμέτρων που την επηρεάζουν, κάνοντας έτσι φανερό το πόσο γενικό μπορεί να γίνει το μοντέλο PSP .<u>Το μοντέλο PSP απλά στηρίζεται στην ιδέα ότι η RMI πρέπει να γίνει διαθέσιμη ξεχωριστά σε κάθε υπολογισμό μετρικής κλάδου, και η τιμή της πρέπει να υπολογιστεί βασιζόμενη στην επιβιώνουσα που σχετίζεται με τον κόμβο από τον οποίο κάθε τέτοια μετάβαση αρχίζει</u>.

Για να αντιπαραθέσουμε την αρχιτεκτονική της PSP με την παλιότερες παραδοσιακές δομές, θα θυμίσουμε εδώ το κλασικό παράδειγμα της ανίχνευσης δεδομένων με βάση την μέγιστη πιθανοφάνεια, το οποίο ονομάζεται προσαρμοζόμενο MLSE. Στην περίπτωση του προσαρμοζόμενου MLSE οι σχετικές διαδικασίες εκτίμησης των άγνωστων και πιθανώς χρονικά μεταβαλλόμενων παραμέτρων που βρίσκονται εμφυτευμένες στην λαμβανόμενη κυματομορφή εκτός των δεδομένων, γίνονται με ένα διαχωριστικό τρόπο, δηλαδή με υποσυστήματα ξεχωριστά από αυτό που πραγματοποιεί τις αποφάσεις πάνω στα δεδομένα. Αυτές οι παράμετροι μπορεί να είναι : πλάτη κυματομορφών, φάση του φέροντος, χρονισμός συμβόλου, μετατόπιση συχνότητας (Doppler), ή τέλος οι συντελεστές διάχυσης του καναλιού (ISI). Έτσι ένας παραδοσιακός δέκτης συνοδεύεται από υποσυστήματα όπως χρονιστές bits, AGCs, βρόχους κατασταλμένου φέροντος, ανιχνευτές συχνότητας, ισοσταθμιστές καναλιού, συσκευές που κάνουν όλες την ίδια διεργασία : εξάγουν τιμές για τις σχετικές παραμέτρους και τις παρέχουν στο αποκωδικοποιητή για την εκτίμηση των δεδομένων στο προκύπτον (προσεγγιστικά) περιβάλλον προσθετικού λευκού Γκαουσιανού θορύβου(AWGN). Είναι φανερό ότι τα σφάλματα σ' αυτή την εκτίμηση των παραμέτρων (Parameter Estimation ή PE) επηρεάζουν αρνητικά το ρυθμό σφαλμάτων στα bits και την επίδοση του συστήματος γενικότερα. Το γεγονός ότι ένας εναλλακτικός τρόπος στην σχεδίαση του δέκτη μπορεί να επιφέρει σημαντικά κέρδη στη απόδοση του συστήματος και να εφαρμοστεί σε περιπτώσεις που είναι σχεδόν αδύνατο να εφαρμοστούν οι παραδοσιακές μέθοδοι είναι ολοφάνερα μεγάλης σημασίας για τους μηχανικούς που ασχολούνται με την σχεδίαση δεκτών.

Μπορούμε να αναπαραστήσουμε την λειτουργία του κλασσικού προσαρμοζόμενου εκτιμητή ακολουθίας μέγιστης πιθανοφάνειας (CA-MLSE) με το ακόλουθο γενικό διάγραμμα:



Εικόνα 1.1 : CA-MLSE

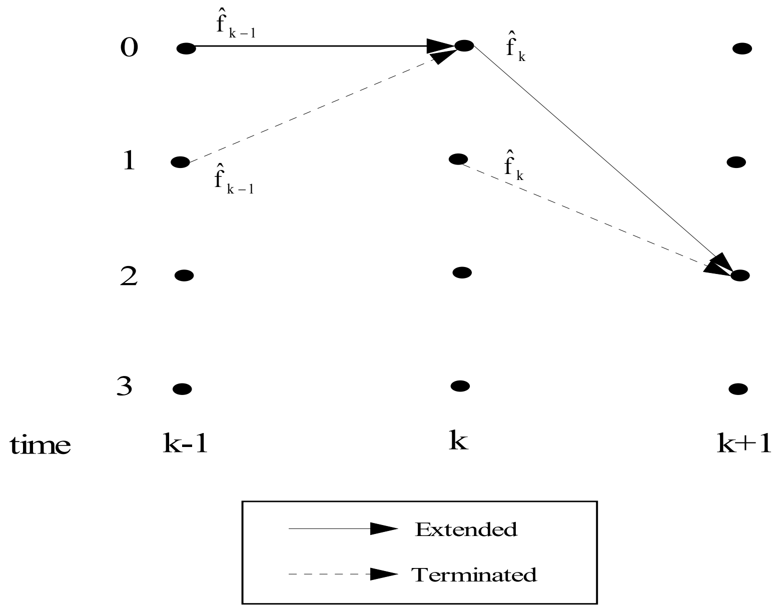
Το PE σύστημα αναπαριστά με ένα γενικό τρόπο όλους τους εξωτερικούς εκτιμητές που απαιτούνται για την απόκτηση των παραμέτρων του καναλιού και ανάλογα με την περίπτωση μπορεί να αποτελείται από πολλά ανεξάρτητα μεταξύ τους υποσυστήματα. Η εικόνα 1.1 απεικονίζει δύο εκδοχές για τον CA-MLSE :

(α) Στην πρώτη περίπτωση (συνεχόμενης γραμμής βέλη) η εκτίμηση των παραμέτρων στηρίζεται μόνο πάνω στα παρατηρούμενα δεδομένα που λαμβάνει ο δέκτης και το εκτιμούμενο διάνυσμα παραμέτρων της PE γίνεται διαθέσιμο στον αποκωδικοποιητή. Μπορούμε να ονομάσουμε αυτή την διάταξη δομή μη οδηγούμενη από τα δεδομένα (non-decision directed structure ή NND).

(β) Στην δεύτερη περίπτωση (απεικονίζεται με διακεκομμένη γραμμή) η οποία περιλαμβάνει μια σύνδεση ανάδρασης από τον ανιχνευτή δεδομένων στο σύστημα PE. Αυτή η σύνδεση παρέχει δοκιμαστικές αποφάσεις πάνω στα δεδομένα, που κάνει ο αποκωδικοποιητής, στον PE για να χρησιμοποιηθούν σαν να επρόκειτο για πραγματικές τιμές δεδομένων με στόχο την απομάκρυνση της αβεβαιότητας των δεδομένων στην παρατήρηση και επικεντρώνοντας το ενδιαφέρον στην εναπομείνασα αβεβαιότητα που υπάρχει για την εκτίμηση των παραμέτρων. Είναι και διαισθητικά φανερό ότι η ακρίβεια στην εκτίμηση των παραμέτρων θα εξαρτάται από την ποιότητα των εκτιμούμενων δεδομένων που τροφοδοτούνται στο PE(π.χ από το ρυθμό σφαλμάτων). Έτσι ένας χαμηλός SNR στην είσοδο θα δημιουργήσει ένα καταστροφικό κύκλο όπου χαμηλής ποιότητας δοκιμαστικές αποφάσεις οδηγούν τον PE και οι λαθεμένοι υπολογισμοί οδηγούν τον αποκωδικοποιητή σε ακόμα χειρότερης ποιότητας αποφάσεις. Αυτή η διάδοση του σφάλματος είναι και το κύριο μειονέκτημα κάθε συστήματος DFE ή CA-MLSE.

Και οι δύο δομές που απεικονίζονται στην εικόνα 1.1 τείνουν να έχουν την ίδια συμπεριφορά όσο αφορά το BER που επιτυγχάνουν.

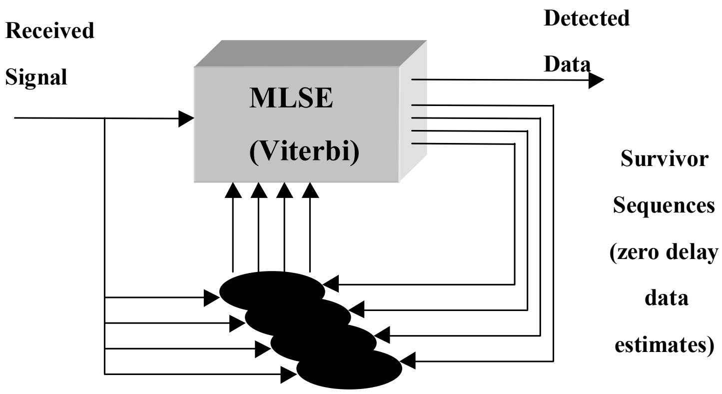
Ένας εναλλακτικός τρόπος να δούμε τον παραπάνω κλασσικό σχεδιασμό είναι να σχεδιάσουμε το διάγραμμα trellis, και να υποθέσουμε ότι ο αποκωδικοποιητής MLSE πραγματοποιεί αναζητήσεις πάνω στο trellis του οποίου οι μετρικές των κλάδων επηρεάζονται από ένα εξωτερικό καθολικό διάνυσμα παραμέτρων όπως φαίνεται στην παρακάτω εικόνα :



Εικόνα 1.2 CA-MLSE

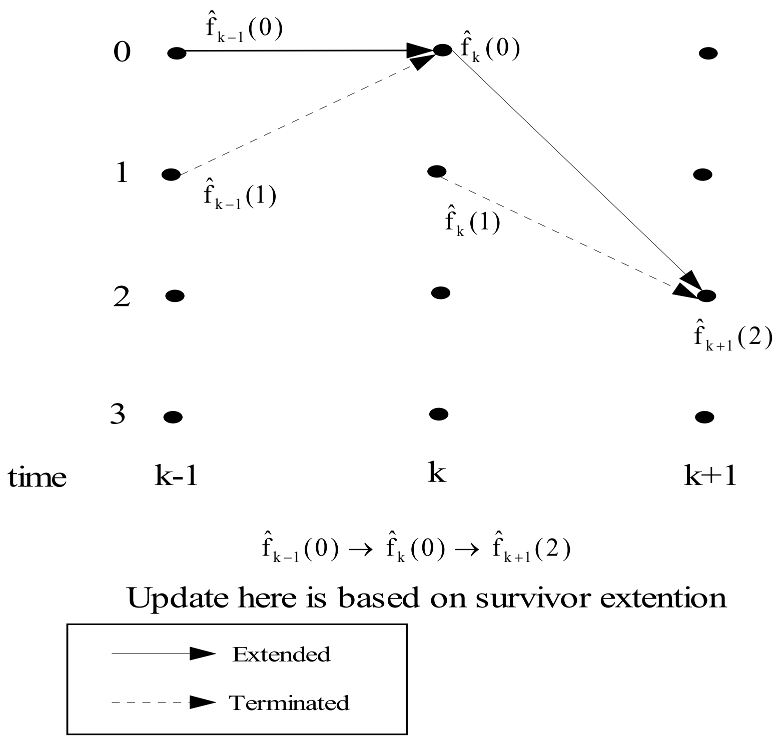
Στο παραπάνω διάγραμμα θεωρούμε ότι το εκτιμώμενο διάνυσμα $\hat{f}$ ενημερώνεται για κάθε σύμβολο, βάση δοκιμαστικών αποφάσεων του MLSE που τροφοδοτούν την ανάδραση PE που περιγράφτηκε παραπάνω.

Ας δούμε τώρα και την αρχιτεκτονική που προτείνεται για την τεχνική PSP. Το λειτουργικό διάγραμμα που μπορεί να απεικονίσει την λειτουργία της PSP φαίνεται στην παρακάτω εικόνα :



Εικόνα 1.3 : PSP Structure

Εδώ η εκτίμηση των παραμέτρων πραγματοποιείται με <u>κατανεμημένο</u> τρόπο, σε αντίθεση με τον συγκεντρωτικό τρόπο που χρησιμοποιείται στη CA-MLSE. Το κάθε διατηρούμενο μονοπάτι στην διαδικασία της αναζήτησης κάθε επιβιώνουσας μετάβασης κρατά και ενημερώνει το δικό του, ατομικό διάνυσμα των εκτιμώμενων παραμέτρων, βασιζόμενο πάνω στην δική του ακολουθία δεδομένων δηλαδή τη συσχετιζόμενη μ' αυτό το μονοπάτι ιστορία δ εδομένων. Αυτό γίνεται περισσότερο κατανοητό βλέποντας το αντίστοιχο διάγραμμα trellis για την PSP που ακολουθεί :



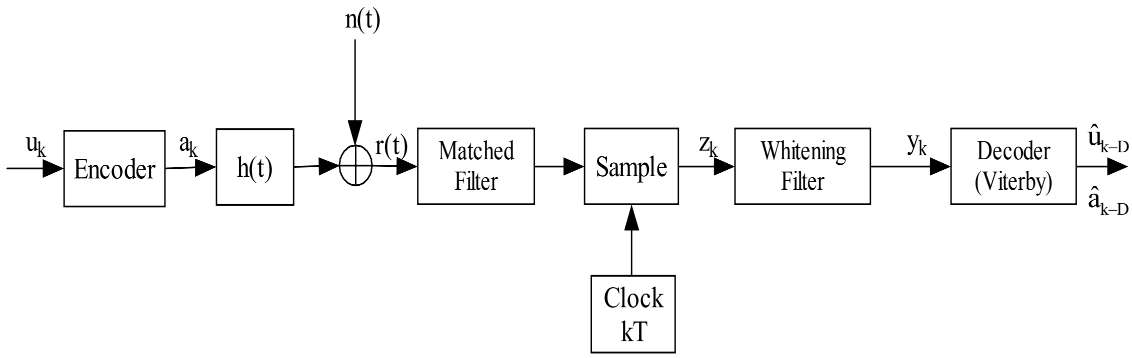
Εικόνα 1.4 PSP

και συγκρίνοντάς το με το αντίστοιχο για την τεχνική CA-MLSE (Εικόνα 1.2). Αυτή είναι η βασική ιδέα του PSP.

Οι πιθανές παραλλαγές της τεχνικής PSP περιορίζονται μονάχα από την φαντασία μας. Αυτές περιλαμβάνουν τον τύπο αναζήτησης (trellis, σειριακό, σύμβολο με σύμβολο, κτλ), τη δομή καταστάσεων / επιβιώνουσων μεταβάσεων (πλήρες σε καταστάσεις αντί μειωμένων καταστάσεων ή “mini-PSP”, μια αντί για πολλές επιβιώνουσες μεταβάσεις για κάθε κατάσταση, κτλ), τον αριθμό των δειγμάτων και τις ενημερώσεις των παραμέτρων ανά σύμβολο (symbol/processing rate ratio), και διάφορες εναλλακτικές επιλογές μοντέλων.

1.3 ΜΟΝΤΕΛΟ ΔΙΑΚΡΙΤΟΥ ΧΡΟΝΟΥ ΓΙΑ ΔΙΑΥΛΟ ΜΕ ISI ΚΑΙ ΙΔΑΝΙΚΟ MLSE

Στην περίπτωση διαύλων περιορισμένου εύρους ζώνης, στους οποίους παρουσιάζεται το φαινόμενο της ISI, είναι βολικό να χρησιμοποιηθεί ένα ισοδύναμο διακριτού χρόνου μοντέλο για το αντίστοιχο αναλογικό (συνεχούς χρόνου) σύστημα. Η κλασική διάταξη που θεωρούμε περιγράφεται από το παρακάτω διάγραμμα :

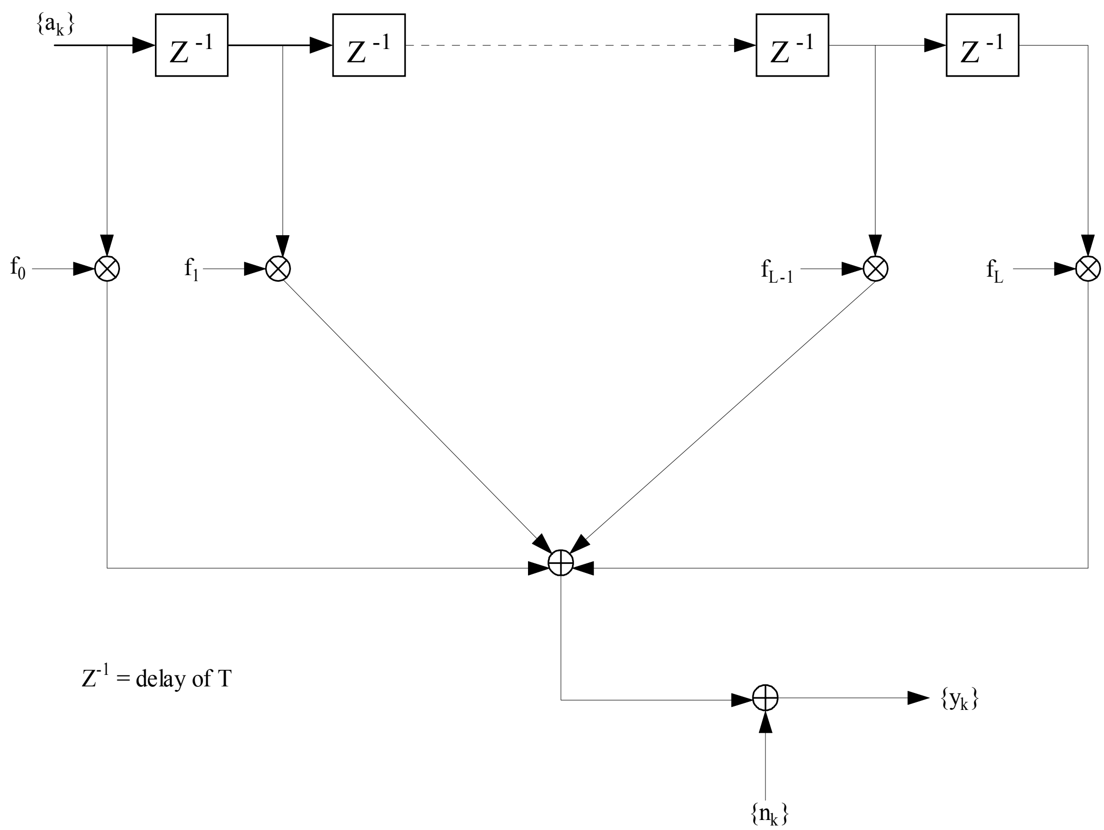


Εικόνα 1.5 : Analog Telecommunication System

Στο παραπάνω διάγραμμα ένα σήμα πληροφορίας u_k τροφοδοτεί τον κωδικοποιητή κάθε T δευτερόλεπτα. Το κωδικό σύμβολο a_k που παράγεται και που ανήκει σε ένα M-αδικό αλφάβητο, μεταδίδεται μέσα σ' ένα μιγαδικό γραμμικό κανάλι που χαρακτηρίζεται από την κρουστική απόκριση h(t) (το φίλτρο αυτό αντιπροσωπεύει την εν σειρά διάταξη του φίλτρου του πομπού και του φυσικού καναλιού). Η μιγαδική περιβάλλουσα του σήματος που λαμβάνει ο δέκτης είναι :

$$r(t) = \sum_{k=-\infty}^{+\infty} a_k \cdot h(t - kT) + \eta(t) \qquad (1\text{-}3\text{-}1)$$

όπου η στοχαστική διαδικασία η(t) είναι το ισοδύναμο βασικής ζώνης λευκού γκαουσιανού θορύβου με φάσμα ισχύος $N_0/2$, ανεξάρτητη από την ακολουθία των δεδομένων. Κάτω από την υπόθεση ότι η h(t) είναι γνωστή, ο βέλτιστος δέκτης αποτελείται από ένα φίλτρο που προσαρμόζεται στην h(t) με κρουστική απόκριση $h^*(-t)$ (Matched Filter) και ένα δειγματολήπτη ρυθμού 1/T δείγματα το δευτερόλεπτο. Το Whitening Filter είναι ένα φίλτρο διακριτού χρόνου και χρησιμοποιείται για την λεύκανση της ακολουθίας θορύβου με επιπλέον φιλτράρισμα της ακολουθίας z_k. Όπως είναι γνωστό η παραπάνω εν σειρά διάταξη (Transmitter/Channel/Matched Filter /Sampler/Whitening Filter) που προηγείται του αποκωδικοποιητή μπορεί να αντικατασταθεί με ένα ισοδύναμο διακριτού χρόνου λευκού θορύβου εγκάρσιο φίλτρο, το διάγραμμα του οποίου φαίνεται στην παρακάτω εικόνα :



Εικόνα 1.6 : Equivalent discrete-time model of interference channel with WGN

Όπως είναι φανερό τα $\{f_i\}_{i=0}^{L}$ (σημεία απομάστευσης, taps) συνιστούν την κρουστική απόκριση του ισοδύναμου διαύλου και το L αντιπροσωπεύει την μνήμη του (έκταση της ενδοσυμβολικής παρεμβολής, δηλαδή πόσα σύμβολα επηρεάζει). Η έξοδος του θα είναι :

$$y_k = \sum_{n=0}^{L} f_n \cdot a_{k-n} + \eta_k \qquad (1\text{-}3\text{-}2)$$

Η προσθετική ακολουθία θορύβου $\{\eta_k\}$ που επηρέαζει την έξοδο του ισοδύναμου διακριτού χρόνου εγκάρσιου φίλτρου είναι λευκή γκαουσιανή ακολουθία θορύβου με μηδενική μέση τιμή και διασπορα $\sigma^2 = N_0$. Η έξοδος $\{y_k\}$ αποτελεί είσοδο στον αποκωδικοποιητή Viterbi.

Όταν η κρουστική απόκριση του διαύλου μεταβάλλεται με το χρόνο, το προσαρμοζόμενο φίλτρο στο δέκτη γίνεται ένα χρονικά μεταβλητό φίλτρο. Στην περίπτωση αυτή οι χρονικές μεταβολλές του καναλιού και του WMF μπορούν να αναπαρασταθούν από ένα φίλτρο διακριτού χρόνου με χρονικά μεταβαλλόμενους συντελεστές. Σαν αποτέλεσμα, έχουμε φαινόμενα χρονικά μεταβλητής ενδοσυμβολικής παρεμβολής, η οποία μπορεί να μοντελοποιηθεί με το φίλτρο της εικόνας 1.6, όπου οι συντελεστές $\{f_i\}_{i=0}^{L}$ μεταβάλλονται με τον χρόνο.

Η διαδικασία MLSE πάνω στην ακολουθία πληροφορίας a_k περιγράφεται πιο εύκολα με βάση την λαμβανόμενη ακολουθία y_k στην έξοδο του δειγματολήπτη. Με την παρουσία ενδοσυμβολικής παρεμβολής που εκτείνεται πάνω σε L+1 σύμβολα (δηλαδή, L σύμβολα που παρεμβάλλονται) το κριτήριο MLSE είναι ισοδύναμο με το πρόβλημα της εκτίμησης της κατάστασης μιας διακριτού χρόνου πεπερασμένων καταστάσεων μηχανής (discrete-time finite-state machine).Η μηχανή πεπερασμένων καταστάσεων είναι σ' αυτή την περίπτωση το ισοδύναμο διακριτού χρόνου κανάλι με συντελεστές $\{f_i\}_{i=0}^{L}$ και η κατάσταση της σε οποιαδήποτε στιγμή στο χρόνο δίνεται από τις L τον αριθμό πιο πρόσφατες εισόδους. Έτσι η κατάσταση την χρονική στιγμή k θα είναι :

$$S_k = (a_{k-1}, a_{k-2}, ..., a_{k-L}) \qquad (1\text{-}3\text{-}3)$$

όπου $a_k = 0$ για $k \le 0$. Άρα για σύμβολα πληροφορίας που ανήκουν σ' ένα Μ-αδικό αλφάβητο το κανάλι θα έχει M^L δυνατές καταστάσεις. Συνεπώς το κανάλι μπορεί να περιγραφεί από ένα διάγραμμα trellis M^L καταστάσεων και ο αλγόριθμος του Viterbi μπορεί να χρησιμοποιηθεί για τον υπολογισμό του πιο πιθανού μονοπατιού μέσα στο διάγραμμα trellis.

Οι μετρικές που χρησιμοποιούνται για την αναζήτηση στο διάγραμμα trellis είναι ανάλογες με τις μετρικές της αποκωδικοποίησης συνελικτικών κωδίκων με χαλαρές αποφάσεις (soft-decision decoding). Ξεκινάμε από τα δείγματα $y_1, y_2, \ldots, y_{L+1}$, από τα οποία υπολογίζουμε τις M^{L+1} μετρικές :

$$\sum_{k=1}^{L+1} \ln p(y_k \mid a_k, a_{k-1}, \ldots, a_{k-L}) \tag{1-3-4}$$

Οι M^{L+1} πιθανές ακολουθίες $a_{L+1}, a_L, \ldots, a_2, a_1$ υποδιαιρούνται σε M^L ομάδες που αντιστοιχούν στις M^L καταστάσεις ($a_{L+1}, a_L, \ldots, a_2$). Οι Μ ακολουθίες σε κάθε ομάδα (κατάσταση) διαφέρουν στο a_1 και αντιστοιχούν σε μονοπάτια στο trellis που ενώνονται σε κάποιο κόμβο. Από τις Μ ακολουθίες σε κάθε μια από τις M^L καταστάσεις, επιλέγουμε την ακολουθία με την μεγαλύτερη πιθανότητα (όσο αφορά το a_1) και αντιστοιχίζουμε στην επιβιώνουσα ακολουθία την παρακάτω μετρική :

$$PM_1 \equiv PM_1(a_{L+1}, a_L, \ldots, a_2) = \max_{a_1} \sum_{k=1}^{L+1} \ln p(y_k \mid a_k, a_{k-1}, \ldots, a_{k-L}) \tag{1-3-5}$$

Οι Μ-1 που απομένουν από τις M^L ομάδες απορρίπτονται. Έτσι έχουμε τις M^L επιβιώνουσες ακολουθίες και τις μετρικές τους.

Όταν λάβουμε το δείγμα y_{L+2} ,οι M^L επιβιώνουσες ακολουθίες επεκτείνονται κατά ένα στάδιο, και οι αντίστοιχες M^{L+1} πιθανότητες υπολογίζονται για τις επεκταμένες ακολουθίες χρησιμοποιώντας τις προηγούμενες μετρικές και την καινούργια αύξηση, η οποία είναι $\ln[p(y_{k+2} \mid a_{k+2}, a_{k+1}, \ldots, a_2)]$. Ξανά οι M^{L+1} ακολουθίες υποδιαιρούνται σε M^L ομάδες που αντιστοιχούν στις M^L πιθανές καταστάσεις

$(a_{L+2},...,a_3)$ και η πιο πιθανή ακολουθία από κάθε ομάδα επιλέγεται, ενώ οι άλλες M-1 ακολουθίες απορρίπτονται. Η διαδικασία που περιγράφτηκε συνεχίζεται με λήψη και των επόμενων δειγμάτων . Γενικά με την λήψη του y_{k+L} δείγματος, οι μετρικές [1]

$$PM_k(\mathbf{a}_{L+k}) = \max_{I_k}[\ln(p(y_{L+k} \mid a_{L+k},...,a_k) + PM_{k-1}(\mathbf{a}_{L+k-1}))] \qquad (1\text{-}3\text{-}6)$$

δίνουν τις πιθανότητες των M^L επιβιωνουσών ακολουθιών. Έτσι καθώς κάθε ένα δείγμα λαμβάνεται, ο αλγόριθμος του Viterbi αφορά πρώτα τον υπολογισμό των M^{L+1} πιθανοτήτων $\ln p(y_{L+k}|a_{L+k},...,a_k) + PM_{k-1}(a_{L+k-1})$ που αντιστοιχούν στις M^{L+1} ακολουθίες οι οποίες αποτελούν την επέκταση των M^L επιβιώνουσων ακολουθιών από το προηγούμενο στάδιο της διαδικασίας. Τότε οι M^{L+1} ακολουθίες υποδιαιρούνται σε M^L ομάδες, με κάθε ομάδα να περιλαμβάνει M ακολουθίες οι οποίες τερματίζουν στο ίδιο σύνολο συμβόλων $a_{L+k},...,a_{k+1}$, και διαφέρουν στο σύμβολο a_k. Από κάθε ομάδα από τις M ακολουθίες επιλέγουμε αυτή με την μεγαλύτερη πιθανότητα, ενώ οι υπόλοιπες M-1 ακολουθίες απορρίπτονται. Έτσι απομένουν M^L ακολουθίες που έχουν μετρικές $PM_k(\mathbf{a}_{L+K})$.

Η καθυστέρηση στην ανίχνευση κάθε συμβόλου πληροφορίας είναι μεταβλητή. Σε πρακτικές υλοποιήσεις η μεταβλητή καθυστέρηση στην αποκωδικοποίηση μπορεί να αποφευχθεί με την μείωση των επιβιωνουσών ακολουθιών που κρατάμε στα d πιο πρόσφατα σύμβολα, όπου d >> L, επιτυγχάνοντας έτσι σταθέρη καθυστέρηση. Στην περίπτωση που οι M^L επιβιώνουσες ακολουθίες στην χρονική στιγμή k διαφέρουν στο σύμβολο a_{k-d}, επιλέγουμε το σύμβολο που φέρει η πιο πιθανή ακολουθία. Η απώλεια στην επίδοση που προέρχεται από αυτή την υποβέλτιστη απόφαση είναι αμελητέα αν η $d \geq 5L$.

---

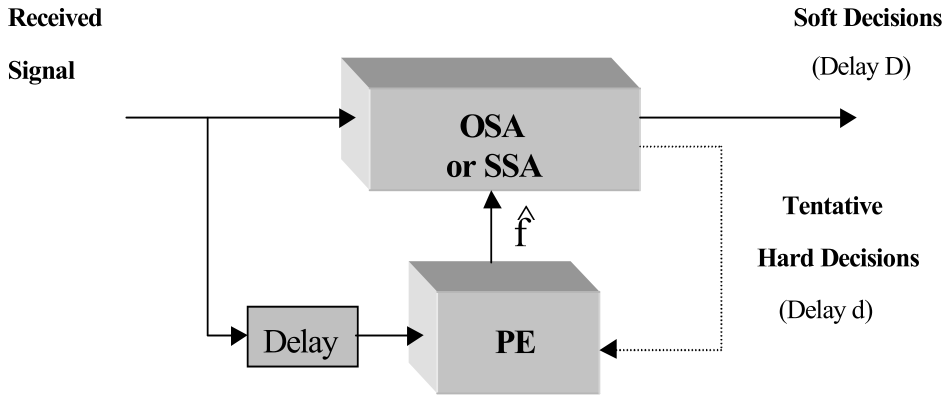
[1] Οι μετρικές $PM_k(\mathbf{a})$, όταν ο προσθετικός θόρυβος είναι γκαουσιανός, συνδέονται άμεσα με τις μετρικές ευκλείδιας απόστασης.

1.4 Ο ΠΡΟΣΑΡΜΟΖΟΜΕΝΟΣ ΥΠΟΒΕΛΤΙΣΤΟΣ ΑΛΓΟΡΙΘΜΟΣ ΧΑΛΑΡΩΝ ΑΠΟΦΑΣΕΩΝ (SSA)

Η MLSE και γενικά οι μέθοδοι που στηρίζονται στον αλγόριθμο του Viterbi για κανάλια με ISI έχουν υπολογιστική πολυπλοκότητα που αυξάνει εκθετικά με το μέγεθος της κρουστικής απόκρισης του ισοδύναμου διακριτού χρόνου διαύλου. Πράγματι αν το μέγεθος του χρησιμοποιούμενου αλφάβητου των συμβόλων a_k είναι M και ο αριθμός των συμβόλων που συνεισφέρουν στην ISI είναι L, τότε ο αλγόριθμος του Viterbi απαιτεί τον υπολογισμό M^{L+1} μετρικών για κάθε νέο σύμβολο που λαμβάνουμε. Σε πολλούς διαύλους πρακτικού ενδιαφέροντος μια τόσο μεγάλη υπολογιστική πολυπλοκότητα είναι απαγορευτική.

Πρόσφατα [11], ο ανασχηματισμένος Abend & Fritcham αλγόριθμος, που ονομάστηκε βέλτιστος χαλαρών αποφάσεων αλγόριθμος (OSA), έχει υπολογιστική πολυπλοκότητα που αυξάνει μόνο γραμμικά με την παράμετρο της καθυστέρησης. Ο OSA είναι μια βελτιωμένη έκδοση του δεύτερου τύπου MAP(maximum a posteriori probability) αλγόριθμου. Χαρακτηρίζεται από το ότι δίνει βέλτιστες χαλαρές αποφάσεις (επιτυγχάνοντας μεγιστοποίση του MAP κριτήριου) και από το ότι απαιτεί αναδρομή μόνο προς τα εμπρός, μπορεί δηλαδή να χρησιμοποιηθεί για συνεχή αποκωδικοποίηση της εισερχόμενης ακολουθίας συμβόλων.

Ο υποβέλτιστος χαλαρών αποφάσεων αλγόριθμος (SSA) που συναντήσαμε στο [11],παρουσιάζει μια μικρή υστέρηση στα αποτελέσματα σε σύγκριση με τον OSA ενώ καταφέρνει να ξεπερνά τα ανεπιθύμητα χαρακτηριστικά των υπολοίπων MAP αλγορίθμων (συμπεριλαμβανόμενου και του OSA). Συγκεκριμένα δέν απαιτεί τη γνώση της διασποράς του θορύβου και οι υπολογισμοί του γίνονται στο λογαριθμικό πεδίο όπως στον αλγόριθμο Viterbi, αποφεύγοντας έτσι τους υπολογισμούς εκθετικών. Οι πράξεις που εκτελούνται είναι κυρίως της μορφής add-compare-select (ACS). Τα παραπάνω χαρακτηριστικά καθιστούν τον SSA ένα εξαιρετικά πρακτικό αλγόριθμο αποκωδικοποίησης.Παρακάτω ακολουθεί ένα διάγραμμα που περιγράφει τη λειτουργία των συμβατικών αλγόριθμων OSA και SSA όπως και στην περίπτωση του CA-MLSE .



Εικόνα 1.7 : Convenional adaptive OSA/SSA algorithms

Οι προσαρμοζόμενοι OSA και SSA αλγόριθμοι λειτουργούν με μία εκτίμηση των άγνωστων παραμέτρων (channel taps) στην θέση των πραγματικών.Η εκτίμηση των παραμέτρων αυτών γίνεται μέσω δοκιμαστικών αποφάσεων από τους OSA και SSA.

Συγκεκριμένα και σύμφωνα με το ισοδύναμο διακριτού χρόνου λευκού θορύβου εγκάρσιο φίλτρο στο διάγραμμα 1.6 και τις εξισώσεις της παραγράφου 1.3, εάν f_κ είναι τα εκτιμώμενα σημεία απομάστευσης (taps) του καναλίου ,που βασίζονται στις δοκιμαστικές αποφάσεις που έγιναν για τα μεταδιδόμενα σύμβολα πέραν της χρονικής στιγμής k-1-d,τότε οι προσαρμοζόμενοι OSA και SSA αλγόριθμοι εφαρμόζονται χρησιμοποιώντας τις εξής μετρικές για την αναζήτηση στο διάγραμμα trellis:

$$m_m\left(\xi_k \mid \hat{f}_k\right) = p\left(y_k \mid \xi_k, \hat{f}_k\right) p\left(S_{k+1} \mid S_k\right), \qquad (1-4-1)$$

(multiplicative metric)

$$m_a\left(\xi_k \mid \hat{f}_k\right) = -\gamma_2 \ln\left[m_m\left(\xi_k \mid \hat{f}_k\right)\right] + \gamma_1, \qquad (1-4-2)$$

(additive metric)

όπου ξ_κ οι μεταβάσεις από την μία κατάσταση του trellis στην άλλη $\xi_\kappa=(\alpha_\kappa,S_\kappa)=(\alpha_\kappa,\ldots\alpha_{\kappa-L})$ και γ_1 , γ_2 κατάλληλα επιλεγμένες σταθερές.

Η εκτίμηση των παραμέτρων του καναλιού γίνεται μέσω της σχέσης

$$\hat{f}_{k+1} = h\left(\hat{f}_k, \hat{a}_{k-d}, y_{k-d}\right) \qquad (1-4-3)$$

Η συνάρτηση h(˙) αντιπροσωπεύει τις μεθόδους LMS,RLS,Kalman,κτλ.

$$m_a(\xi_\kappa) = -\gamma_2 \ln\left[m_m(\xi_k)\right] + \gamma_1 \qquad (1-4-4)$$

$$m_a(\pi_k) = \sum_{i=1}^{k} m_a(\xi_k) \qquad (1-4-5)$$

$$m_a(S_k) = \min\left[m_a(\pi_k) \mid S_k\right] \qquad (1-4-6)$$

$$m_a(S_k, a_{k-\delta}) = \min\left[m_a(\pi_k) \mid S_k, a_{k-\delta}\right] \quad (\mathrm{L} < \delta < \mathrm{D}) \qquad (1-4-7)$$

Η εφαρμογή του προσαρμοζόμενου υποβέλτιστου χαλαρών αποφάσεων αλγόριθμου (SSA) γίνεται χρησιμοποιώντας τις παρακάτω εξισώσεις :

όπου το “μονοπάτι” π_k ορίζεται ως $\pi_k=(\xi_1,\ldots.\xi_k)$.Ο “σκληρός” επιζών (hard survivor) $H(S_k)$ και ο “χαλαρός” επιζών (soft survivor) $G(S_k)$ ορίζονται από :

$$H(S_k) = (\hat{a}_{k-D},\ldots,\hat{a}_{k-L-1}) \qquad (1-4-8)$$

$$\mathrm{G}(\mathrm{S}_k) = \left[g_{i,j}(S_k)\right] = \left[m_a(S_k, a_{k-L-i} = a_j)\right] \qquad [\text{matrix}] \qquad (1-4-9)$$

$$i = 1,\ldots,D-L \quad \mathrm{j} = 1,\ldots,\mathrm{m}-1$$

Εδώ $\hat{a}_{k-L-I}$ είναι η ‘σκληρής απόφασης εκτίμηση του α_{k-L-I} και η ι-οστή σειρά του $G(S_k)$ περιέχει τις m-1 μετρικές m_α (S_k,α_{k-L-i}). Οι εξισώσεις που διαμορφώνουν τις παραπάνω ποσότητες είναι οι εξής:

$$m_a(S_{k+1}) = \min_{S_k} \; m_a(S_k) + m_a(\xi_k) \qquad (1-4-10)$$

$$m_a(S_{k+1}, a_{k-\delta}) = \min_{S_k} \; m_a(S_k, a_{k-\delta}) + m_a(\xi_k) \quad (\mathrm{L} < \delta \leq D) \qquad (1-4-11)$$

$$m_a(S_{k+1}, a_{k-L}) = m_a(S_k) + m_a(\xi_\kappa) \qquad (1-4-12)$$

Το πακέτο πληροφορίας την χρονική στιγμή κ θα είναι:

$$\min_{S_{k+1}} m_a\left(S_{k+1}, a_{k-D} = a_j\right) \quad j = 1, \ldots, m$$

Τέλος,συνοψίζοντας τα στάδια του SSA αλγορίθμου είναι:

1) Υπολογισμός των $m_\alpha(\xi_\kappa)$ για όλες τις μεταβάσεις ξ_κ από την (1-4-4).
2) Για κάθε κατάσταση S_{k+1}:

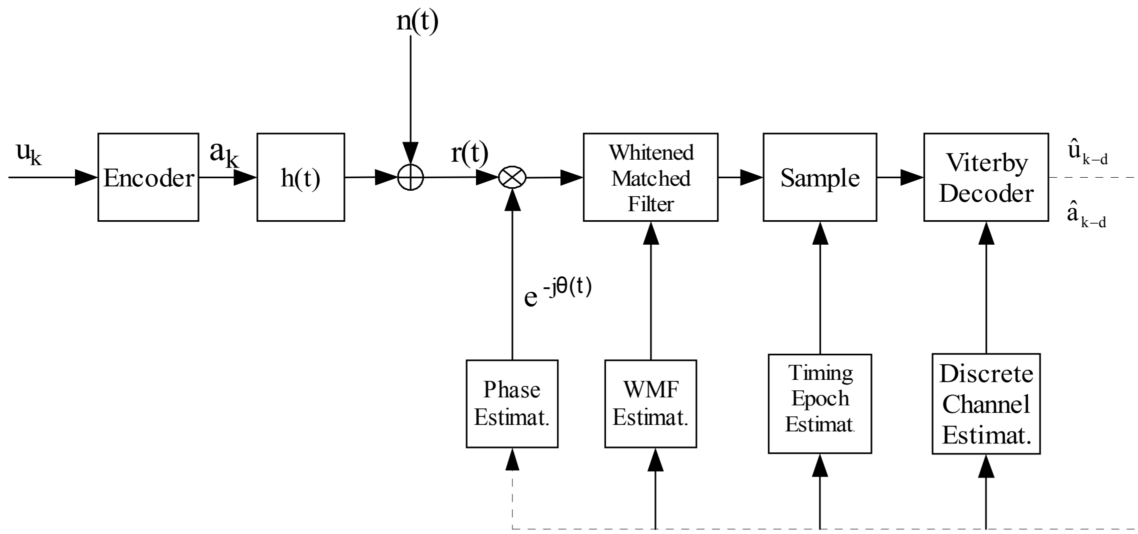
   α) Υπολόγισε τα $m_\alpha\,(S_{k+1})$ από την (1-4-10).

   β) Υπολόγισε τον $H(S_{k+1})$ όπως ακριβώς στον Viterbi αλγόριθμο.

   γ) Υπολόγισε το $G(S_{k+1})$ από την (1-4-11).
3) Υπολόγισε το πακέτο πληροφορίας από την τελευταία σειρά του $G(S_{k+1})$.
4) Μετακίνησε τις σειρές του $G(S_{k+1})$ κατά μία και συμπλήρωσε την πρώτη σειρά με το $m_\alpha(S_{k+1},\alpha_{k-L})$ από την (1-4-12).

1.5 Η ΜΕΘΟΔΟΣ ΕΚΤΙΜΗΣΗΣ MLSE ΠΑΡΟΥΣΙΑ ΑΒΕΒΑΙΟΤΗΤΩΝ

Επανερχόμαστε στην συζήτηση της παραγράφου 1.3 όπου είδαμε ότι κάτω από την υπόθεση ότι η h(t) είναι πλήρως γνωστή, ο βέλτιστος δέκτης υλοποιείται από ένα φίλτρο προσαρμοζόμενο στην h(t) (matched filter), ένα δειγματολήπτη ρυθμού $1/T$ και ένα αποκωδικοποιητή Viterbi ο οποίος αναζητά το μονοπάτι ελαχίστου κόστους (με την μικρότερη μετρική) στο διάγραμμα trellis μιας πεπερασμένων καταστάσεων μηχανής, και η οποία μοντελοποιεί την εν σειρά διάταξη του κωδικοποιητή και του διαύλου πάνω στον οποίο γίνεται η μετάδοση. Δυστυχώς η ιδανική αυτή περίπτωση δεν είναι εφικτή πάντα στην πράξη. Η υπερβολική πολυπλοκότητα που προέρχεται από την πλήρη σε καταστάσεις περιγραφή οδηγεί σε υποβέλτιστες δομές δεκτών που στηρίζονται σε απλοποιημένη περιγραφή καταστάσεων. Επίσης η έλλειψη γνώσης ενός συνόλου παραμέτρων του διαύλου ή της ίδιας της h(t), απαιτεί την ταυτόχρονη εκτίμηση των άγνωστων παραμέτρων και των δεδομένων. Η κύρια προσέγγιση στην υποβέλτιστη αποκωδικοποίηση με την παρουσία αβεβαιοτήτων λόγω των παραπάνω προβλημάτων είναι η χρήση εκτιμητών οδηγούμενων από τα δεδομένα. Ένας τέτοιος εκτιμητής είναι και ο συμβατικός προσαρμοζόμενος MLSE εκτιμητής που περιγράφεται από το παρακάτω διάγραμμα :



Εικόνα 1.8 : Conventional MLSE with unknown channel parameters

Για το παραπάνω σύστημα υπάρχει ένα διακριτού χρόνου λευκού θορύβου ισοδύναμο μοντέλο το οποίο χαρακτηρίζει πλήρως το σύστημα που προηγείται του αποκωδικοποιητή Viterbi. Αν συμβολίσουμε με $\boldsymbol{\theta}_k$ το χρονικά εξαρτώμενο διάνυσμα παραμέτρων που αναπαριστά τις πιθανές άγνωστες παραμέτρους του διαύλου (και ίσως την υπολειπόμενη ISI λόγω της μείωσης των καταστάσεων περιγραφής), η κρουστική απόκριση του ισοδύναμου διαύλου περιγράφεται από τους συντελεστές $\{f_i(\boldsymbol{\theta}_k)\}_{i=0}^{L}$ και γενικά εξαρτάται από το διάνυσμα παραμέτρων $\boldsymbol{\theta}_k$, όπου L είναι η μνήμη του διαύλου.

Αν συμβολίσουμε με μ_k μία κατάσταση του διαγράμματος trellis που περιγράφει την ISI τότε αυτή θα καθορίζεται από την παρακάτω εξίσωση :

$$\mu_k = (a_{k-1}, a_{k-2}, ..., a_{k-K}) \tag{1-5-1}$$

όπου K είναι η μειωμένη μνήμη του διαύλου ($K \le L$ με L να συμβολίζει την πραγματική μνήμη του διαύλου), και η K-οστή γραμμή ($a_{k-1}, a_{k-2}, ..., a_{k-K}$) να αναπαριστά την μειωμένη κατάσταση του διαύλου. Αν K=L τότε έχουμε πλήρη σε καταστάσεις περιγραφή.

Αν συμβολίσουμε τις μεταβάσεις κατάστασης με $\mu_k \rightarrow \mu_{k+1}$ που συμβαίνουν με την λήψη κάθε νέου συμβόλου, τότε οι μετρικές μετάβασης (ή μετρικές των κλάδων) την χρονική στιγμή k θα είναι :

$$\lambda(\mu_k \rightarrow \mu_{k+1}) = F[\mu_k \rightarrow \mu_{k+1}, r(t), \theta_k] \qquad (1\text{-}5\text{-}2)$$

Η $F[\cdot]$ δηλώνει την συναρτησιακή εξάρτηση των μετρικών $\lambda(\mu_k \rightarrow \mu_{k+1})$ από την συγκεκριμένη μετάβαση $\mu_k \rightarrow \mu_{k+1}$, από το συνεχούς χρόνου λαμβανόμενο σήμα και από το διάνυσμα των παραμέτρων $\boldsymbol{\theta}_k$. Πλήρης γνώση του διανύσματος $\boldsymbol{\theta}_k$ θα μας επέτρεπε την υλοποίηση του ιδανικού αλγόριθμου MLSE, για την συγκεκριμένη πολυπλοκότητα των καταστάσεων.

Σε πολλά πρακτικά συστήματα το διάνυσμα $\boldsymbol{\theta}_k$ δεν είναι γνωστό και πρέπει να εκτιμηθεί έτσι ώστε να γίνει δυνατός ο υπολογισμός της (1-5-2). Μια κοινή προσέγγιση του προβλήματος αυτού βασίζεται σε οδηγούμενες από τα δεδομένα τεχνικές εκτίμησης των αγνώστων παραμέτρων. Στις τεχνικές αυτές η ακολουθία δεδομένων που βοηθά στην εκτίμηση των $\boldsymbol{\theta}_k$ αποκτάται με τρόπο κατευθυνόμενο από τις δοκιμαστικές (μικρής καθυστέρησης) αποφάσεις στην έξοδο του αποκωδικοποιητή Viterbi. Ας συμβολίσουμε με $\hat{a}_{k-d-1}$ την δοκιμαστική απόφαση στην ακολουθία των δεδομένων την χρονική στιγμή k. Τότε ο οδηγούμενος από τα δεδομένα εκτιμητής παραμέτρων βασιζόμενος στην ακολουθία δοκιμαστικών αποφάσεων $\{\hat{a}_i\}_{-\infty}^{k-d-1}$ και στο λαμβανόμενο σήμα $r(t)$ παρέχει στον αποκωδικοποιητή Viterbi μια εκτίμηση των παραμέτρων :

$$\hat{\theta}_k = G[r(t), \{\hat{a}_i\}_{-\infty}^{k-d-1}] \qquad (1\text{-}5\text{-}3)$$

όπου η $G[\cdot]$ δηλώνει την συναρτησιακή εξάρτηση της εκτίμησης $\hat{\boldsymbol{\theta}}_k$ από το λαμβανόμενο σήμα και την ακολουθία δοκιμαστικών αποφάσεων. Παρατηρούμε ότι μια καθυστέρηση d συμβόλων είναι αναπόφευκτη στην εκτίμηση $\hat{\boldsymbol{\theta}}_k$ σε σχέση με το πραγματικό διάνυσμα παραμέτρων $\boldsymbol{\theta}_k$.

Η κύρια προσέγγιση στην υποβέλτιστη υλοποίηση του MLSE με την παρουσία αβεβαιοτήτων είναι η χρήση της εκτίμησης από την εξίσωση (1-5-3) στον υπολογισμό των μετρικών των κλάδων στην (1-5-2) δηλαδή

$$\lambda(\mu_k \rightarrow \mu_{k+1}) = F[\mu_k \rightarrow \mu_{k+1}, r(t), \hat{\theta}_k] \qquad (1\text{-}5\text{-}4)$$

Συμβολίζοντας με $\Gamma(\mu_k)$ τις επιβιώνουσες μετρικές, το βήμα ενημέρωσης του αλγόριθμου Viterbi θα είναι : Για όλες τις επόμενες καταστάσεις μ_{k+1}, οι συσσωρευμένες μετρικές $\Gamma(\mu_{k+1})$ καθορίζονται με ελαχιστοποίηση πάνω στις τρέχουσες καταστάσεις μ_k, δηλαδή

$$\Gamma(\mu_{k+1}) = \min_{\mu_k}[\Gamma(\mu_k) + \lambda(\mu_k \rightarrow \mu_{k+1})] \qquad (1\text{-}5\text{-}5)$$

Τελικά οι επιβιώνουσες μεταβάσεις που τερματίζουν στις τρέχουσες καταστάσεις επεκτείνονται με την ενσωμάτωση των μεταβάσεων οι οποίες συμφωνούν με την (1-5-5).

1.6 ΣΥΜΒΑΤΙΚΗ ΜΕΘΟΔΟΣ ΠΡΟΣΑΡΜΟΖΟΜΕΝΗΣ MLSE ΒΑΣΙΣΜΕΝΗΣ ΣΤΗΝ ΤΕΧΝΙΚΗ PSP

Σαν εναλλακτική στην παραπάνω κλασική προσέγγιση των υποβέλτιστων εκτιμήσεων με την μέθοδο MLSE παρουσία αβεβαιοτήτων, παραθέτουμε τον υπολογισμό των άγνωστων παραμέτρων με την σχετικά πρόσφατη μέθοδο PSP. Σε αυτή την τεχνική η ακολουθία των δεδομένων που σχετίζεται με κάθε επιβιώνουσα μετάβαση χρησιμοποιείται ως βοηθητική ακολουθία για την εκτίμηση των άγνωστων παραμέτρων. Μια πιο αυστηρή περιγραφή μπορεί να δοθεί αν συμβολίσουμε την ακολουθία των δεδομένων που σχετίζεται με την επιβιώνουσα μετάβαση μιας κατάστασης μ_k ως $\{\hat{a}_i(\mu_k)\}_{i=-\infty}^{k-1}$.Οι ανά επιβιώνουσα μετάβαση εκτιμήσεις του άγνωστου διανύσματος $\boldsymbol{\theta}_k$ που βασίζονται στο βοηθούμενο από τα δεδομένα εκτιμητή $G[\cdot]$ και τις ακολουθίες δεδομένων που συσχετίζονται με το κάθε επιβιώνον μονοπάτι μπορούν να οριστούν σαν $\hat{\boldsymbol{\theta}}(\mu_k)$ σύμφωνα με την εξίσωση :

$$\hat{\boldsymbol{\theta}}(\mu_k) = G[r(t), \{\hat{a}_i(\mu_k)\}_{i=-\infty}^{k-1}] \qquad (1\text{-}6\text{-}1)$$

Αυτές οι ανά επιβιώνουσα μετάβαση εκτιμήσεις χρησιμοποιούνται για τον υπολογισμό των μετρικών των κλάδων σύμφωνα με την παρακάτω εξίσωση :

$$\lambda(\mu_k \to \mu_{k+1}) = F[\mu_k \to \mu_{k+1}, r(t), \hat{\boldsymbol{\theta}}(\mu_k)] \qquad (1\text{-}6\text{-}2)$$

Η αποκωδικοποίηση συνεχίζεται όπως στον κλασσικό αλγόριθμο Viterbi.

Αυτό που προκύπτει σαν συμπέρασμα από αυτού του τύπου την προσέγγιση του MLSE είναι : όποτε η ατελής γνώση ορισμένων ποσοτήτων μας εμποδίζει να υπολογίσουμε κάποια μετρική κλάδου η οποία σχετίζεται με μια συγκεκριμένη μετάβαση με ακριβή και προβλέψιμη μορφή, χρησιμοποιούμε προσεγγίσεις αυτών των ποσοτήτων που βασίζονται στην ακολουθία των δεδομένων που σχετίζεται με την επιβιώνουσα μετάβαση. Αν οποιαδήποτε συγκεκριμένη επιβιώνουσα μετάβαση είναι σωστή (ένα γεγονός με μεγάλη πιθανότητα κάτω από φυσιολογικές συνθήκες λειτουργίας), οι αντίστοιχες εκτιμήσεις θα υπολογιστούν με βάση την σωστή ακολουθία δεδομένων. Εφόσον σε κάθε στάδιο δεν ξέρουμε ποια επιβιώνουσα μετάβαση είναι η σωστή, επεκτείνουμε κάθε επιβιώνουσα μετάβαση βασισμένοι σε υπολογισμούς που στηρίζονται στην συσχετιζόμενη μ' αυτήν ακολουθία δεδομένων. Μπορούμε δηλαδή να πούμε ότι η καλύτερη επιβιώνουσα μετάβαση επεκτείνεται χρησιμοποιώντας την καλύτερη ακολουθία δεδομένων που είναι διαθέσιμη (η οποία είναι η ακολουθία δεδομένων που σχετίζεται μ' αυτή), ανεξάρτητα από την προσωρινή μας άγνοια για το ποιά επιβιώνουσα μετάβαση είναι η καλύτερη. Αυτό έχει σαν αποτέλεσμα μια σημαντική μείωση της διάδοσης σφαλμάτων σε αντιπαράθεση με την κλασσική προσέγγιση. Αυτή η ιδέα, έμφυτη στα DFSE και στα RSSE, μπορεί να γενικευτεί με την παραπάνω έννοια σε πολλούς τύπους αβέβαιου περιβάλλοντος.

Υποθέτοντας ότι ο βοηθούμενος από τα δεδομένα εκτιμητής έχει την ιδιότητα ότι στην απουσία θορύβου και για την σωστή ακολουθία δεδομένων αρωγής να παράγει σωστές εκτιμήσεις των άγνωστων παραμέτρων, τότε η PSP προσέγγιση του αλγόριθμου MSLE θα παρέχει μια σωστή εκτίμηση της ακολουθίας των δεδομένων απουσία θορύβου. Βασισμένοι σε όλα τα παραπάνω, μπορούμε να πούμε ότι ένας αλγόριθμος βασισμένος στο PSP γίνεται ένας ασυμπτωτικά βέλτιστος αλγόριθμος αποκωδικοποίησης για φθίνουσας ισχύος θόρυβο.

1.7 Η ΜΕΘΟΔΟΣ PSP ΣΕ ΣΥΝΔΥΑΣΜΟ ΜΕ ΤΟΝ ΥΠΟΒΕΛΤΙΣΤΟ ΑΛΓΟΡΙΘΜΟ ΧΑΛΑΡΩΝ ΑΠΟΦΑΣΕΩΝ (PSP-SSA)

Σαν εναλλακτική του υποβέλτιστου αλγόριθμου χαλαρών αποφάσεων (SSA) που παρουσιάστηκε στην παράγραφο 1.4, παραθέτουμε τον υπολογισμό των άγνωστων παραμέτρων του καναλιού με την μέθοδο PSP.Είναι γνωστό οτι η τεχνική PSP βασισμένη σε trellis – δομή παρέχει μία επαρκή μέθοδο για την μείωση της πολυπλοκότητας.Αυτό επιτυγχάνεται αποθηκεύοντας και ανανεώνοντας μόνο έναν πεπερασμένο αριθμό μονοπατιών και επομένως εκτιμήσεων του καναλιού.

Επομένως η trellis – δομή του SSA μας επιτρέπει να χρησιμοποιήσουμε το παραπάνω πλεονέκτημα της μεθόδου PSP.Με αυτό τον τρόπο σε κάθε μία από τις M καταστάσεις του trellis συσχετίζεται και μία εκτίμηση του καναλιού, η οποία ανανεώνεται μέσω μιας αναδρομικής διαδικασίας.

Συγκεκριμένα οι αναδρομικές σχέσεις που χρησιμοποιούμε στον PSP – SSA είναι:

$$\mathrm{L}(\mathrm{S}_{k+1}) = \min_{\mathrm{S}_k} \mathrm{L}(\mathrm{S}_k) + \mathrm{m}_a\left(\xi_k \mid \hat{\mathrm{f}}(\mathrm{S}_k)\right) \quad \text{additive metric} \qquad (1\text{-}7\text{-}1)$$

και

$$\mathrm{G}(\mathrm{S}_{k+1}) = \min_{\mathrm{S}_k} \mathrm{G}(\mathrm{S}_k) + \mathrm{m}_a\left(\xi_k \mid \hat{\mathrm{f}}(\mathrm{S}_k)\right) \quad \text{soft-survivor matrix,} \qquad (1\text{-}7\text{-}2)$$

όπου :

$$m_a\left(\xi_k \mid \hat{f}_k\right) = -\gamma_2 \ln\left[m_m\left(\xi_k \mid \hat{f}_k\right)\right] + \gamma_1 \qquad (1\text{-}7\text{-}3)$$

$$m_m\left(\xi_\kappa \mid \hat{f}_k\right) = p\left(y_k \mid \xi_k, \hat{f}_k\right) p\left(S_{k+1} \mid S_k\right) \qquad (1\text{-}7\text{-}4)$$

Το πακέτο πληροφορίας θα δίνεται από την εξής σχέση:

$$\min_{Sk+1}\; g_{D-L,j}\left(S_{k+1}\right) \qquad (1\text{-}7\text{-}5)$$

Η ανανέωση των σημείων απομάστευσης (channel taps) γίνεται όπως και στην περίπτωση του SSA ,σύμφωνα με την σχέση (1-4-3).

Συγκρίνοντας τους δύο αλγόριθμους PSP – SSA και PSP – MLSE μπορούμε να εξάγουμε τα εξής συμπεράσματα.

Οι δύο αλγόριθμοι είναι σχεδόν όμοιοι ,από την άποψη οτι οι μετρικές και οι εκτιμήσεις των παραμέτρων του καναλιού που χρησιμοποιούνται ,ανανεώνονται με τον ίδιο ακριβώς τρόπο.Με άλλα λόγια σε κάθε βήμα k ,οι μετρικές και οι εκτιμήσεις των παραμέτρων του καναλιού ,που έχουν συσχετιστεί με κάθε κατάσταση του trellis ,θα έχουν παρόμοιες τιμές για τους δύο αλγόριθμους.Η μόνη διαφορά τους είναι στον τρόπο που υπολογίζουν τα δεδομένα εξόδου.

Στην περίπτωση του PSP – MLSE ,αποθηκεύονται οι καλύτερες επιβιώνουσες μεταβάσεις κάθε βήματος και τα δεδομένα εξόδου προκύπτουν (με καθυστέρηση μεγαλύτερη από 5L) από την καλύτερη επιβιώνουσα μετάβαση ,δηλαδή από αυτήν με την μικρότερη μετρική.

Από την άλλη μεριά ο PSP – SSA εξάγει χαλαρές αποφάσεις πάνω στα δεδομένα εισόδου με καθυστέρηση L .Οπότε το πλεονέκτημα του PSP – SSA , στον οποίο δεν χρειαζόμαστε μνήμη για να αποθηκεύσουμε τις καλύτερες επιβιώνουσες ματαβάσεις,είναι εμφανές. Επιπλέον στον PSP – SSA δεν απαιτείται περαιτέρω επεξεργασία για τον εντοπισμό της κάθε επιβιώνουσας κατάστασης προκειμένου να αποφασιστεί τι σύμβολο μεταδόθηκε, σε αντίθεση με τον PSP – MLSE .

ΚΕΦΑΛΑΙΟ 2^0

ΕΦΑΡΜΟΓΗ ΤΩΝ ΤΕΧΝΙΚΩΝ PSP-MLSE ΚΑΙ PSP-SSA ΣΕ ΔΙΑΥΛΟ ΠΟΛΥ ΓΡΗΓΟΡΗΣ ΚΑΙ ΑΚΡΑΙΑ ΔΥΝΑΜΙΚΗΣ

2.1 ΟΡΘΟΓΩΝΙΚΗ ΜΕΤΑΛΛΑΓΗ ΜΕΤΑΤΟΠΙΣΗΣ ΦΑΣΗΣ (QPSK)

Η διάταξη διαμόρφωσης QPSK χαρακτηρίζεται από το γεγονός ότι η πληροφορία που μεταφέρεται από το μεταδιδόμενο σήμα εμπεριέχεται στη φάση . Συγκεκριμένα , σε μια κυματομορφή ορθογωνικής μεταλλαγής μετατόπισης φάσης (quadriphase-shift keying ,QPSK) , η φάση του φέροντος λαμβάνει μια από τις τέσσερις δυνατές τιμές , π/4 , 3π/4 , 5π/4 και 7π/4 , όπως φαίνεται από την παρακάτω σχέση :

$$s_i(t) = \begin{cases} \sqrt{\dfrac{2E}{T}} \cos[2\pi f_c t + (2i-1)\dfrac{\pi}{4}], & 0 \le t \le T \\ 0, & \text{otherwise} \end{cases} \qquad (2\text{-}1\text{-}1)$$

όπου i = 1, 2, 3, 4 , E είναι η ενέργεια του μεταδιδόμενου σήματος ανά σύμβολο , T είναι η διάρκεια του συμβόλου και η συχνότητα του φέροντος f_c ισούται με n_c/T για κάποιο σταθερό ακέραιο αριθμό n_c . Κάθε δυνατή τιμή της φάσης αντιστοιχεί σε ένα μοναδικό ζευγάρι bit που ονομάζεται dibit . Έτσι , για παράδειγμα , μπορούμε να επιλέξουμε το παραπάνω σύνολο τιμών της φάσης για να αναπαριστά το παρακάτω σύνολο dibit : 10 , 00 , 01 και 11 .

Χρησιμοποιώντας μια πολύ γνωστή τριγωνομετρική ταυτότητα , μπορούμε να γράψουμε τη παραπάνω εξίσωση στην ισοδύναμη μορφή :

$$s_i(t) = \begin{cases} \sqrt{\dfrac{2E}{T}} \cos[(2i-1)\dfrac{\pi}{4}]\cos(2\pi f_c t) - \sqrt{\dfrac{2E}{T}} \sin[(2i-1)\dfrac{\pi}{4}]\sin(2\pi f_c t), & 0 \le t \le T \\ 0, & otherwise \end{cases}$$

(2-1-2)

όπου i = 1, 2, 3, 4. Βασισμένοι σ'αυτήν την αναπαράσταση , μπορούμε να κάνουμε τις παρακάτω παρατηρήσεις :

1.Υπάρχουν μόνο δύο συναρτήσεις ορθοκανονικής βάσης, $\varphi_1(t)$ και $\varphi_2(t)$, που περιέχονται στην ανάπτυξη του $s_i(t)$ και η κατάλληλη μορφή για τις $\varphi_1(t)$ και $\varphi_2(t)$ ορίζεται από :

$$\varphi_1(t) = \sqrt{\frac{2}{T}} \cos(2\pi f_c t), \qquad 0 \le t \le T \qquad (2-1-3)$$

$$\varphi_2(t) = \sqrt{\frac{2}{T}} \sin(2\pi f_c t), \qquad 0 \le t \le T \qquad (2-1-4)$$

2. Υπάρχουν τέσσερα σημεία πληροφορίας και τα αντίστοιχα διανύσματα σήματος ορίζονται από :

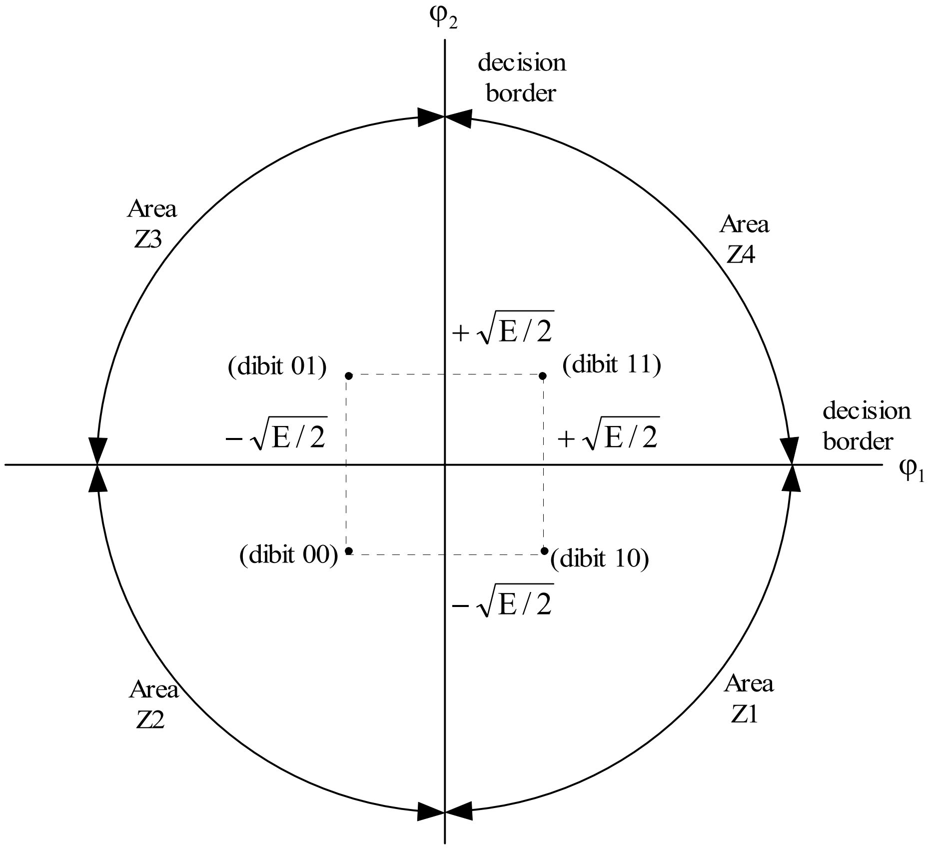
$$\mathrm{s_i} = \begin{bmatrix} \sqrt{\mathrm{E}}\cos[(2\mathrm{i}-1)\dfrac{\pi}{4} \\ -\sqrt{\mathrm{E}}\sin[(2\mathrm{i}-1)\dfrac{\pi}{4} \end{bmatrix}, \qquad \mathrm{i} = 1, 2, 3, 4 \qquad (2\text{-}1\text{-}5)$$

Οι τιμές των στοιχείων των διανυσμάτων σήματος, δηλαδή, s_{i1} και s_{i2}, είναι συγκεντρωμένες στον παρακάτω πίνακα. Οι δύο πρώτες στήλες αυτού του πίνακα δίνουν τα αντίστοιχα dibit και τη φάση του σήματος QPSK .

Dibit εισόδου $0 \le t \le T$	Φάση του σήματος QPSK(rad)	Συντεταγμένες σημείων πληροφορίας	
		s_{i1}	s_{i2}
10	π/4	$+\sqrt{E/2}$	$-\sqrt{E/2}$
00	3π/4	$-\sqrt{E/2}$	$-\sqrt{E/2}$
01	5π/4	$-\sqrt{E/2}$	$+\sqrt{E/2}$
11	7π/4	$+\sqrt{E/2}$	$+\sqrt{E/2}$

Πίνακας 2.1: Τιμές των στοιχείων των διανυσμάτων s_{i1} και s_{i2}.

Επομένως ,ένα σήμα QPSK χαρακτηρίζεται από το ότι έχει δισδιάστατο χώρο σημάτων (δηλαδή N = 2) και τέσσερα σημεία πληροφορίας (δηλαδή M = 4), όπως απεικονίζεται και στο σχήμα που ακολουθεί :



Εικόνα 2.1 : Διάγραμμα χώρου σημάτων για ομόδυνο σύστημα QPSK

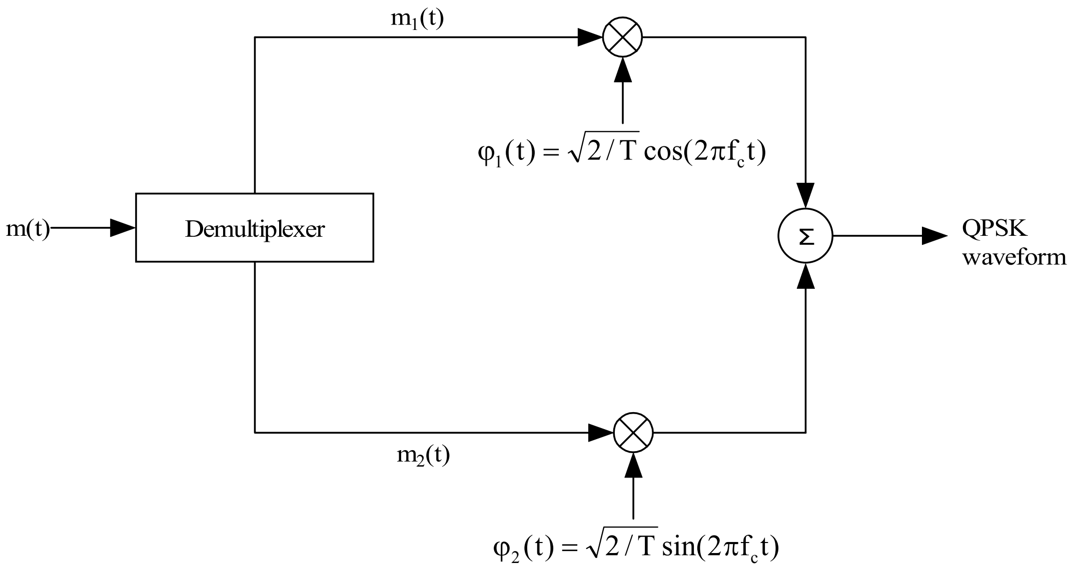
<u>Πομπός QPSK</u>

Η δυαδική ακολουθία εισόδου αναπαριστάνεται στην πολική της μορφή , με τα σύμβολα 1 και 0 να παριστάνονται από $+\sqrt{E_b}$ και $-\sqrt{E_b}$ volt, αντίστοιχα .Αυτή η δυαδική κυματομορφή διαιρείται μέσω ενός αποπολυπλέκτη σε δύο χωριστές δυαδικές κυματομορφές που αποτελούνται από τα περιττά και άρτια αριθμημένα bit της εισόδου. Αυτές οι δύο δυαδικές κυματομορφές συμβολίζονται με $m_1(t)$ και $m_2(t)$. Σημειώνουμε ότι σε οποιοδήποτε διάστημα, τα πλάτη των $m_1(t)$ και $m_2(t)$ είναι ίσα με s_{i1} και s_{i2} ,αντίστοιχα, ανάλογα με το συγκεκριμένο dibit που μεταδίδεται. Οι δύο δυαδικές κυματομορφές $m_1(t)$ και $m_2(t)$ χρησιμοποιούνται για να ένα ζευγάρι ορθογωνικών φερόντων ή συναρτήσεων ορθοκανονικής βάσης :

$$\varphi_1(t) = \sqrt{2/T}\cos(2\pi f_c t) \qquad (2-1-6)$$
$$\varphi_2(t) = \sqrt{2/T}\sin(2\pi f_c t) \qquad (2-1-7)$$

Το αποτέλεσμα είναι ένα ζευγάρι δυαδικών σημάτων PSK , τα οποία μπορουν να φωραθούν ανεξάρτητα εξαιτίας της ορθογωνιότητας των $\varphi_1(t)$ και $\varphi_2(t)$. Τελικά, οι δύο δυαδικές κυματομορφές PSK προστίθενται για να παράγουν την επιθυμητή κυματομορφή QPSK . Σημειώστε ότι η διάρκεια του συμβόλου, T, μιας κυματομορφής QPSK είναι δύο φορές πιο μεγάλη από τη διάρκεια του bit, T_b , της δυαδικής κυματομορφής εισόδου .Αυτό σημαίνει ότι για δοσμένο ρυθμό bit $1/T_b$ μια κυματομορφή QPSK απαιτεί το μισό εύρος ζώνης μετάδοσης από ότι η αντίστοιχη δυαδική κυματομορφή PSK. Ισοδύναμα, για δοσμένο εύρος ζώνης μετάδοσης , η κυματομορφή QPSK μεταφέρει δύο φορές περισσότερα bit πληροφορίας από την αντίστοιχη δυαδική μορφή PSK.

Στη συνέχεια ακολουθεί το δομικό διάγραμμα του πομπού QPSK :

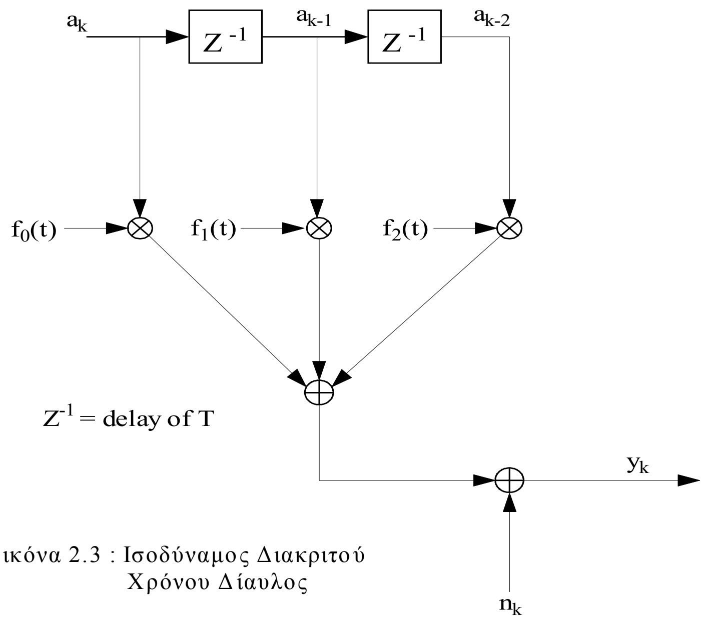


Εικόνα 2.2 : Δομικό διάγραμμα για τον πομπό QPSK

2.2 ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΚΟΥ ΔΙΑΥΛΟΥ (FREQUENCY SELECTIVE RAYLEIGH FADING CHANNEL)

Όταν ένας τηλεπικοινωνιακός δίαυλος εμφανίζει καθυστερήσεις μονοπατιών διάδοσης που είναι μεγάλες συγκρινόμενες με το αντίστροφο του εύρους ζώνης του σήματος, τότε οι συχνότητες του μεταδιδόμενου σήματος υφίστανται διαφορετικές μετατοπίσεις στη φάση κατά μήκος των διαφορετικών μονοπατιών. Καθώς οι διαφορές στις καθυστερήσεις μεταξύ των μονοπατιών αυξάνουν, ακόμα και συχνότητες οι οποίες βρίσκονται κοντά θα εμφανίζουν διαφορές στη μετατόπιση φάσης . Κάτω από αυτές τις συνθήκες το κανάλι εισάγει παραμόρφωση πλάτους και φάσης στη κυματομορφή της πληροφορίας. Ένα τέτοιο κανάλι λέγεται κανάλι επιλεκτικής εξασθένισης συχνότητας (frequency-selective fading channel). Κανάλια που εμφανίζουν τα παραπάνω χαρακτηριστικά,π.χ. σε προαστιακά μακροκυψελικά συστήματα, περιέχουν συνήθως ισχυρούς σκεδαστές, όπως για παράδειγμα πολύ ψηλά κτίρια και μακρινά περιβαλλοντικά χαραχτηριστικά όπως βουνά .

Ένα ικανοποιητικό μοντέλο για το κανάλι επιλεκτικής εξασθένισης συχνότητας είναι το ισοδύναμο διακριτού χρόνου, λευκού θορύβου, εγκάρσιο φίλτρο που αναλύθηκε στην παράγραφο 1.3 . Επαναλαμβάνεται εδώ το διάγραμμα για ευκολία , στη μορφή με την οποία χρησιμοποιήθηκε στις προσομοιώσεις :



Εικόνα 2.3 : Ισοδύναμος Διακριτού Χρόνου Δίαυλος

Ο αριθμός των συμβόλων που συμμετέχουν στη διαμόρφωση της εξόδου του καναλιού, προσδιορίζει τη χρονική διασπορά του διαύλου που συνήθως ονομάζεται multipath spread και στη συγκεκριμένη περίπτωση θεωρείται ίση με T_m= 3T όπου T είναι η περίοδος του συμβόλου. Η αντίστροφή ποσότητα του multipath spread ονομάζεται coherence bandwidth του καναλιού $B_{cb} \approx 1/T_m$.Όταν έχουμε μετάδοση ενός πληροφοριακού σήματος και το εύρος ζώνης του είναι μεγαλύτερο από το coherence bandwidth του διαύλου τότε το κανάλι χαρακτηρίζεται ως frequency selective . Στο μοντέλο που χρησιμοποιήθηκε το εύρος ζώνης του πληροφοριακού σήματος είναι W = 1/T . Και προφανώς ισχύει :

$$W = \frac{1}{T} > \frac{1}{3T} = B_{cb} \qquad (2\text{-}2\text{-}1)$$

Γεγονός που επαληθεύει το ότι προκειται για frequency selective κανάλι .

Τα σημεία απομάστευσης (taps) του παραπάνω μοντέλου είναι συναρτήσεις του χρόνου όπως μπορεί να παρατηρήσει κανείς και από το διάγραμμα. Εν γένη τα taps είναι ασυσχέτιστες μεταξύ τους, μιγαδικές γκαουσιανές στοχαστικές ανελίξεις και μπορούν να γραφούν στη μορφή :

$$f(t) = f_r(t) + jf_i(t) \qquad (2\text{-}2\text{-}2)$$

όπου τα $f_r(t)$ και $f_i(t)$ αντιπροσωπεύουν πραγματικές γκαουσιανές στοχαστικές ανελίξεις. Τα taps μοντελοποιούνται ως γκαουσιανές στοχαστικές ανελίξεις γιατί εκφράζουν την απόκριση του καναλιού σε ένα μεγάλο αριθμό σημάτων που προκύπτουν από σκέδαση. Θεωρούμε επίσης ότι τα $f_r(t)$ και $f_i(t)$ είναι στατικές και στοχαστικά ανεξάρτητες στοχαστικές ανελίξεις. Η υπόθεση αυτή ισχύει γενικά για το μοντέλο απομαστευμένης γραμμής. Τα taps είναι δυνατό να γραφούν και στη μορφή :

$$f(t) = |f(t)|e^{j\varphi(t)} \qquad (2\text{-}2\text{-}3)$$

όπου

$$|f(t)| = \sqrt{f_r^2(t) + f_i^2(t)} \qquad (2\text{-}2\text{-}4)$$

και

$$\varphi(t) = \tan^{-1}\frac{f_i(t)}{f_r(t)} \qquad (2\text{-}2\text{-}5)$$

Σύμφωνα την προηγούμενη αναπαράσταση, αν τα $f_r(t)$ και $f_i(t)$ είναι γκαουσιανές με μηδενική μέση τιμή, τότε το μέτρο $|f(t)|$ χαρακτηρίζεται στατιστικά από την κατανομή πιθανοτήτων Rayleigh ενώ η φάση $\varphi(t)$ είναι ομοιόμορφα κατανεμημένη στο διάστημα [0,2π). Προκειμένου να έχουμε μια πιο ακριβή μοντελοποίηση του φυσικού καναλιού θα πρέπει οι ανελίξεις $f_r(t)$ και $f_i(t)$ να έχουν συναρτήσεις αυτοσυσχέτισης της μορφής:

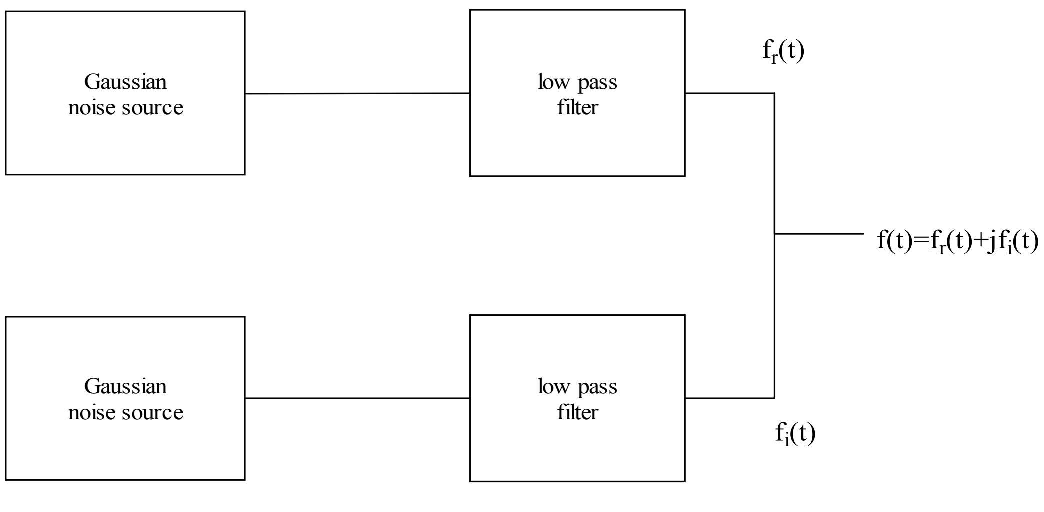
$$R_{f_r f_r}(n) = R_{f_i f_i}(n) = (\Omega_p/2)J_0(2\pi f_d nT) \qquad (2\text{-}2\text{-}6)$$

όπου

$$(\Omega_p/2) = E\{f_r^2\} = E\{f_i^2\} \qquad (2\text{-}2\text{-}7)$$

και f_d είναι η μέγιστη μετατόπιση συχνότητας λόγω του φαινομένου Doppler (η f_d περιγράφει το πόσο γρήγορα μεταβάλονται τα taps του καναλιού), $J_0(.)$ είναι η μηδενικής τάξης συνάρτηση Bessel του πρώτου είδους και T το βήμα της προσομοίωσης . Η παραπάνω συνάρτηση αυτοσυσχέτισης προέρχεται από το μοντέλο της ισοτροπικής σκέδασης δύο διαστάσεων που προτάθηκε από τον Clarke[41].

Προκειμένου να δημιουργηθεί μια γκαουσιανή διαδικάσια που να χαρακτηρίζεται από μηδενική μέση τιμή και την παραπάνω συνάρτηση αυτοσυσχέτισης (μια γκαουσιανή στοχαστική διαδικασία προσδιορίζεται πλήρως από τις δύο αυτές παραμέτρους) χρησιμοποιούμε την ακόλουθη γεννητρια για taps :



Εικόνα 2.4 : Σύστημα Παραγωγής Taps

Το παραπάνω σύστημα ακολουθεί την εξής λογική :

Από τις γεννήτριες λευκού θορύβου παίρνουμε δύο ακολουθίες ανεξάρτητων γκαουσιανών τυχαίων μεταβλητών, μηδενικής μέσης τιμής και διασποράς σ_w^2. Εισάγοντας τα βαθυπερατά φίλτρα επιβάλλουμε μια συσχέτιση μεταξύ των τιμών των ανωτέρων ανεξάρτητων τυχαίων μεταβλητών. Επιλέγοντας κατάλληλα τις παραμέτρους των φίλτρων, τάξη και συχνότητα αποκοπής, καθώς και τη διασπορά του λευκού γκαουσιανού θορύβου είναι δυνατό να προσεγγίσουμε την θεωρητική συνάρτηση αυτοσυσχέτισης όπως αυτή προκύπτει από το μοντέλο του Clarke. Επειδή όμως ο λευκός γκαουσιανός θόρυβος είναι πρακτικά μη υλοποιήσιμος μας είναι άγνωστα τα φασματικά χαραχτηριστικά της ακολουθίας γκαουσιανών τυχαίων μεταβλητών που παράγεται από τη γεννήτρια θορύβου και εισάγεται στο φίλτρο (δεν γίνεται χρήση κάποιου συγκεκριμένου συστήματος δειγματολειψίας και άρα δεν γνωρίζουμε το χρονικό διάστημα που μεσολαβεί μεταξύ των τιμών της ακολουθίας που μας παρέχει η γεννήτρια). Γι'αυτούς τους λόγους δεν θα επιχειρήσουμε μια φυσική ερμηνεία της λειτουργίας του φίλτρου αλλά θα προσδιορίσουμε εξ'ολοκλήρου την ακολουθία γκαουσιανών τυχαίων μεταβλητών εξόδου από τη μέση τιμή της και τη συνάρτηση αυτοσυσχέτισης της , πληροφορία αρκετή άλλωστε λόγω του γκαουσιανού χαραχτήρα της. Αυτό είναι δυνατό να επιτευχθεί μεταβάλλοντας τα χαρακτηριστικά του φίλτρου και τη διασπορά της γεννήτριας θορύβου. Επιπλέον η χρήση βαθυπερατού φίλτρου μας επιτρέπει να παράγουμε μόνο ρητές μορφές πυκνότητας φάσματος ισχύος για την ακολουθία τυχαίων μεταβλητών στην έξοδο του φίλτρου, ενώ η πυκνότητα φάσματος ισχύος ενός Rayleigh fading channel είναι εν γένη μη ρητή και έχει τη μορφή :

$$S_{ff}(f) = \begin{cases} \dfrac{3\Omega_p}{4\pi f_d\sqrt{1-\left(\dfrac{|f-f_c|}{f_d}\right)^2}} & |f-f_c| \le f_d \\ 0 & \text{otherwise} \end{cases} \qquad (2\text{-}2\text{-}8)$$

όπου f_c είναι η συχνότητα του φέροντος. Η προσεγγιστική μέθοδος που χρησιμοποιεί τα δύο βαθυπερατά φίλτρα δίνει ικανοποιητικά αποτελέσματα με πολύ μικρό υπολογιστικό κόστος. Η βέλτιστη μέθοδος πάντως για τη προσομοίωση ενός multipath καναλιού παραμένει η χρήση ακολουθιών τιμών που προκύπτουν από μετρήσεις στο ίδιο το κανάλι.

Επιθυμούμε τα taps του καναλιού να έχουν τα εξής χαρακτηριστικά :

$$m_f = E\{|f|\} = 0 \qquad (2-2-9)$$

$$\sigma_f^2 = E\left\{\left|f - m_f\right|^2\right\} = E\left\{|f|^2\right\} = 1 \qquad (2-2-10)$$

Προκειμένου να επιτύχουμε τις παραπάνω τιμές για τη μέση τιμή και τη διασπορά θα πρέπει οι έξοδοι των βαθυπερατών φίλτρων $f_r(t)$ και $f_i(t)$ να έχουν :

$$m_{f_r} = m_{f_i} = E\{f_r\} = E\{f_i\} = 0 \qquad (2-2-11)$$

$$\sigma_{f_r}^2 = \sigma_{f_i}^2 = Var\{f_r\} = Var\{f_i\} = 1/2 \qquad (2-2-12)$$

Η τιμή του ½ για τη διασπορά των $f_r(t)$ και $f_i(t)$ συνεπάγεται από τις παρακάτω σχέσεις :

$$\left.\begin{array}{l} m_f = m_{f_r} = m_{f_i} = 0 \\ f = f_r + jf_i \end{array}\right\} \Rightarrow E\left\{|f|^2\right\} = E\left\{|f_r|^2\right\} + E\left\{|f_i|^2\right\} = Var\{f_r\} + Var\{f_i\} \qquad (2\text{-}2\text{-}13)$$

Τελικά, για να επιτύχουμε τις παραπάνω τιμές για τα $f_r(t)$ και $f_i(t)$ πρέπει η μέση τιμή και η διασπορά των πηγών λευκού γκαουσιανού θορύβου να προσδιοριστούν κατάλληλα. Συγκεκριμένα η μέση τιμή του λευκού θορύβου m_w σχετίζεται με τη μέση τιμή των $f_r(t)$ και $f_i(t)$ σύμφωνα με την ακόλουθη σχέση :

$$m_{f_r} = m_{f_i} = m_w H(0) \qquad (2\text{-}2\text{-}14)$$

όπου H(0) είναι η τιμή του μετασχηματισμού Fourier της κρουστικής απόκρισης του φίλτρου στο 0. Προφανώς επιλέγοντας $m_w = 0$ επιτυγχάνουμε το επιθυμητό αποτέλεσμα.

Η σχέση που σχετίζει τη διασπορά της ακολουθίας εξόδου με τη διασπορά της ακολουθίας εισόδου του φίλτρου είναι εξαιρετικά πολύπλοκη. Είναι ευκολότερο να προσδιορίσουμε τη διασπορά της εξόδου από την ακόλουθη σχέση :

$$\sigma_{f_r}^2 = \sigma_{f_i}^2 = R_{f_r}(0) = R_{f_i}(0) \qquad (2\text{-}2\text{-}15)$$

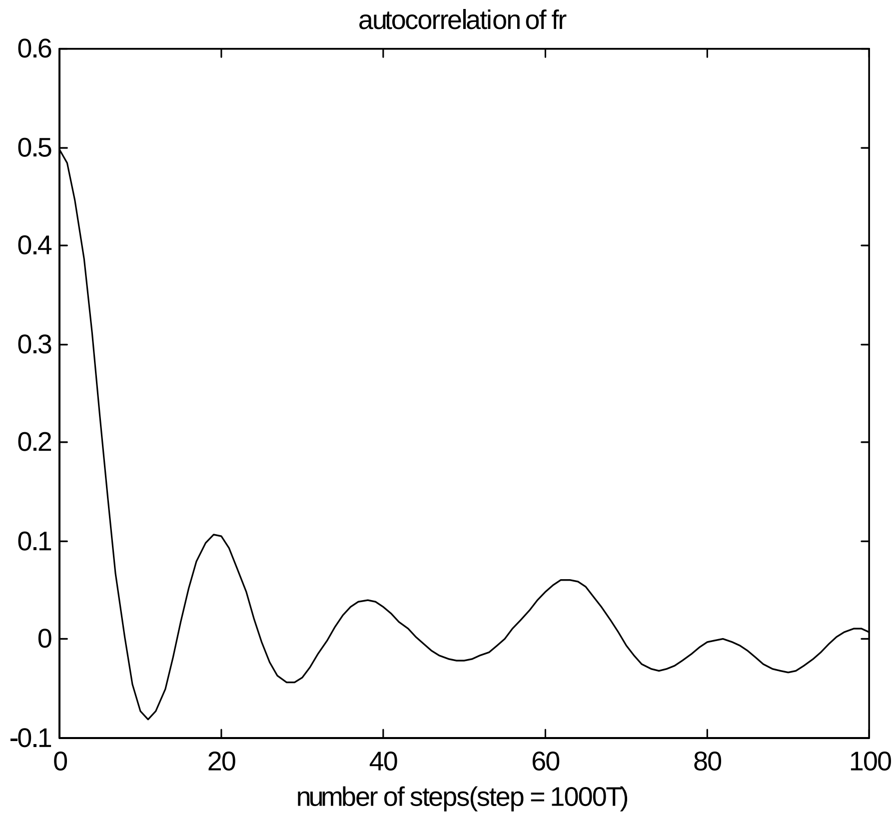
παίρνουμε δηλαδή την τιμή της συνάρτησης αυτοσυσχέτισης της ακολουθίας εξόδου στο 0. Προκειμένου να πετύχουμε την κατάλληλη τιμή για τις διασπορές $\sigma_{f_r}^2$ και $\sigma_{f_i}^2$ μεταβάλουμε την διασπορά του λευκού γκαουσιανού θορύβου και παρατηρούμε τη συνάρτηση αυτοσυσχέτισης της εξόδου.

Στη συνέχεια θα δείξουμε πως εφαρμόστηκαν τα παραπάνω για την παραγωγή των taps στη περίπτωση του πολύ γρήγορα μεταβαλλόμενου καναλιού και στην περίπτωση του ακραία μεταβαλλόμενου καναλιού.

Πολύ γρήγορα μεταβαλλόμενο κανάλι

Το κανάλι αυτό χαρακτηρίζεται από το γινόμενο $f_d T_s = 0.001$, όπου T_s είναι η περίοδος του μεταδιδόμενου συμβόλου και λαμβάνεται στη τιμή 3.69 μsec. ακολουθώντας τις προδιαγραφές του μοντέλου GSM. Το γινόμενο αυτό δηλώνει οτι θα έχουμε αλλαγή των τιμών των taps ανα χίλια σύμβολα. Γι'αυτό τον τύπο του καναλιού προσδιορίστηκε η διασπορά του λευκού θορύβου στην τιμή $\sigma_w^2 = 2.3$ και οι παράμετροι του φίλτρου στις τιμές 1/10 για τη συχνότητα αποκοπής και 10 για το βαθμό του.

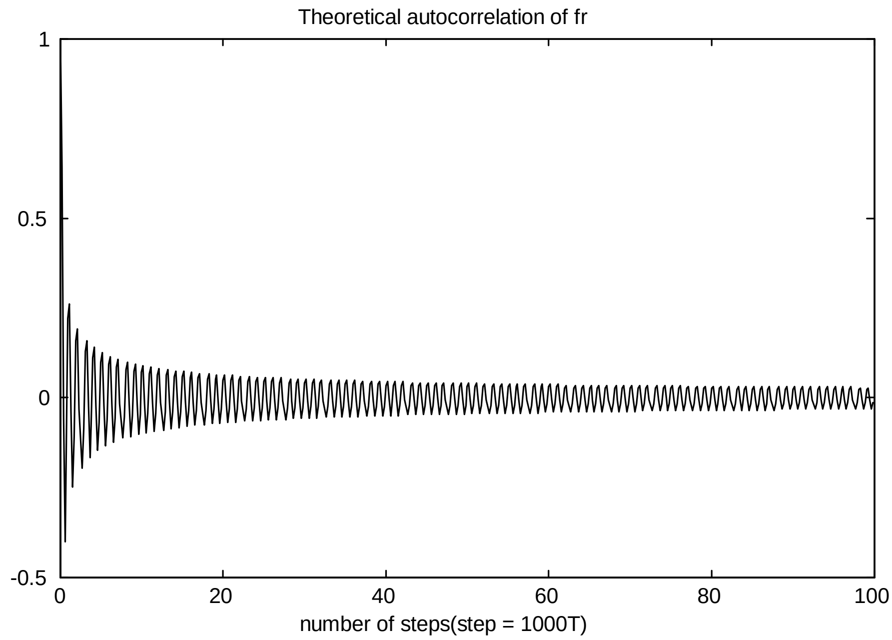
Η συνάρτηση αυτοσυσχέτισης που παίρνουμε με τις παραπάνω τιμές των παραμέτρων για την $f_r(t)$ φαίνεται στο διάγραμμα που ακολουθεί :



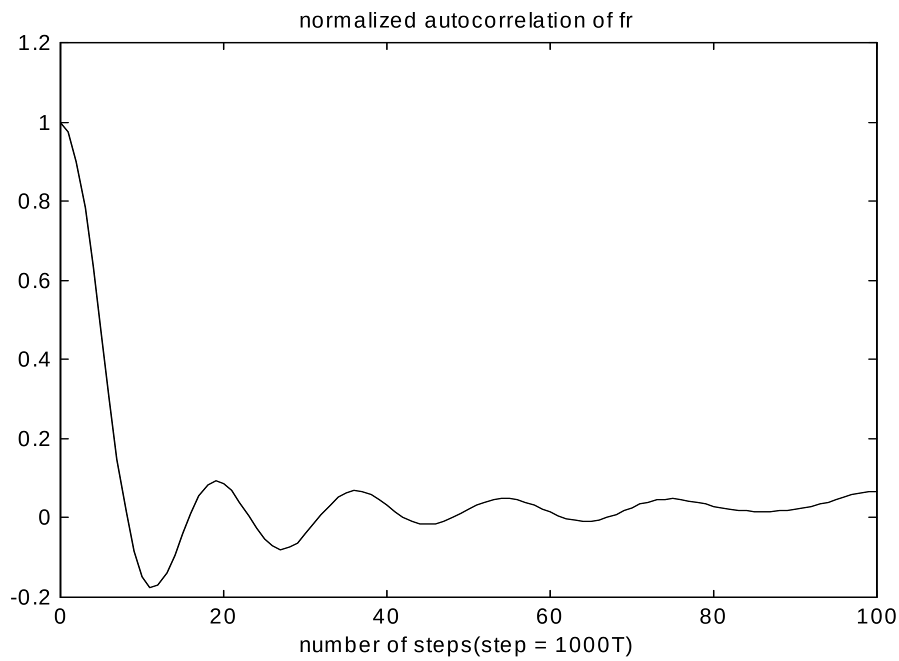
Εικόνα 2.5 : Συνάρτηση αυτοσυσχέτισης για την $f_r(t)$

Παρατηρούμε ότι η τιμή της αυτοσυσχέτισης στο 0 είναι η επιθυμητή, δηλαδή περίπου ½.

Στη συνέχεια ακολουθούν κανονικοποιημένα διαγράμματα της θεωρητικής συνάρτησης αυτοσυσχέτισης και της συνάρτησης αυτοσυσχέτισης του συστήματος προσομοίωσης, για βήμα προσομοίωσης $1000T_s$ και για μικρό αριθμό δειγμάτων (για μια πιο ευκρινή εικόνα της συμπεριφοράς των συναρτήσεων στην αρχή των αξόνων):

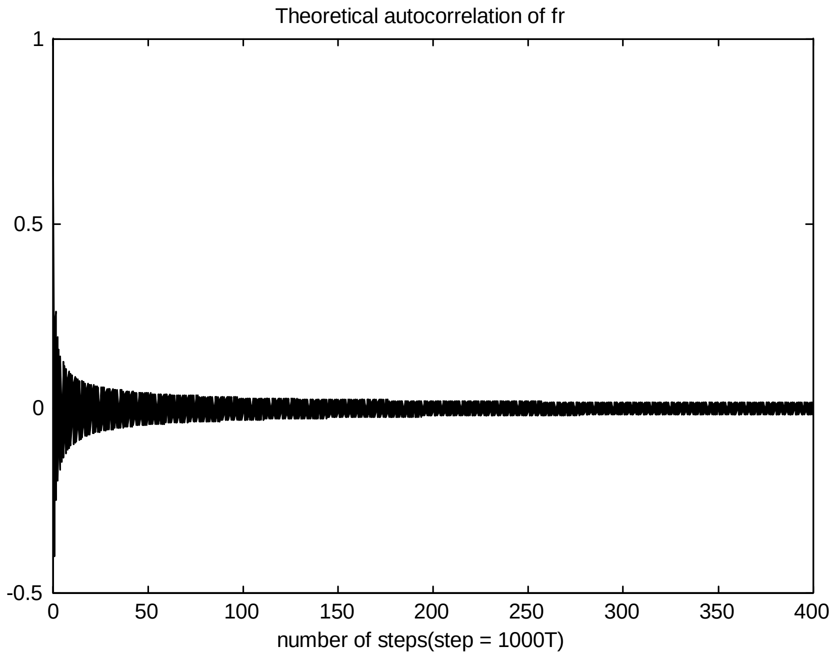


Εικόνα 2.6: Θεωρητική κανονικοποιημένη αυτοσυσχέτιση για την $f_r(t)$

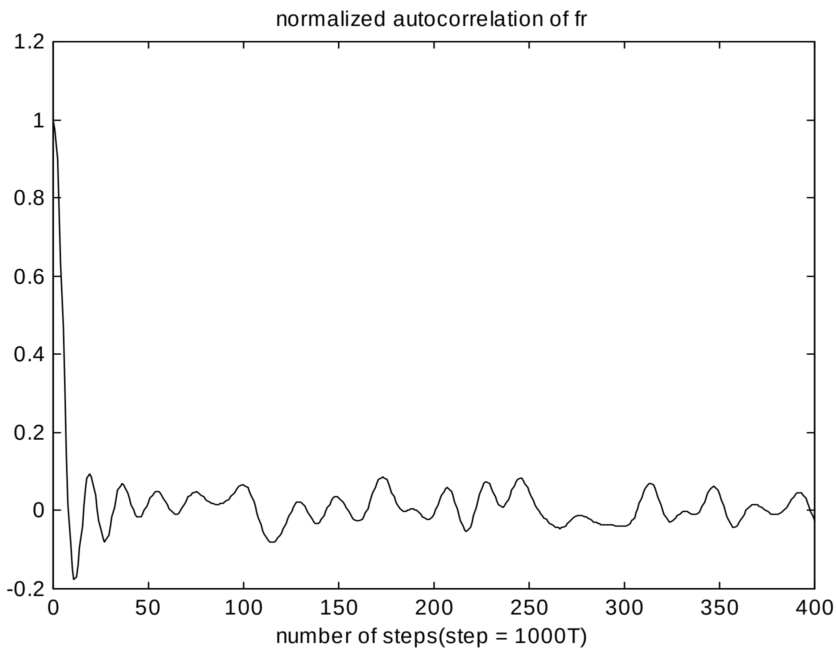


Εικόνα 2.7: Προσεγγιστική κανονικοποιημένη αυτοσυσχέτιση για την $f_r(t)$

Ενώ για μεγάλο αριθμό δειγμάτων :



Εικόνα 2.8: Θεωρητική κανονικοποιημένη αυτοσυσχέτιση για την $f_r(t)$ για μεγάλο αριθμό δειγμάτων

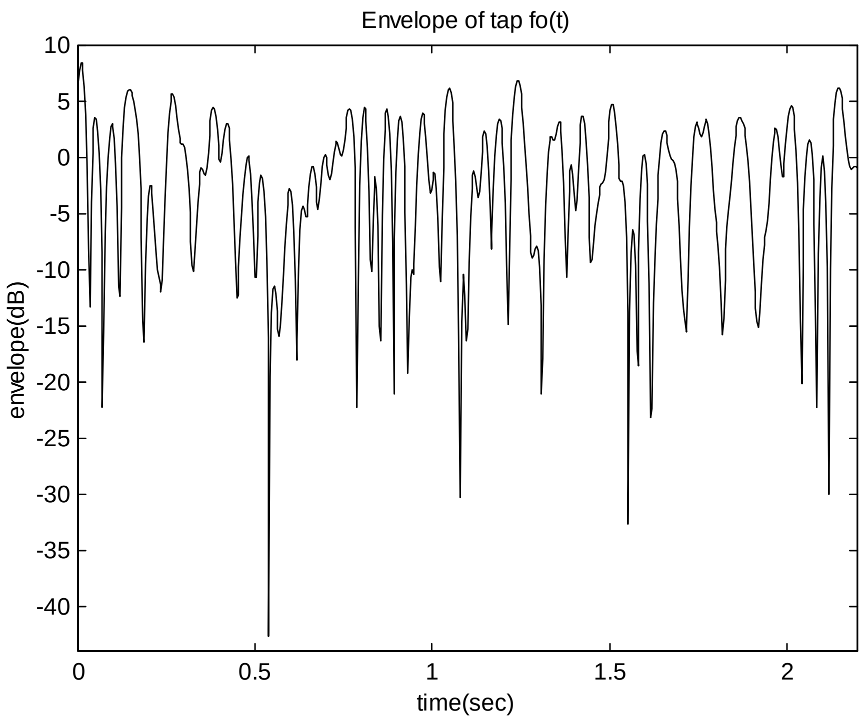


Εικόνα 2.9: Προσεγγιστική κανονικοποιημένη αυτοσυσχέτιση για την $f_r(t)$ για μεγάλο αριθμό δειγμάτων

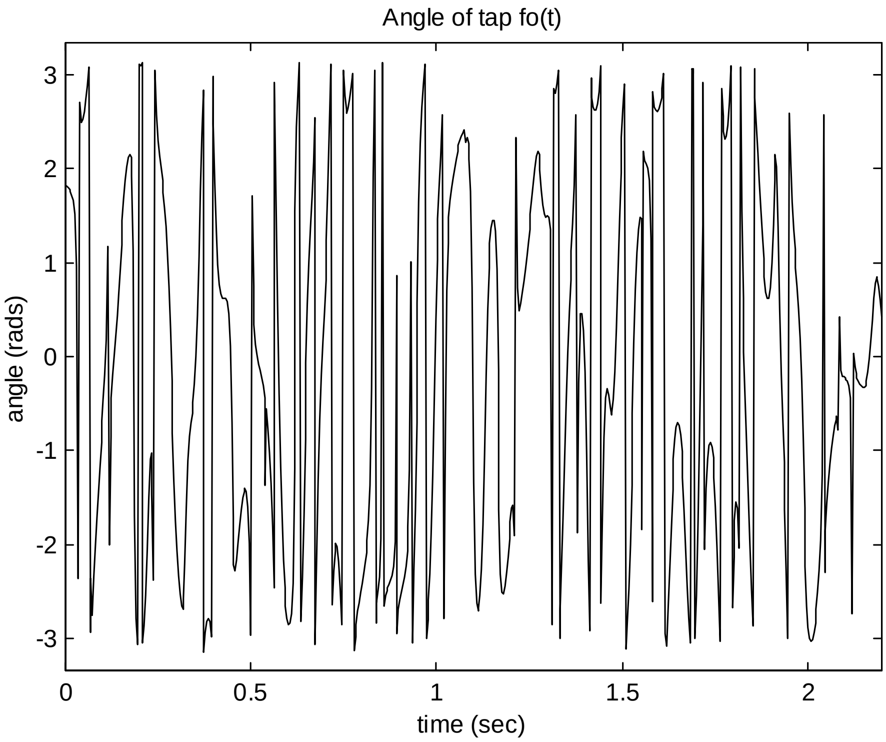
Παρατηρούμε από τα διαγράμματα ότι το σύστημα που χρησιμοποιήθηκε προσεγγίζει τη ζητούμενη συνάρτηση αυτοσυσχέτισης χωρίς όμως να μπορεί να την επιτύχει ακριβώς, γεγονός που οφείλεται όπως προαναφέρθηκε στην δυνατότητα του συστήματος να παράγει μόνο ρητές μορφές πυκνότητας φάσματος ισχύος, ενώ η θεωρητική είναι εν γένη μη ρητής μορφής.

Τα παραπάνω ισχύουν και για την ακολουθία τιμών της $f_i(t)$.

Δείγματα της περιβάλλουσας (σε dB) και της φάσης (σε rad) για το πρώτο tap του καναλιού δίνονται στη συνέχεια :



Εικόνα 2.10: Περιβάλλουσα του πρώτου tap



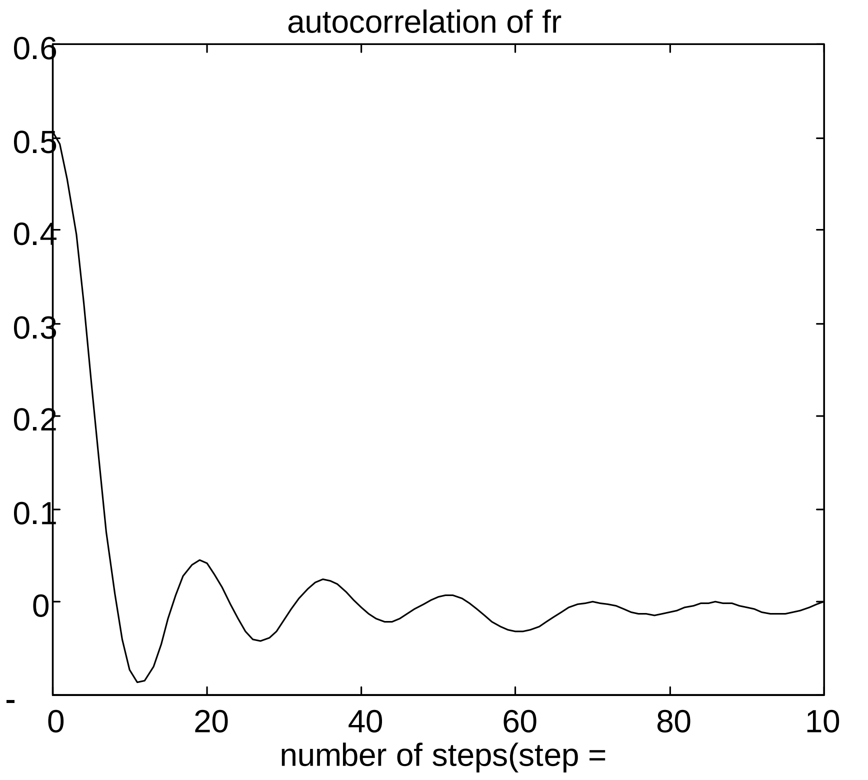
Εικόνα 2.11: Φάση του πρώτου tap

Πρέπει να σημειωθεί ότι η φάση έχει ιδιαίτερη σημασία για τη δημιουργική ή καταστροφική άθροιση των σημάτων που φθάνουν στο δέκτη μετά από σκέδαση επάνω στα αντικείμενα του περιβάλλοντος χώρου. Επιπλέον μικρές αλλαγές στο μέσο μετάδοσης μπορούν να προκαλέσουν μεταβολές της φάσης κατά 2π ή και περισσότερο όπως άλλωστε φαίνεται και από το παραπάνω διάγραμμα.

Κανάλι ακραίας δυναμικής

Για τον τύπο αυτό του καναλιού το γινόμενο $f_d T_s$ παίρνει την τιμή 0.005 που δηλώνει ότι στο κανάλι μας γίνονται 5 αλλαγές ανα 1000 σύμβολα ή 1 αλλαγή ανα 200. Οι τιμές των παραμέτρων του συστήματος που μας δίνουν μια ικανοποιητική προσέγγιση του θεωρητικού μοντέλου είναι 2 για τη διασπορά του λευκού θορύβου, 10 για το βαθμό του φίλτρου και 1/8 για τη συχνότητα αποκοπής του.

Όπως και στη προηγούμενη περίπτωση ακολουθεί η συνάρτηση αυτοσυσχέτισης για την $f_r(t)$.

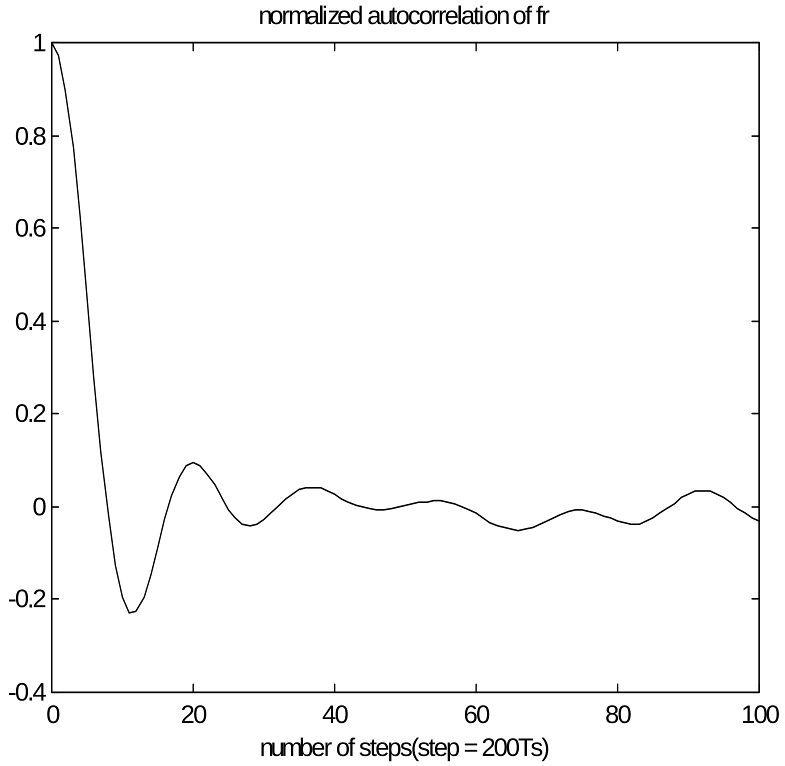


Εικόνα 2.12 : Συνάρτηση αυτοσυσχέτισης για την $f_r(t)$

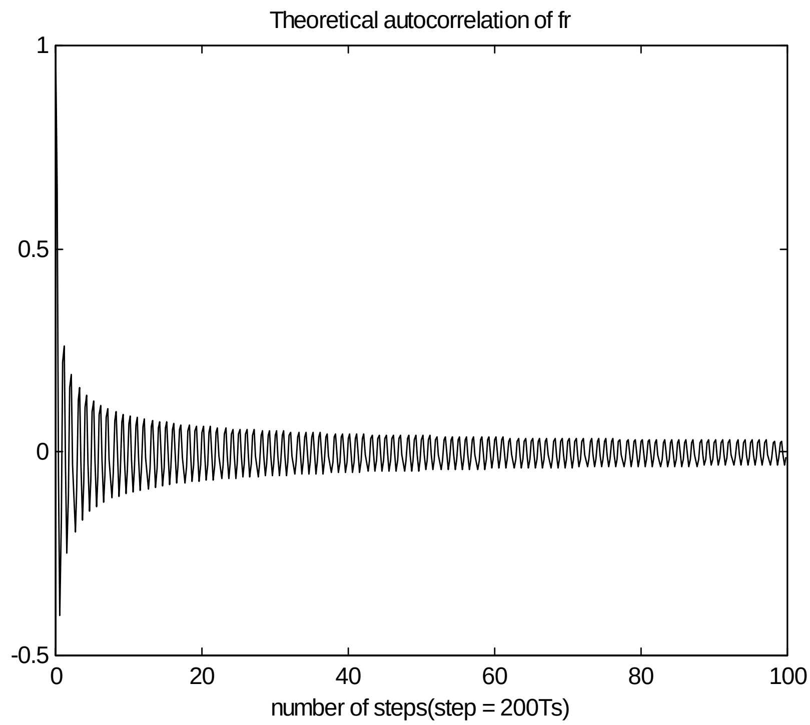
Διαπιστώνουμε ότι και στην περίπτωση αυτή η διασπορά της $f_r(t)$ είναι ½ όπως επιθυμούσαμε.

Παρακάτω παρουσιάζονται τα διαγράμματα για τη σύγκριση θεωρητικής και προσεγγιστικής συνάρτησης αυτοσυσχέτισης.

Για μικρό αριθμό δειγμάτων :

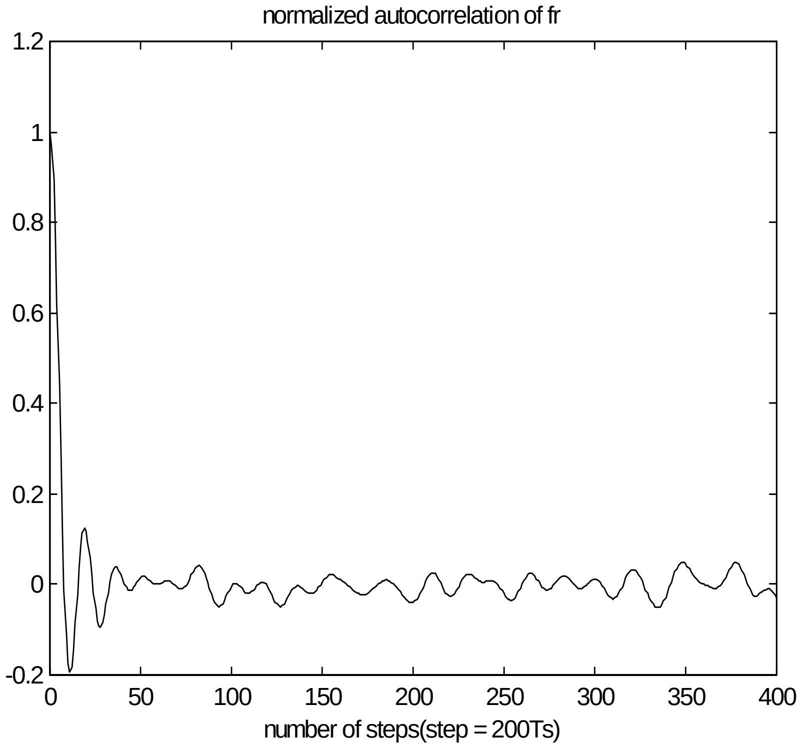


Εικόνα 2.13 : Προσεγγιστική κανονικοποιημένη συνάρτηση αυτοσυσχέτισης για την $f_r(t)$

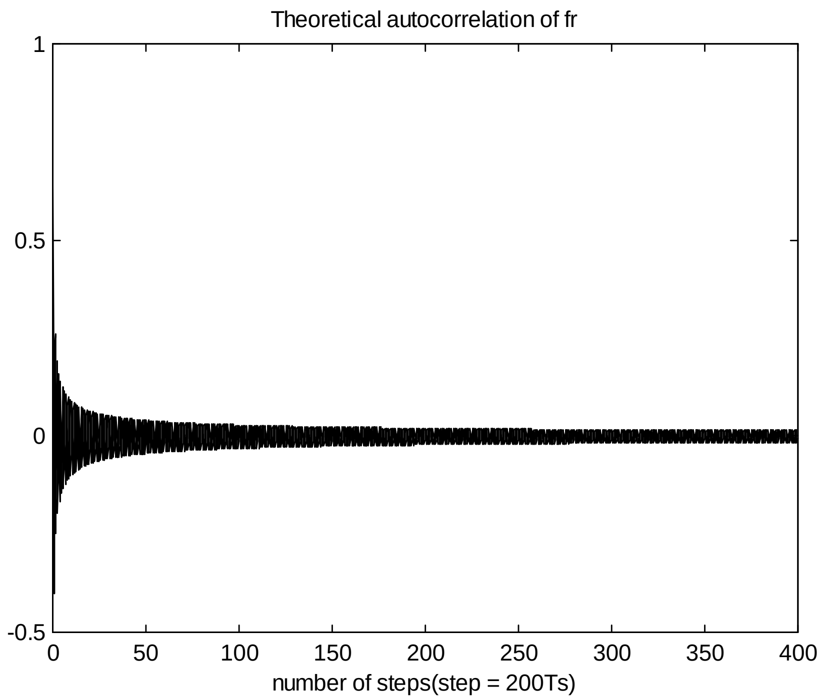


Εικόνα 2.14 : Θεωρητική κανονικοποιημένη συνάρτηση αυτοσυσχέτισης για την $f_r(t)$

Για μεγάλο αριθμό δειγμάτων :



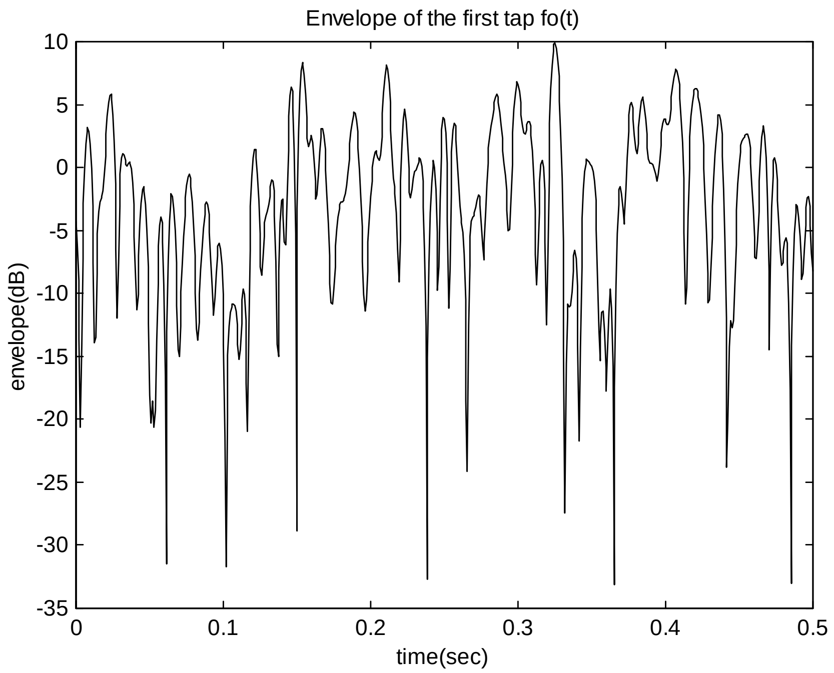
Εικόνα 2.15 : Προσεγγιστική κανονικοποιημένη συνάρτηση αυτοσυσχέτισης για την $f_r(t)$



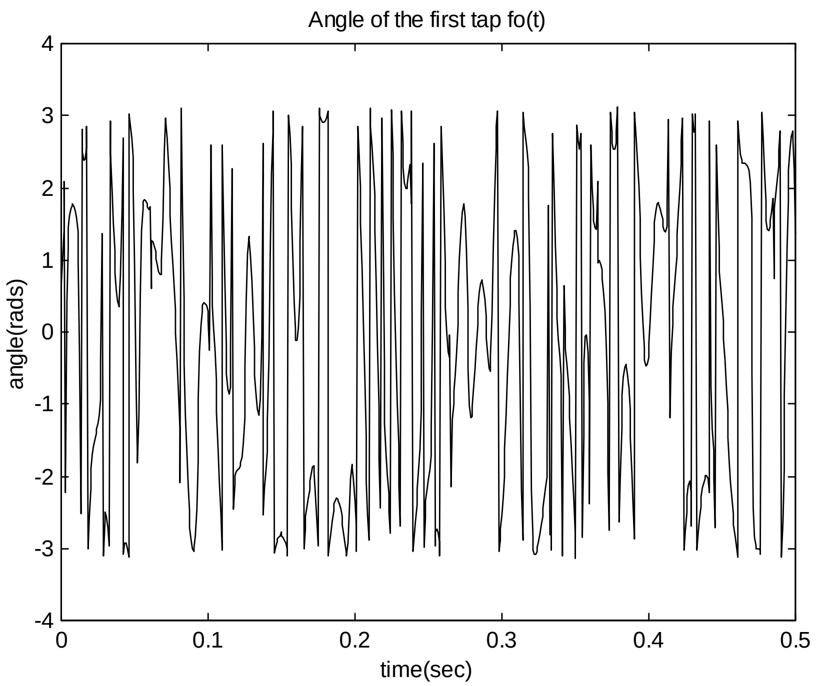
Εικόνα 2.16 : Θεωρητική κανονικοποιημένη συνάρτηση αυτοσυσχέτισης για την $f_r(t)$

Και στη περίπτωση αυτή ισχύουν οι παρατηρήσεις που έγιναν σχετικά με την δυνατότητα του συστήματος να προσεγγίσει τη ζητούμενη κυματομορφή της συνάρτησης Bessel.

Στη συνέχεια δίνεται ένα δείγμα της περιβάλλουσας και της φάσης του πρώτου tap.



Εικόνα 2.17 : Δείγμα της περιβάλλουσας της $f_r(t)$



Εικόνα 2.18 : Δείγμα της φάσης της $f_r(t)$

2.3 Ο ΑΝΑΔΡΟΜΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ ΕΛΑΧΙΣΤΟΥ ΤΕΤΡΑΓΩΝΟΥ

Όπως είδαμε και σε προηγούμενες παραγράφους οι αλγόριθμοι PSP-MLSE και PSP-SSA προκειμένου να εξάγουν σωστές αποφάσεις για τη μεταδιδόμενη ακολουθία συμβόλων είναι απαραίτητο να έχουν μια εκτίμηση του διανύσματος παραμέτρων του καναλιού $\theta(\mu_\kappa)$ ανα επιβιώνουσα μετάβαση. Η εκτίμηση αυτή χρησιμοποιείται για τον υπολογισμό των μετρικών των κλάδων :

$$\lambda(\mu_\kappa \to \mu_{\kappa+1}) = F[\mu_\kappa \to \mu_{\kappa+1}, r(t), \hat{\theta}(\mu_\kappa)] \qquad (2\text{-}3\text{-}1)$$

Στη περίπτωση μας η άγνωστη παράμετρος για το δέκτη είναι η ίδια η διακριτού χρόνου κρουστική απόκριση :

$$\mathbf{f} = (f_0, f_1, \ldots, f_L)^T \qquad (2\text{-}3\text{-}2)$$

Παράλληλα είδαμε στη προηγούμενη παράγραφο ότι η κρουστική απόκριση του υπό εξέταση διαύλου μεταβάλεται και μάλιστα με εξαιρετικά γρήγορο ρυθμό (και στις δύο περιπτώσεις που περιγράφηκαν). Έτσι είναι απαραίτητη η χρήση ενός αλγόριθμου που θα μας δίνει μια όσο το δυνατό πιο ακριβή εκτίμηση του διανύσματος **f** ανά επιβιώνουσα μετάβαση και που θα είναι ικανός να προσαρμόζεται στις αλλαγές του διαύλου.

Ο αναδρομικός αλγόριθμος ελαχίστου μέσου τετραγώνου (RLS) μπορεί να χρησιμοποιηθεί για τον παραπάνω σκοπό. Η λειτουργία του αλγόριθμου RLS στηρίζεται στην ανα επιβιώνουσα μετάβαση εκτίμηση του διανύσματος **f** ώστε να ελαχιστοποιείται η τιμή του χρονικού μέσου του τετραγώνου του σφάλματος. Θα συμβολίσουμε την εκτίμηση του διανύσματος **f** που σχετίζεται με μια κατάσταση μ τη χρονική στιγμή k με :

$$\hat{\mathbf{f}}(\mu_\kappa) = [\hat{f}_0(\mu_\kappa), \hat{f}_1(\mu_\kappa), \ldots, \hat{f}_L(\mu_\kappa)]^T \qquad (2\text{-}3\text{-}3)$$

Αν συμβολίσουμε με t τη παράμετρο που εκφράζει τη διέλευση του διακριτού χρόνου από 0 ως τη στιγμή κ, δηλαδή t = 0, 1, 2,..., κ , τότε y_t θα είναι η είσοδος στον αποκωδικοποιητή τη χρονική στιγμή t, ενώ η ακολουθία $\{a_i(\mu_t \rightarrow \mu_{t+1})\}_{i=t-L}^{t}$ είναι η ακολουθία των συμβόλων που σχετίζεται με τη συγκεκριμένη μετάβαση και μπορεί να γραφεί με τη μορφή διανύσματος ως $\mathbf{a}(\mu_t \rightarrow \mu_{t+1})$. Τέλος τα μη θορυβικά δείγματα στην έξοδο του διαύλου δίνονται από τη σχέση :

$$x(\mu_t \rightarrow \mu_{t+1}) = \sum_{i}^{L} f_i a_{k-i}(\mu_t \rightarrow \mu_{t+1}) \qquad (2\text{-}3\text{-}4)$$

Οι εκτιμήσεις των παραπάνω δειγμάτων δίνονται από το βαθμωτό γινόμενο :

$$\widetilde{x}(\mu_t \rightarrow \mu_{t+1}) = \hat{\mathbf{f}}^T(\mu_t) \cdot \mathbf{a}(\mu_t \rightarrow \mu_{t+1}) \qquad (2\text{-}3\text{-}5)$$

Έχοντας προσδιορίσει τα παραπάνω μεγέθη μπορούμε να γράψουμε την μετρική κόστους που επιχειρεί να ελαχιστοποιήσει ο RLS ως :

$$E^{LS} = \sum_{t=0}^{k} w^{k-t} \left| e(\mu_t \rightarrow \mu_{t+1}) \right|^2 \qquad (2\text{-}3\text{-}6)$$

όπου το σφάλμα e ορίζεται από τη σχέση :

$$e(\mu_t \rightarrow \mu_{t+1}) = y_t - \widetilde{x}(\mu_t \rightarrow \mu_{t+1}) = y_t - \hat{\mathbf{f}}^T(\mu_t) \cdot \mathbf{a}(\mu_t \rightarrow \mu_{t+1}) \qquad (2\text{-}3\text{-}7)$$

και w είναι ένα παράγοντας βάρους που παίρνει τιμές στο διάστημα (0,1]. Ο παράγοντας αυτός εισάγει ένα εκθετικό βάρος στα παρελθώντα σφάλματα που έχει βαρύνουσα σημασία ιδίως όταν το κανάλι μεταβάλεται με το χρόνο. Το μέγεθος $(1\text{-}w)^{-1}$ προσδιορίζει τη μνήμη του αλγόριθμου, και για w=1 θα έχουμε άπειρη μνήμη, ενώ όσο πλησιάζουμε το μηδέν τόσο πιο γρήγορα «ξεχνά» ο αλγόριθμος τα παλαιότερα δεδομένα.

Παρακάτω ακολουθεί μια περιγραφή του αναδρομικού αλγόριθμου σε πέντε βήματα:

Α)

Ο RLS ξεκινά τη χρονική στιγμή 0 με τις ακόλουθες αρχικοποιήσεις :

$$\hat{\mathbf{f}}(\mu_0) = [0,0,...,0]^T \tag{2-3-8}$$

$$\mathbf{P}(\mu_0) = \delta\mathbf{I} \tag{2-3-9}$$

όπου **P** είναι ένας NxN (N ίσο με τον αριθμό των taps)πίνακας που χρησιμοποιεί ο αλγόριθμος (ο αντίστροφος του πίνακα συσχετίσεων του εισερχόμενου σήματος από την έξοδο του διαύλου) ,δ είναι ένας μεγάλος ακέραιος και **I** είναι ο μοναδιαίος πινακας .

Β)

Σε κάθε χρονική στιγμή κ+1 γίνεται υπολογισμός του σφάλματος $e(\mu_t \to \mu_{t+1})$ για κάθε επιβιώνουσα μετάβαση $\mu_\kappa \to \mu_{\kappa+1}$ χρησιμοποιώντας το διάνυσμα $\hat{\mathbf{f}}(\mu_\kappa)$ με βάση τη σχέση :

$$e(\mu_t \to \mu_{t+1}) = y_t - \tilde{x}(\mu_t \to \mu_{t+1}) = y_t - \hat{\mathbf{f}}^T(\mu_t)\cdot\mathbf{a}(\mu_t \to \mu_{t+1}) \tag{2-3-10}$$

Γ)

Υπολογίζουμε το διάνυσμα κέρδους του φίλτρου Kalman (gain vector) από τη σχέση:

$$\mathbf{k}(\mu_{\kappa+1}) = \frac{\mathbf{P}(\mu_\kappa)\mathbf{y}^*(k+1)}{w + \mathbf{y}^T(k+1)\mathbf{P}(\mu_\kappa)\mathbf{y}^*(k+1)} \tag{2-3-11}$$

όπου $\mathbf{y}^T(k+1)$ είναι το διάνυσμα των εισόδων του αποκωδικοποιητή από τον διακριτού χρόνου δίαυλο :

$$\mathbf{y}^T(k+1) = [y(k+1), y(k), y(k-1)]^T \tag{2-3-12}$$

Δ)

Ενημέρωση του αντίστροφου του πίνακα συσχετίσεων P :

$$\mathbf{P}(\mu_{\kappa+1}) = \frac{1}{w}[\mathbf{P}(\mu_{\kappa}) - \mathbf{k}(\mu_{\kappa+1})\mathbf{y}^{T}(k+1)\mathbf{P}(\mu_{\kappa})] \qquad (2\text{-}3\text{-}13)$$

E)

Τελικά γίνεται ενημέρωση του διανύσματος των taps :

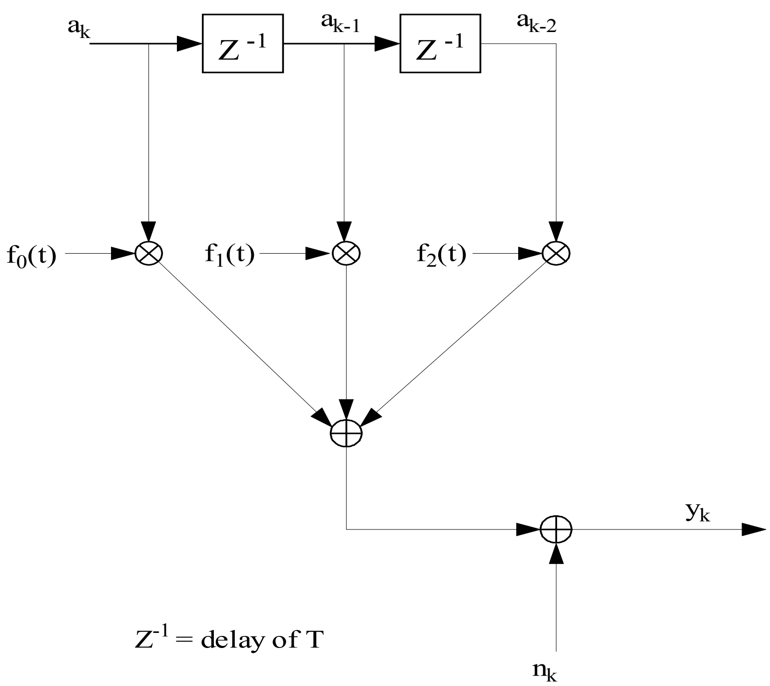
$$\hat{\mathbf{f}}(\mu_{\kappa+1}) = \hat{\mathbf{f}}(\mu_{\kappa}) + \mathbf{k}(\mu_{\kappa+1})e(\mu_{t} \rightarrow \mu_{t+1}) \qquad (2\text{-}3\text{-}14)$$

Η επιλογή του αλγόριθμου RLS για την εκτίμηση των taps του καναλιού έγινε επειδή ο συγκεκριμένος αλγόριθμος έχει τη δυνατότητα να συγκλίνει στις επιθυμητές τιμές πολύ γρήγορα. Η ταχύτητα δε αυτή αυξάνεται ακόμη περισσότερο με τη χρήση σύντομης ακολουθίας εκπαίδευσης. Το χαρακτηριστικό αυτό είναι σπουδαίας σημασίας για τη διαδικασία ανίχνευσης ενός γρήγορα μεταβαλομενου καναλιού, καθώς εξαιτίας του μικρού μεγέθους πακέτου που χρησιμοποιείται δεν επιτρέπεται η χρήση μεγάλης ακολουθίας εκπαίδευσης διότι τότε θα οδηγούμασταν σε μείωση του ποσοστού μεταδιδόμενης πληροφορίας σε σχέση με το πλεόνασμα πληροφορίας (overhead).

Το κόστος βέβαια που καλούμαστε να πληρώσουμε για τη μεγάλη ταχύτητα σύγκλισης είναι η μεγάλη πολυπλοκότητα του αλγόριθμου, καθώς αυτή είναι της τάξης του N^2 ($2.5N^2+4.5N$). Επιπλέον, όταν η παράμετρος w είναι μικρότερη της μονάδας η σύγκλιση του αλγόριθμου στις βέλτιστες τιμές χαρακτηρίζεται από αλγοριθμικό «θόρυβο» ιδιαίτερα στη στάσιμη κατάσταση, γεγονός που μειώνει την απόδοση του αλγόριθμου.

2.4 ΠΑΡΑΔΕΙΓΜΑ ΕΦΑΡΜΟΓΗΣ ΤΩΝ ΑΛΓΟΡΙΘΜΩΝ PSP-MLSE ΚΑΙ PSP-SSA ΣΕ ΔΙΑΥΛΟ ΜΕΤΑΒΛΗΤΩΝ ΠΑΡΑΜΕΤΡΩΝ

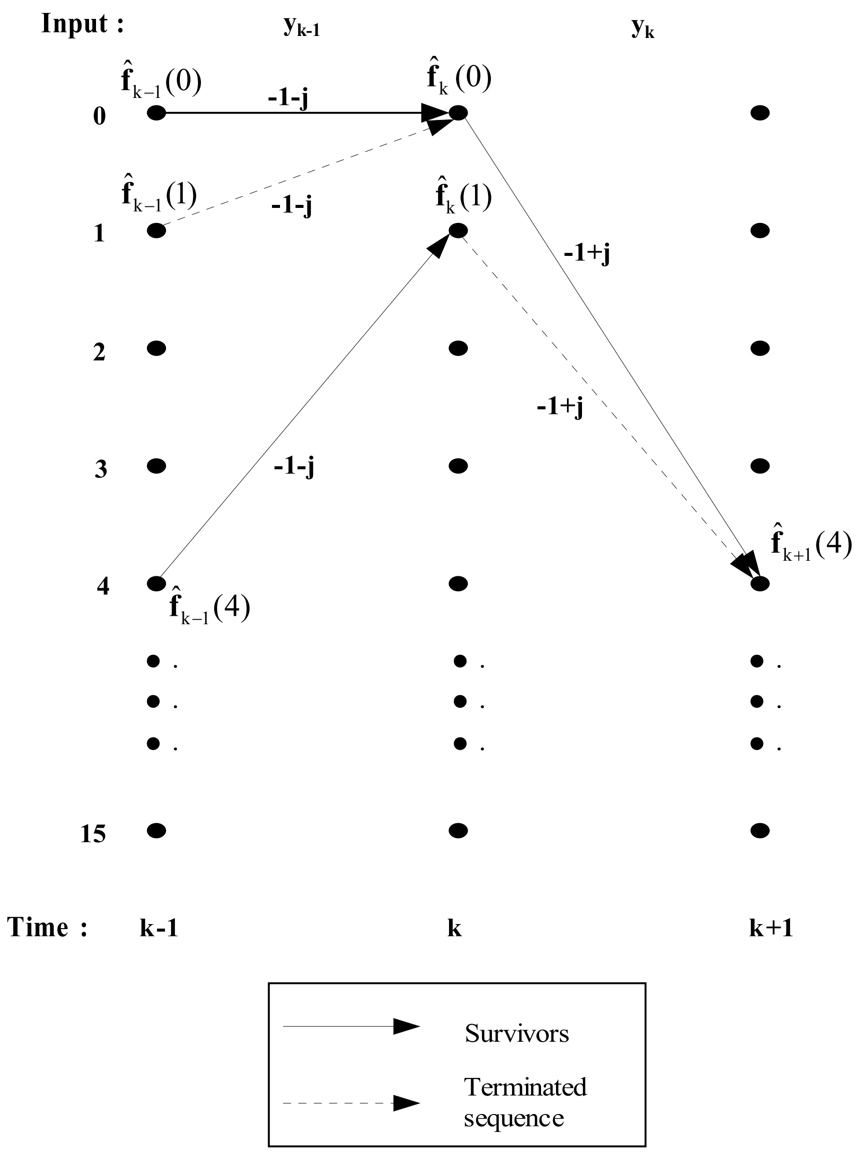
Στο παράδειγμα θα θεωρήσουμε QPSK διαμόρφωση για τη μεταδιδόμενη πληροφορία μέσα από δίαυλο μεταβλητών παραμέτρων με μνήμη L=2. Τα σύμβολα της πληροφορίας a_k ανήκουν στο αλφάβητο { (-1-j), (-1+j), (1-j), (1+j) } και συνεπώς έχουμε Μ = 4. Το διακριτού χρόνου ισοδύναμο μοντέλο του διαύλου επαναλαμβάνεται στο παρακάτω διάγραμμα :



Εικόνα 2.19 : Μοντέλο ισοδύναμου διακριτού χρόνου διαύλου με AWGN

Οι συντελεστές $\left\{ f_i(t) \right\}_{i=0}^{2}$ είναι άγνωστοι στο δέκτη και μεταβάλλονται με το χρόνο,και συνεπώς πρέπει να εκτιμηθούν. Το διάγραμμα trellis που περιγράφει την παραπάνω διασυμβολική παρεμβολή, θεωρώντας πλήρη σε καταστάσεις περιγραφή, θα έχει $M^{L} = 4^2 = 16$ καταστάσεις σε κάθε στάδιο και $M^{L+1} = 4^3 = 64$ δυνατές μεταβάσεις.

Το διάγραμμα trellis κατά τη χρονική στιγμή k θα έχει την παρακάτω μορφή :



Εικόνα 2.20 : Διάγραμμα trellis με ενδεικτικές μεταβάσεις

Στο παραπάνω διάγραμμα trellis εμφανίζονται μερικές ενδεικτικές μεταβάσεις που απεικονίζονται με βέλη. Για κάθε μετάβαση αναγράφεται το σύμβολο a_k που προκαλεί τη συγκεκριμένη μετάβαση και δίπλα σε κάθε κατάσταση εμφανίζεται το διάνυσμα των παραμέτρων της. Το μητρώο συμβόλων $\left(a_{k-1}, a_{k-2}\right)$ που σχετίζεται με την κατάσταση αυτή περιγράφεται από τον παρακάτω πίνακα :

Αριθμός Κατάστασης	$\left(a_{k-1}, a_{k-2}\right)$
0	[(-1-j) , (-1-j)]
1	[(-1-j) , (-1+j)]
2	[(-1-j) , (1-j)]
3	[(-1-j) , (1+j)]
4	[(-1+j) , (-1-j)]
5	[(-1+j) ,(-1+j)]
6	[(-1+j) , (1-j)]
7	[(-1+j) , (1+j)]
8	[(1-j) , (-1-j)]
9	[(1-j) , (-1+j)]
10	[(1-j) , (1-j)]
11	[(1-j) , (1+j)]
12	[(1+j) , (-1-j)]
13	[(1+j) ,(-1+j)]
14	[(1+j) , (1-j)]
15	[(1+j) , (1+j)]

Πινακας 2.2 : Μητρώο συμβόλων που σχετίζεται με κάθε κατάσταση

Έτσι τη χρονική στιγμή k για κάθε κατάσταση μ_k με μ=0, 1, 2,... ,15 υπάρχει ένα διάνυσμα εκτίμησης παραμέτρων που αντιστοιχεί στην επιβιώνουσα μετάβαση η οποία τερματίζει στην κατάσταση αυτή (π.χ., για την κατάσταση 0 υπάρχει το $\hat{\mathbf{f}}_{k-1}(0)$) και οι συσσωρευμένες μετρικές $\Gamma(\mu_\kappa)$).Στη περίπτωση του PSP-MLSE για κάθε πιθανή μετάβαση $\mu_k \rightarrow \mu_{k+1}$ υπολογίζονται τα σφάλματα σύμφωνα με την εξίσωση :

$$e(\mu_t \rightarrow \mu_{t+1}) = y_t - \widetilde{x}(\mu_t \rightarrow \mu_{t+1}) = y_t - \hat{\mathbf{f}}^T(\mu_t)\cdot \mathbf{a}(\mu_t \rightarrow \mu_{t+1})$$

όπου y_k είναι η είσοδος στον αποκωδικοποιητή κατά τη χρονική στιγμή k. Παραδείγματος χάριν για την κατάσταση 4 υπολογίζονται τα ακόλουθα σφάλματα :

$$e(0_k \rightarrow 4_{k+1}) = y_k - \hat{\mathbf{f}}_k^T(0)\cdot \mathbf{a}(0_k \rightarrow 4_{k+1})$$
$$e(1_k \rightarrow 4_{k+1}) = y_k - \hat{\mathbf{f}}_k^T(1)\cdot \mathbf{a}(1_k \rightarrow 4_{k+1})$$

όπου

$$\mathbf{a}(0_k \rightarrow 4_{k+1}) = \begin{bmatrix} -1+j \\ -1-j \\ -1-j \end{bmatrix} \quad \text{και} \quad \mathbf{a}(1_k \rightarrow 4_{k+1}) = \begin{bmatrix} -1+j \\ -1-j \\ -1+j \end{bmatrix}$$

και

$$\hat{\mathbf{f}}_k^T(0) = [\hat{f}_0(0_k), \hat{f}_1(0_k), \hat{f}_2(0_k)]$$
$$\hat{\mathbf{f}}_k^T(1) = [\hat{f}_0(1_k), \hat{f}_1(1_k), \hat{f}_2(1_k)]$$

Οι μετρικές των μεταβάσεων στην κατάσταση 4 είναι :

$$\lambda(0_k \rightarrow 4_{k+1}) = \left|e(0_k \rightarrow 4_{k+1})\right|^2$$
$$\lambda(1_k \rightarrow 4_{k+1}) = \left|e(1_k \rightarrow 4_{k+1})\right|^2$$

και η συνολική μετρική της κατάστασης 4 είναι :

$$\Gamma(4_{k+1}) = \min_{0_k, 1_k}[\Gamma(0_k) + \lambda(0_k \rightarrow 4_{k+1}), \Gamma(1_k) + \lambda(1_k \rightarrow 4_{k+1})]$$

Θεωρώντας (όπως φαίνεται και στο σχήμα) ότι :

$$\Gamma(0_k) + \lambda(0_k \to 4_{k+1}) < \Gamma(1_k) + \lambda(1_k \to 4_{k+1})$$

έχουμε :

$$\Gamma(4_{k+1}) = \Gamma(0_k) + \lambda(0_k \to 4_{k+1})\,.$$

Η παραπάνω διαδικασία επαναλαμβάνεται για όλες τις κατάστασεις μ_{k+1} προκειμένου να βρεθούν οι επιβιώνουσες μεταβάσεις. Στη συνέχεια για κάθε επιβιώνουσα μετάβαση χρησιμοποιείται ο αλγόριθμος RLS προκειμένου να ανανεώσει την εκτίμηση των taps υπολογίζοντας το διάνυσμα $\hat{\mathbf{f}}_{k+1}(\mu)$, και στην περίπτωση του παραδείγματος το $\hat{\mathbf{f}}_{k+1}(4)$.

Η αποκωδικοποίηση γίνεται όπως στον κλασικό αλγόριθμο Viterbi επιλέγοντας μια καθυστέρηση D μεγαλύτερη από 5L για την αποκωδικοποίηση ενός συμβόλου ή στη περίπτωση που η ακολουθία των συμβόλων είναι μικρού μήκους (π.χ., μικρού μεγέθους πακέτο) είναι προτιμότερη η επεξεργασία και αποκωδικοποίηση ολόκληρου του πακέτου. Στην περίπτωση αυτή βέβαια θα έχουμε μεταβλητή καθυστέρηση κατά την αποκωδικοποίηση του κάθε συμβόλου. Από τα παραπάνω είναι φανερό ότι κάθε διατηρούμενο μονοπάτι κατά την αναζήτηση κρατά και ενημερώνει το δικό του διάνυσμα παραμέτρων, βασισμένο στη δική του σχετιζόμενη ακολουθία δεδομένων. Πρόκειται δηλαδή για εκτίμηση παραμέτρων με αποκεντρικοποιημένο τρόπο και η οποία μπορεί να ερμηνευτεί ως μηδενικής καθυστέρησης τοπική ανάδραση απόφασης.

Ο αλγόριθμος PSP-SSA, στην ειδική περίπτωση που η σταθερά καθυστέρησης για την αποκωδικοποίηση ενός συμβόλου D επιλεγεί ίση με L, δηλαδή ίση με την μνήμη του διαύλου, αποκτά μια απλοποιημένη μορφή. Αν συγκρίνουμε τους PSP-MLSE και PSP-SSA όταν ο τελευταίος χρησιμοποιείται στην πρακτική του μορφή μπορούμε να κάνουμε τις εξής παρατηρήσεις :

(α) Οι δύο αλγόριθμοι είναι ταυτόσημοι ώς προς το ότι οι συσσωρευμένες μετρικές και οι εκτιμήσεις των παραμέτρων του διαύλου ενημερώνονται με τον ίδιο τρόπο. Σε κάθε βήμα δηλαδή οι συσσωρευμένες μετρικές και οι εκτιμήσεις είναι *πανομοιότυπες* για τους δύο αλγόριθμους.

(β) Η μόνη διαφορά έγκειται στον τρόπο με τον οποίο εκτιμούν τα δεδομένα : ο PSP-MLSE όπως προαναφέρθηκε κρατά τις επιβιώνουσες και η έξοδος του είναι η ακολουθία συμβόλων που αντιστοιχεί στην επιβιώνουσα με μια καθυστέρηση που κυμαίνεται πρακτικά μεταξύ 5L και 7L. Από την άλλη πλευρά ο PSP-SSA παίρνει αποφάσεις για τα σύμβολα με καθυστέρηση L. Ένα προφανές πλεονέκτημα της τελευταίας ιδιότητας είναι ότι δεν χρειάζεται επεξεργασία για την ανίχνευση κάθε επιβιώνουσας σε κάθε βήμα k, διαδικασία που αποτελεί σημαντικό μέρος της όλης πολυπλοκότητας που ενέχεται στην υλοποίηση του PSP-MLSE.

Το παραπάνω παράδειγμα είναι λοιπόν ενδεικτικό και για τον τρόπο λειτουργίας του PSP-SSA.

ΚΕΦΑΛΑΙΟ 3^{ο}

ΠΡΟΣΟΜΟΙΩΣΕΙΣ

Στο κεφάλαιο αυτό παρουσιάζουμε τα αποτελέσματα των προσομοιώσεων των αλγορίθμων που περιγράφηκαν στο πρώτο κεφάλαιο. Ο κώδικας των προσομοιώσεων έχει γραφεί σε C++ και MATLAB. Η C++ προσφέρει την αντιστοίχιση του φυσικού μοντέλου με στοιχεία της γλώσσας (τα blocks αντιστοιχίζονται σε objects), ένα πολύ γρήγορο εκτελέσιμο κώδικα και τη δυνατότητα επαναχρησιμοποίσης των διαφόρων τμημάτων του κώδικα. Το MATLAB επιλέχθηκε κυρίως για την προσομοίωση του καναλιού καθώς προσφέρει ένα ολοκληρωμένο πακέτο εφαρμογών για στατιστική επεξεργασία τυχαίων διακριτών σημάτων όπως για παράδειγμα φίλτρα και γεννητριες τυχαίων ακολουθιών. Επιπλέον χρησιμοποιήθηκε για τη σχεδίαση των γραφικών παραστάσεων.

Η κύρια μετρική της επίδοσης των αλγορίθμων που μας απασχόλησε είναι ο μέσος ρυθμός σφαλμάτων στα μεταδιδόμενα ψηφία, δηλαδή το bit error rate (BER). Ο θόρυβος που προστίθεται στη έξοδο του διακριτού χρόνου διαύλου είναι λευκός γκαουσιανός μηδενικής μέσης τιμής και διασποράς $\sigma^2 = N_0$. Οι γραφικές παραστάσεις έγιναν ως προς τη σχέση :

$$10\log(\frac{E_b}{N_0})$$

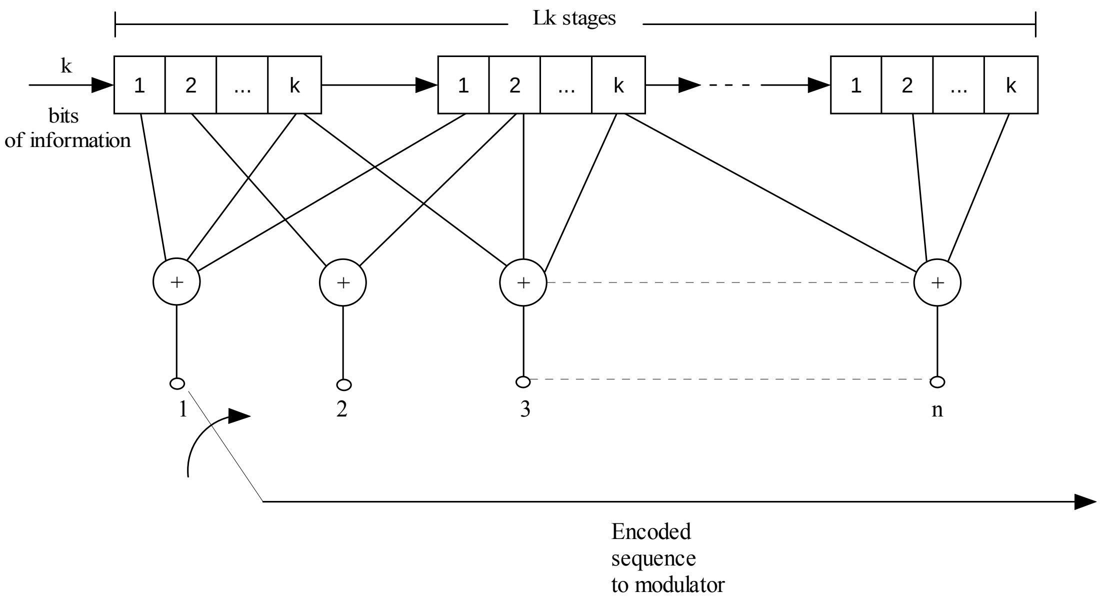
όπου E_b είναι η ενέργεια του μη διαμορφωμένου bit (raw bit) και έχει επιλεγεί ίση με τη μονάδα.

Στις παραγράφους 3.1 και 3.2 περιγράφονται το σύστημα συνελικτικής κωδικοποίησης και το σύστημα του Block Interleaver αντίστοιχα, που χρησιμοποιούνται στην προσομοίωση προκειμένου να έχουμε ένα ολοκληρωμένο σύστημα. Η παράγραφος 3.3 παρουσιάζει τη δομή του συνολικού τηλεπικοινωνιακού συστήματος σε μορφή block διαγράμματος και αναλύονται οι τιμές των διαφόρων παραμέτρων που χρησιμοποιήθηκαν. Τα αποτελέσματα των προσομοιώσεων για κάθε αλγόριθμο εμφανίζονται στην παράγραφο 3.4, ενώ στην 3.5 γίνεται σύγκριση της επίδοσης των δύο αλγορίθμων.

3.1 ΣΥΣΤΗΜΑ ΣΥΝΕΛΙΚΤΙΚΗΣ ΚΩΔΙΚΟΠΟΙΗΣΗΣ ΤΗΣ ΜΕΤΑΔΙΔΟΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΑΣ

Οι συνελικτικοί κώδικες χαρακτηρίζονται από την εισαγωγή μνήμης στη διαδικασία κωδικοποίησης. Έτσι κάθε ακολουθία από k bits αντιστοιχίζεται σε μια νέα ακολουθία των n bits για να μεταδοθεί μέσω του καναλιού, αυτά τα n bits όμως δεν προσδιορίζονται μόνο από τα πρόσφατα k bits, αλλά και από προηγούμενα bits πληροφορίας. Αυτή η εξάρτηση από προηγούμενα bit πληροφορίας κάνει τον κωδικοποιητή να συμπεριφέρεται ώς μηχανή πεπερασμένων καταστάσεων.

Για να γίνουν πιο σαφή τα παραπάνω δίνεται το διάγραμμα του συνελικτικού κωδικοποιητή :

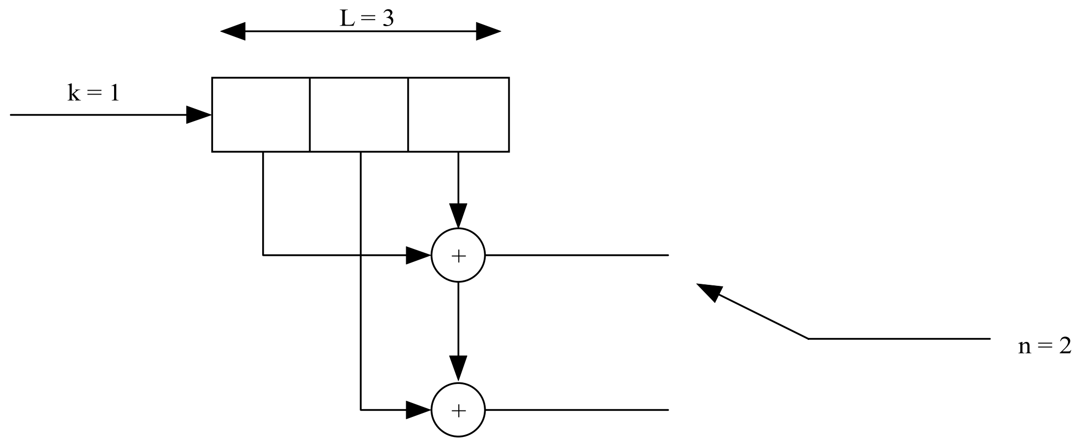


Εικόνα 3.1 : Διάγραμμα γενικού μοντέλου συνελικτικτικού κωδικοποιητή

Ο συνελικτικός κωδικοποιητής αποτελείται από ένα καταχωρητή ολίσθησης (shift register) με kL βαθμίδες όπου το L ονομάζεται μήκος περιορισμού του κώδικα. Σε κάθε χρονική στιγμή, k bits πληροφορίας εισέρχονται στον καταχωρητή ολίσθησης και τα περιεχόμενα των τελευταίων k stages του καταχωρητή αποβάλλονται. Μετά την εισαγωγή των k bits στον καταχωρητή υπολογίζονται n γραμμικοί συνδυασμοί (exclusive-OR) των περιεχομένων του καταχωρητή όπως φαίνεται και στο σχήμα, οι οποίοι χρησιμοποιούνται για τη δημιουργία της κωδικοποιημένης κυματομορφής. Από την παραπάνω διαδικασία κωδικοποιήσης είναι προφανές ότι τα n bits εξόδου του κωδικοποιητή δεν εξαρτώνται μόνο από τα πρόσφατα k bits που εισήχθησαν σ'αυτόν, αλλά και από τα (L-1)k περιεχόμενα των πρώτων (L-1)k βαθμίδων του καταχωρητή πριν φθάσουν τα καινούργια k bits. Για το λόγο αυτό ο καταχωρητής ολίσθησης είναι μια μηχανή πεπερασμένων καταστάσεων με $2^{(L-1)k}$ καταστάσεις. Επιπρόσθετα, επειδή για κάθε k bits εισόδου έχουμε n bits εξόδου ο ρυθμός του κώδικα θα είναι :

$$R_{conv} = \frac{k}{n} \qquad (3\text{-}1\text{-}1)$$

Ο συνελικτικός κωδικοποιητής που χρησιμοποιήθηκε στη προσομοίωση περιγράφεται διαγραμματικά στο παρακάτω σχήμα :



Εικόνα 3.2 : Διάγραμμα συνελικτικτικού κωδικοποιητή (k=1,L=3,n=2)

Για αυτόν τον κωδικοποιητή έχουμε k = 1, n = 2, και L = 3. Έτσι ο ρυθμός του είναι ½ και ο αριθμός των καταστάσεων είναι $2^{(L-1)k} = 4$. Ένας τρόπος για την περιγραφή ενός τέτοιου κώδικα είναι να προσδιορίσουμε πώς τα δύο bits εξόδου του κωδικοποιητή εξαρτώνται από τα περιεχόμενα του shift register. Αυτό γίνεται συνήθως προσδιορίζοντας n διανύσματα g_1, g_2,…, g_n, που ονομάζονται γεννήτριες ακολουθίες του συνελικτικού κώδικα. Το στοιχείο i , $1\leq i \leq kL$, του g_j , $1\leq j \leq n$, είναι

1 αν η i-οστή stage του καταχωρητή ολίσθησης συνδέεται στο συνδυαστή που αντιστοιχεί το j bit της εξόδου, αλλοιώς είναι μηδέν. Στη περίπτωση που υλοποιήθηκε, οι γεννήτριες ακολουθίες έχουν ως εξής :

$$g_1 = [\ 1\ 0\ 1\ ] \tag{3-1-2}$$

$$g_2 = [\ 1\ 1\ 1\ ] \tag{3-1-3}$$

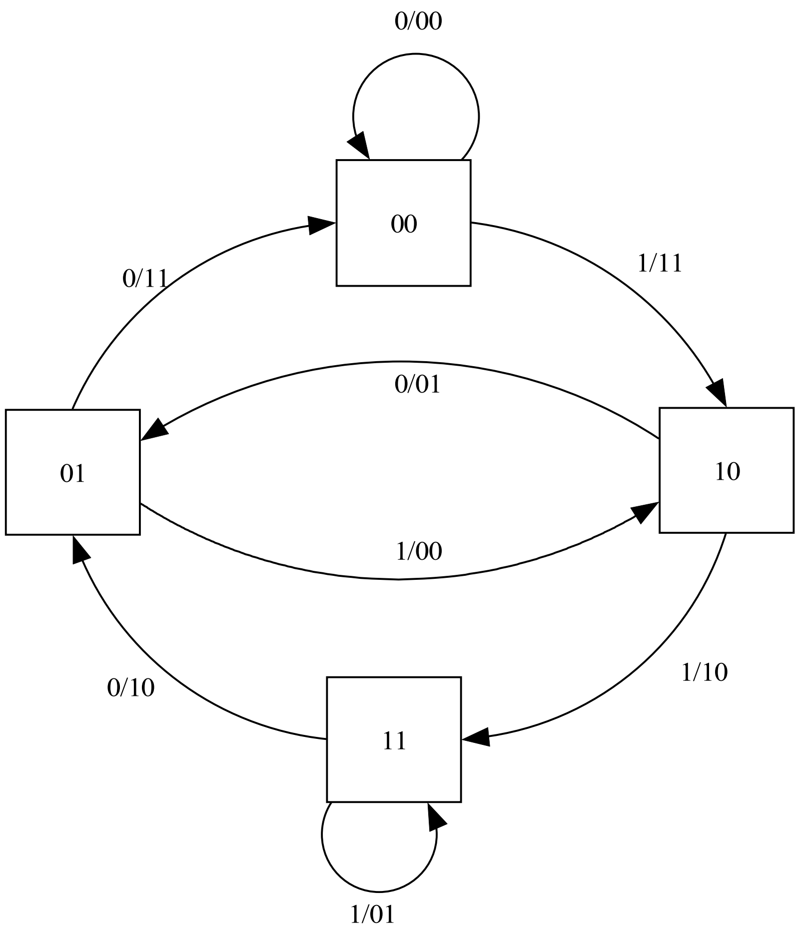
Η επιλογή των παραπάνω γεννητριών συναρτήσεων έγινε με βάση τη βιβλιογραφία ώστε να αποφευχθεί το ενδεχόμενο χρήσης καταστροφικού κώδικα. Ένας συνελικτικός κώδικας αντιστοιχίζει μια ακολουθία από bits πληροφορίας σε μια κωδική λέξη η οποία στη συνέχεια μεταδίδεται μέσα από το κανάλι. Ο σκοπός της κωδικοποίησης είναι να παρέχει μεγαλύτερο επίπεδο προστασίας ενάντια στο θόρυβο του καναλιού. Προφανώς ένας κώδικας που αντιστοιχίζει ακολουθίες πληροφορίας που διαφέρουν σε μεγάλο αριθμό bits σε κωδικές λέξεις που διαφέρουν μικρό αριθμό από bits δεν είναι ένας καλός κώδικας αφού οι δύο κωδικές λέξεις μπορούν να μπερδευτούν εύκολα γεγονός που θα οδηγήσει σε μεγάλο αριθμό σφαλμάτων στη ακολουθία πληροφορίας. Μια οριακή περίπτωση αυτής της ανεπιθύμητης ιδιότητας συμβαίνει όταν δύο ακολουθίες οι οποίες είναι διαφορετικές σε άπειρο πλήθος θέσεων αντιστοιχίζονται σε κωδικές λέξεις που διαφέρουν σε ένα πεπερασμένο αριθμό θέσεων. Σε μια τέτοια περίπτωση πάντα υπάρχει πιθανότητα οι κωδικές λέξεις να αποκωδικοποιηθούν λάθος και αυτό με τη σειρά του αντιστοιχεί σε άπειρο αριθμό σφαλμάτων κατά την εύρεση της ακλουθίας πληροφορίας. Κώδικες που εμφανίζουν αυτή την ιδιότητα ονομάζονται καταστροφικοί και πρέπει να αποφεύγονται στη πράξη.

State Transition diagram

Επειδή ο συνελικτικός κωδικοποιητής έχει πεπερασμένη μνήμη, μπορεί εύκολα να παρασταθεί από ένα διάγραμμα καταστάσεων – μεταβάσεων. Σε ένα τέτοιο διάγραμμα κάθε κατάσταση του συνελικτικού κωδικοποιητή παριστάνεται με ένα τετράγωνο και οι μεταβάσεις μεταξύ των καταστάσεων παριστάνονται με γραμμές που ενώνουν τα τετράγωνα. Σε κάθε γραμμή προσδιορίζονται τόσο η είσοδος που προκαλεί τη μετάβαση όσο και η αντίστοιχη έξοδος που παράγεται από μια τέτοια μετάβαση.

Ο αριθμός των γραμμών που ξεκινούν από κάθε κατάσταση είναι ίσος με τον αριθμό των πιθανών εισόδων στον κωδικοποιητή, όταν βρίσκεται στη συγκεκριμένη κατάσταση, που είναι ίσος με 2^{κ}.Ο αριθμός των γραμμών που καταλήγουν σε κάθε κατάσταση ισούται με τον αριθμό των καταστάσεων από τις οποίες είναι δυνατή η μετάβαση στη κατάσταση αυτή. Ο αριθμός αυτός ισούται με το πλήθος των καταστάσεων από τις οποίες είναι δυνατή η μετάβαση προς αυτή την κατάσταση.Το πλήθος δε των καταστάσεων ισούται με τον αριθμό των δυνατών συνδυασμών των bits που αφήνουν τον κωδικοποιητή καθώς εισέρχονται τα k νέα bits στον κωδικοποιητή, που είναι ίσος με 2^{κ}.

Το διάγραμμα καταστάσεων-μεταβάσεων για τον συνελικτικό κωδικοποιητή που χρησιμοποιήθηκε φαίνεται παρακάτω :

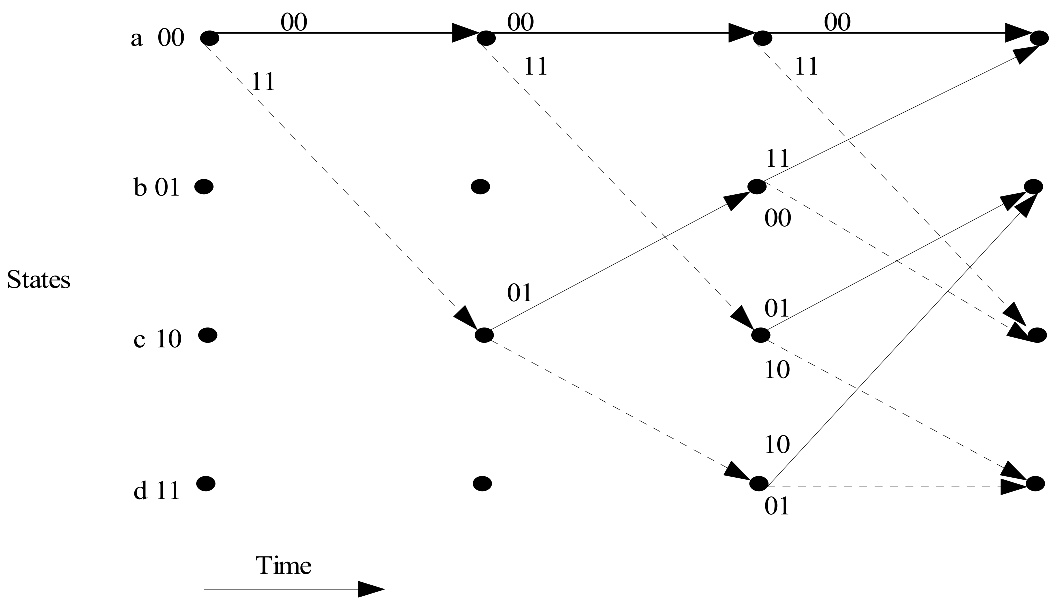


Εικόνα 3.3 : Διάγραμμα καταστάσεων-μεταβάσεων συνελικτικού κωδικοποιητή (k=1,L=3,n=2)

Διάγραμμα trellis

Μια δεύτερη πιο συνηθισμένη μέθοδος για την περιγραφή συνελικτικών κωδίκων είναι να προσδιοριστεί το διάγραμμα trellis τους. Το διάγραμμα trellis είναι ένας τρόπος για να καταγραφούν οι μεταβάσεις ανάμεσα στις διάφορες καταστάσεις κατά τη πάροδο του χρόνου. Η σχεδίαση του διαγράμματος trellis γίνεται με την τοποθέτηση όλων των δυνατών καταστάσεων σε ένα κάθετο άξονα και επαναλαμβάνοντας τον άξονα αυτόν κατα μήκος του άξονα του χρόνου. Κάθε μετάβαση μεταξύ δύο καταστάσεων δηλώνεται με μία γραμμή που συνδέει δύο καταστάσεις δύο γειτονικών κάθετων αξόνων που αντιστοιχούν σε δύο συνεχόμενες χρονικές στιγμές. Όπως στη περίπτωση με το διάγραμμα καταστάσεων-μεταβάσεων έχουμε 2^{κ} κλάδους να εκκινούν από κάθε κατάσταση και 2^{κ} κλάδους να απολήγουν σε κάθε κατάσταση. Για την περίπτωση όπου k = 1, συνηθίζεται να συμβολίζεται ο κλάδος που αντιστοιχεί σε είσοδο 0 με μια γραμμή και ο κλάδος που αντιστοιχεί σε είσοδο 1 με μια διακεκομένη γραμμή.

Παρακάτω ακολουθεί το διάγραμμα trellis του συνελικτικού κώδικα που χρησιμοποιήθηκε.



Εικόνα 3.3 : Διάγραμμα trellis συνελικτικού κωδικοποιητή (k=1,L=3,n=2)

Διαδικασία Κωδικοποίησης

Η διαδικασία κωδικοποίσης σε ένα συνελικτικό κώδικα είναι σχετικά απλή. Θεωρούμε ότι ο κωδικοποιητής πριν την εισαγωγή του πρώτου bit πληροφορίας έχει στα στοιχεία μνήμης του την τιμή 0. Τα bits πληροφορίας εισέρχονται στον κωδικοποιητή, k bits ανα χρονική στιγμή (k = 1 για την περίπτωση που χρησιμοποιήθηκε) και τα n bits εξόδου που παράγονται μεταδίδονται μέσω του καναλίου. Η διαδικασία αυτή συνεχίζεται μέχρι την εισαγωγή της τελευταίας ακολουθίας k bits στον συνελικτικό κωδικοποιητή και τη μετάδοση των αντίστοιχων n bits εξόδου. Μετά την μετάδοση και της της τελευταίας ακολουθίας των n bits θεωρούμε την εισαγωγή μιας ακολουθίας k(L-1) bits τιμής 0 στον κωδικοποιητή και την μετάδοση των n αντίστοιχων εξόδων. Αυτό κάνει τον κωδικοποιητή έτοιμο να δεχθεί την επόμενη ακολουθία πληροφορίας.

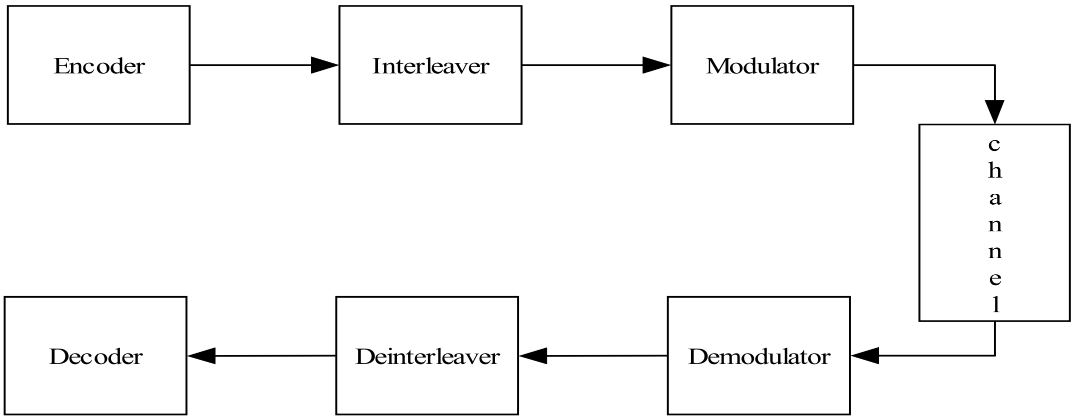
Αποκωδικοποίηση Συνελικτικού Κώδικα

Για την αποκωδικοποίση του συνελικτικού κώδικα χρησιμοποιήθηκε ο κλασικός αλγόριθμος Viterbi με χαλαρές αποφάσεις. Ο συνελικτικός αποκωδικοποιητής προσδιορίζει το μονοπάτι του διαγράμματος trellis που εμφανίζει τη μικρότερη ευκλείδια απόσταση από την ακολουθία των συμβόλων που εξάγει ο δέκτης (PSP-MLSE ή PSP-SSA). Η χρήση της ευκλείδειας απόστασης ως μετρικής αιτιολογείται από το ότι έχουμε δίαυλο με προσθετικό λευκό γκαουσιανό θόρυβο. Γνωρίζουμε ότι κάθε σύμβολο αντιστοιχεί σε ένα συγκεκριμένο dibit που αποτελεί την έξοδο του συνελικτικού κωδικοποιητή κατά τη μετάβαση μεταξύ δύο καταστάσεων. Είναι δυνατό αφού προδιοριστούν η αρχική και τελική κατάσταση της μετάβασης καθώς και το dibit να επισημάνουμε το bit εισόδου στον κωδικοποιητή που προκάλεσε τη συγκεκριμένη μετάβαση από ένα πίνακα μεταβάσεων. Έτσι είναι δυνατή η αναπαραγωγή στο δέκτη του πακέτου πληροφορίας που μεταδώθηκε από τον πομπό.

3.2 ΣΥΣΤΗΜΑ ΔΙΑΣΤΡΩΜΑΤΩΣΗΣ ΤΗΣ ΠΛΗΡΟΦΟΡΙΑΣ (BLOCK INTERLEAVER)

Συγκεκριμένα μοντέλα καναλιών (σ'αυτά περιλαμβάνεται και το κανάλι λευκού προσθετικού γκαουσιανού θορύβου (AWGN)) μπορούν να μοντελοποιηθούν ικανοποιητικά ως κανάλια τυχαίων ανεξάρτητων σφαλμάτων. Σε κάποια άλλα φυσικά κανάλια, όμως, η υπόθεση των ανεξάρτητων σφαλμάτων δεν είναι έγκυρη. Ένα τέτοιο παράδειγμα είναι το Rayleigh fading channel. Σε ένα τέτοιο μοντέλο αν το κανάλι βρίσκεται σε κατάσταση βαθείας εξασθένησης (deep fade) ένας μεγάλος αριθμός σφαλμάτων συμβαίνουν στην ακολουθία μετάδοσης γεγονός που υποδηλώνει την εκρηκτική φύση των σφαλμάτων. Προφανώς σε ένα τέτοιο κανάλι η πιθανότητα σφάλματος σε μια συγκεκριμένη θέση ή χρονική στιγμή εξαρτάται από το αν τα γειτονικά bit έχουν ληφθεί σωστά. Φυσικά κάθε κώδικας διόρθωσης τυχαίων σφαλμάτων μπορεί να χρησιμοποιηθεί και για τη διόρθωση εκρήξεων σφαλμάτων, όσο βέβαια ο αριθμός των σφαλμάτων είναι μικρότερος από το μισό της ελάχιστης απόστασης του κώδικα. Η γνώση όμως της εκρηκτικής φύσης των σφαλμάτων καθιστά δυνατή τη σχεδίαση πιο αποτελεσματικών κωδικοποιητών.

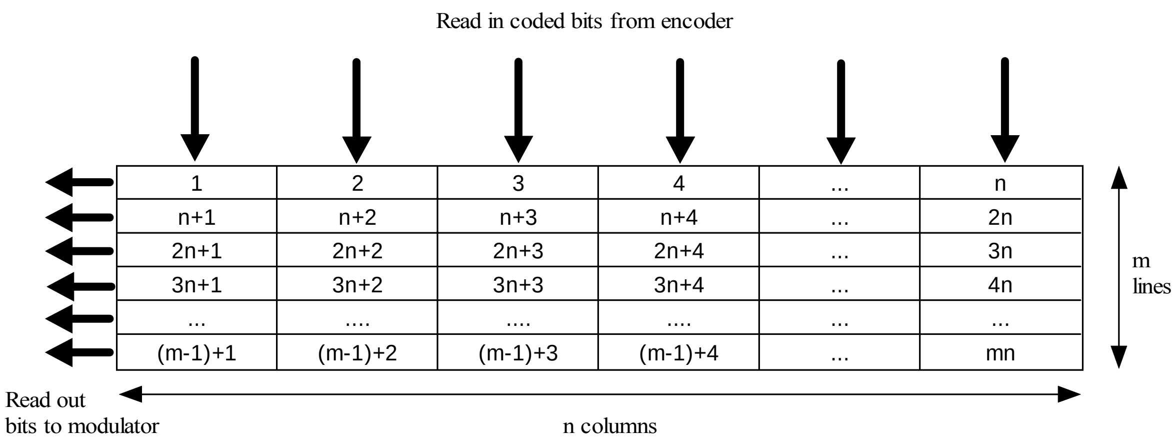
Μια αποτελεσματική μέθοδος για τη διόρθωση εκρήξεων σφαλμάτων είναι η διαστρωμάτωση (interleaving) των κωδικοποιημένων δεδομένων έτσι ώστε οι θέσεις των σφαλμάτων να μοιάζουν τυχαίες και να είναι κατανεμημένες σε πολλές κωδικές λέξεις. Μ'αυτό τον τρόπο ο αριθμός των σφαλμάτων που συμβαίνουν σε κάθε block είναι μικρός και μπορεί να διορθωθεί χρησιμοποιώντας ένα κώδικα διόρθωσης τυχαίων σφαλμάτων. Στο δέκτη γίνεται χρήση ενός deinterleaver για την αναίρεση της λειτουργίας του interleaver. Ένα διάγραμμα συστήματος κωδικοποίησης που χρησιμοποιεί interleaving/deinterleaving φαίνεται παρακάτω :



Εικόνα 3.4 : Σύστημα κωδικοποίησης με interleaver/deinterleaver

Ένας interleaver βάθους m διαβάζει m κωδικές λεξεις μήκους n η κάθε μια και τις τοποθετεί σε ένα πίνακα m γραμμών και n στηλών. Στη συνέχεια αυτός ο πίνακας διαβάζεται ανά στήλη και στέλνεται στον ψηφιακό διαμορφωτή. Στον δέκτη η έξοδος του αποδιαμορφωτή τροφοδοτείται στον deinterleaver ο οποίος δημιουργεί την ίδια mxn δομή, την διαβάζει ανά γραμμή και στέλνει την έξοδο στον αποκωδικοποιητή του καναλιού.

Παρακάτω ακολουθεί το διάγραμμα του interleaver :



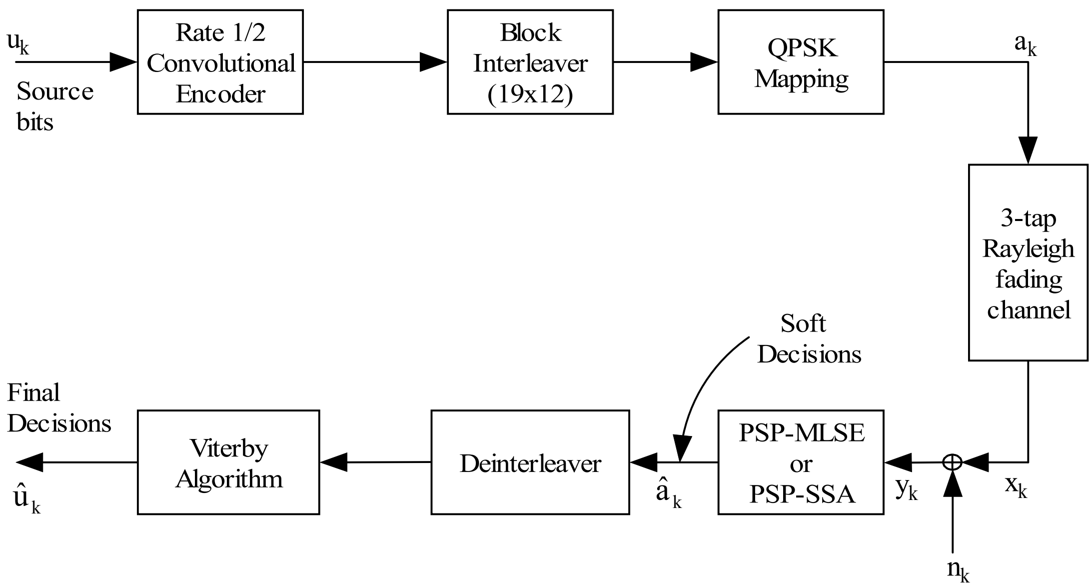
1	2	3	4	...	n
n+1	n+2	n+3	n+4	...	2n
2n+1	2n+2	2n+3	2n+4	...	3n
3n+1	3n+2	3n+3	3n+4	...	4n
...	....	....	....	...	...
(m-1)+1	(m-1)+2	(m-1)+3	(m-1)+4	...	mn

Εικόνα 3.5 : Δομικό διάγραμμα του Block Interleaver

Στον interleaver γίνεται εισαγωγή της ακολουθίας bit με αύξοντες αριθμούς 1 2 3... n … 2n … mn όπως φαίνεται και στο παραπάνω σχήμα. Στη συνέχεια στην έξοδο του interleaver εμφανίζονται τα στοιχεία της πρώτης στήλης δηλαδή τα bits με αύξοντες αριθμούς 1, n+1, 2n+1, 3n+1, 4n+1,…, (m-1)+1. Στο σύστημα του deinterleaver εφαρμόζεται η αντίστροφη διαδικασία για την παραγωγή της αρχικής ακολουθίας.

3.3 ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΔΟΜΗΣ ΤΟΥ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΚΟΥ ΣΥΣΤΗΜΑΤΟΣ

Προκειμένου να εκτιμήσουμε την απόδοση των αλγόριθμων PSP-MLSE και PSP-SSA έγινε προσομοίωση του παρακάτω τηλεπικοινωνιακού συστήματος :



Εικόνα 3.6: Δομικό διάγραμμα του Τηλεπικοινωνιακού Συστήματος

Στο σύστημα εισάγεται μια ακολουθία των 228 bits το τελευταίο bit της οποίας είναι πάντοτε 0. Η ακολουθία αυτή κωδικοποιείται με συνελικτικό κωδικοποιητή, ρυθμού ½ , παράγοντας μια ακολουθία 228 τετραδικών συμβόλων (quaternary symbols). Στη συνέχεια ακολουθεί διαστρωμάτωση (interleaving) των συμβόλων από έναν 19x12 block interleaver. Το τελευταίο block του πομπού πραγματοποιεί διαμόρφωση κατά QPSK της κωδικοποιημένης ακολουθίας. Το πακέτο των 228 συμβόλων μεταδίδεται μέσω ενός frequency selective Rayleigh fading channel, αφού προηγηθεί μετάδοση 26 συμβόλων ακολουθίας εκπαίδευσης. Το παραπάνω σχήμα μετάδοσης είναι όμοιο με το GSM. Διαφορές παρουσιάζονται ως προς το απλούστερο interleaving που εφαρμόζεται εδώ και τη διαμόρφωση, όπου χρησιμοποιείται η QPSK αντί της GMSK.

Ο δέκτης αποτελείται από το block των PSP-MLSE και PSP-SSA που παίρνουν το πακέτο των παραμορφωμένων συμβόλων από διασυμβολική παρεμβολή και λευκό γκαουσιανό θόρυβο διασποράς N_o και εξάγουν χαλαρές αποφάσεις για κάθε σύμβολο του πακέτου. Ακολουθούν τα blocks του de-interleaver, για την αναίρεση του interleaving, και του αποκωδικοποιητή για τον συνελικτικό κώδικα. Οι προσωμοιώσεις έγιναν για πολύ γρήγορης και ακραίας δυναμικής Rayleigh fading κανάλι με $f_dT_s = 0.001$ και $f_dT_s = 0.005$ αντίστοιχα, όπου T_s είναι η περίοδος του συμβόλου και ισούται σύμφωνα με το πρότυπο GSM με 3.69μsec. Το κανάλι θεωρήθηκε ότι διαθέτει τρία ανεξάρτητα μονοπάτια (κυρίως για τον περιορισμό της πολυπλοκότητας των διαγραμμάτων trellis).

Στο δέκτη η εκτίμηση και η παρακολούθηση των μεταβολλών του καναλιού γίνεται χρησιμοποιώντας τον αλγόριθμο RLS. Για τον παράγοντα αμνησίας του RLS, w, χρησιμοποιήθηκαν τιμές μεταξύ 0.94 και 0.96 για το γρήγορα μεταβαλλόμενο κανάλι και μεταξύ 0.76 και 0.80 για το κανάλι ακραίας δυναμικής αντίστοιχα, ανάλογα με την τιμή του σηματοθορυβικού λόγου. Οι παραπάνω τιμές βελτιστοποιούν την απόδοση του αλγόριθμου RLS όπως προέκυψε από τα αποτελέσματα των προσομοιώσεων.

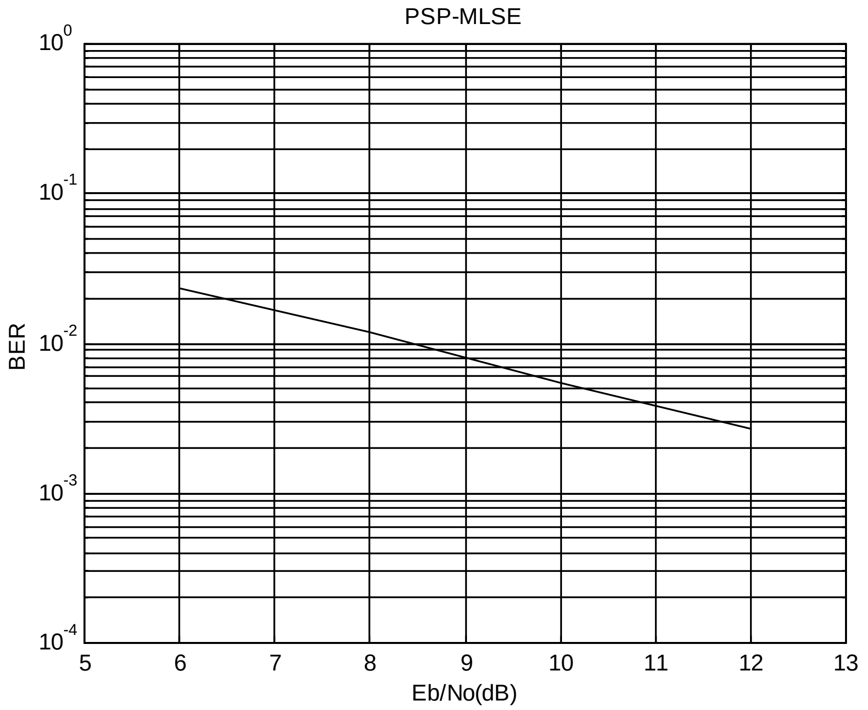
3.4 ΑΠΟΤΕΛΕΣΜΑΤΑ ΠΡΟΣΟΜΟΙΩΣΕΩΝ ΓΙΑ ΤΙΣ ΜΕΘΟΔΟΥΣ PSP-MLSE ΚΑΙ PSP-SSA

PSP-MLSE

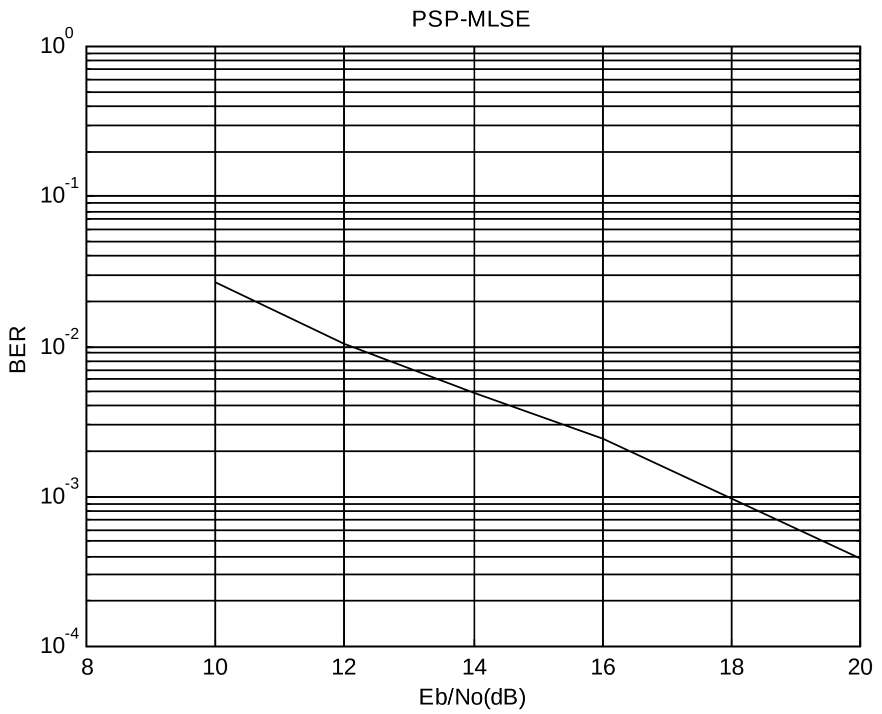
Η διαδικασία επεξεργασίας της λαμβανόμενης ακολουθίας που χρησιμοποιεί ο αλγόριθμος περιγράφεται στο πρώτο κεφάλαιο. Εδώ αξίζει να σημειωθεί ότι από τις δύο δυνατότητες που προσφέρονται για τη σχεδίαση του αλγόριθμου δηλαδή λήψη χαλαρής απόφασης με σταθερή καθυστέρηση $D \geq 5L$ ή επεξεργασία ολόκληρου του πακέτου και εύρεση της συνολικά καλύτερης επιβιώνουσας επιλέχθηκε ο δεύτερος τρόπος. Η επιλογή αυτή βελτιώνει την απόδοση του αλγόριθμου αφού δεν καταφεύγουμε σε πρακτικές μεν, αλλά υποβέλτιστες δε μεθόδους όπως η παραπάνω. Επιπλέον παρέχει τη δυνατότητα υλοποίσης πιο γρήγορου κώδικα (αποφεύγουμε μετακινήσεις block μνήμης που ενέχονται σε υλοποιήσεις του αλγόριθμου με $D \geq 5L$). Το κόστος της επεξεργασίας ολόκληρου του πακέτου, κυρίως σε μνήμη, περιορίζεται αισθητά λόγω του μικρού μεγέθους του πακέτου που χρησιμοποιείται καθιστώντας δυνατή την υλοποίηση της μεθόδου.

Έχοντας περιγράψει τις τιμές που παίρνουν οι παράμετροι του τηλεπικοινωνιακού συστήματος στην παράγραφο 3.2 προχωρούμε στην παρουσίαση των αποτελεσμάτων :

α. Για την περίπτωση του πολύ γρήγορα μεταβαλλόμενου καναλιού με $f_dT_s = 0.001$ και για πλήθος 9000 πακέτων (≡ 2.052.000 σύμβολα) έχουμε :



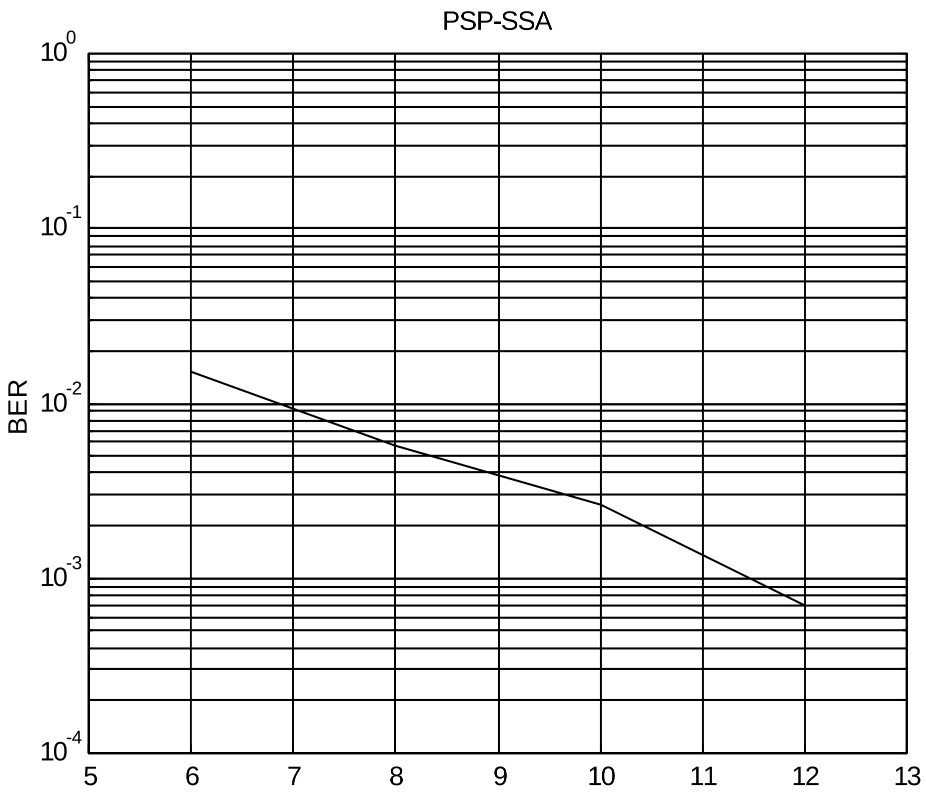
β. Ενώ για το κανάλι της ακραίας δυναμικής με $f_dT_s = 0.005$ και για ίδιο αριθμό πακέτων έχουμε :



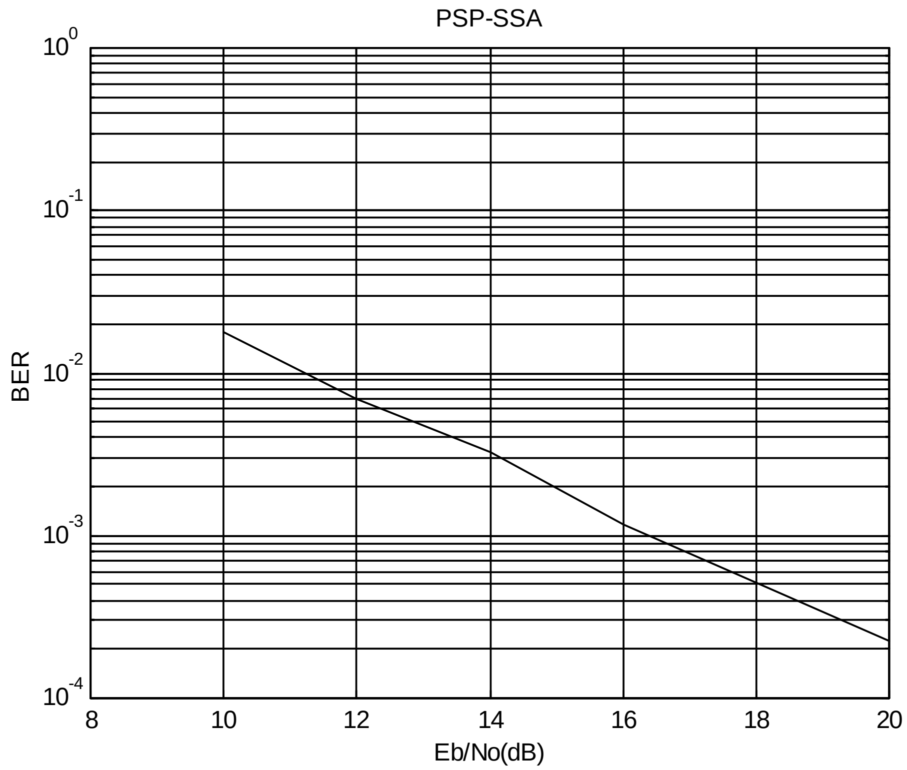
PSP-SSA

Ο αλγόριθμος PSP-SSA του οποίου η λειτουργία περιγράφηκε στην παράγραφο 1.7 χρησιμοποιείται εδώ στην πλέον πρακτική μορφή του, δηλαδή στην περίπτωση όπου D = L. Στην περίπτωση αυτή οι απαιτήσεις του αλγόριθμου σε μνήμη και σε επεξεργασία των πληροφοριών είναι εξαιρετικά περιορισμένες τόσο σε σχέση με την PSP-MLSE όσο και σε σχέση με τη μορφή που παίρνει ο ίδιος ο αλγόριθμος για $D > L$. Αυτό οφείλεται κυρίως στον άμεσο τρόπο με τον οποίο χειρίζεται τη συσσωρευμένη πληροφορία. Επιπλέον ο PSP-SSA έχει το χαρακτηριστικό ότι εξάγει χαλαρές αποφάσεις πάνω στην ακολουθία εισόδου με σταθερό ρυθμό, ιδιότητα που απαιτείται από διάφορες εφαρμογές. Η απόδοση του για τους δύο τύπους καναλιού που χρησιμοποιήθηκαν στις προσομοιώσεις παρουσιάζεται στις παρακάτω γραφικές παραστάσεις (για τον ίδιο αριθμό πακέτων όπως και στη προηγούμενη περίπτωση):

α. Πολύ γρήγορα μεταβαλλόμενο κανάλι $f_dT_s = 0.001$.



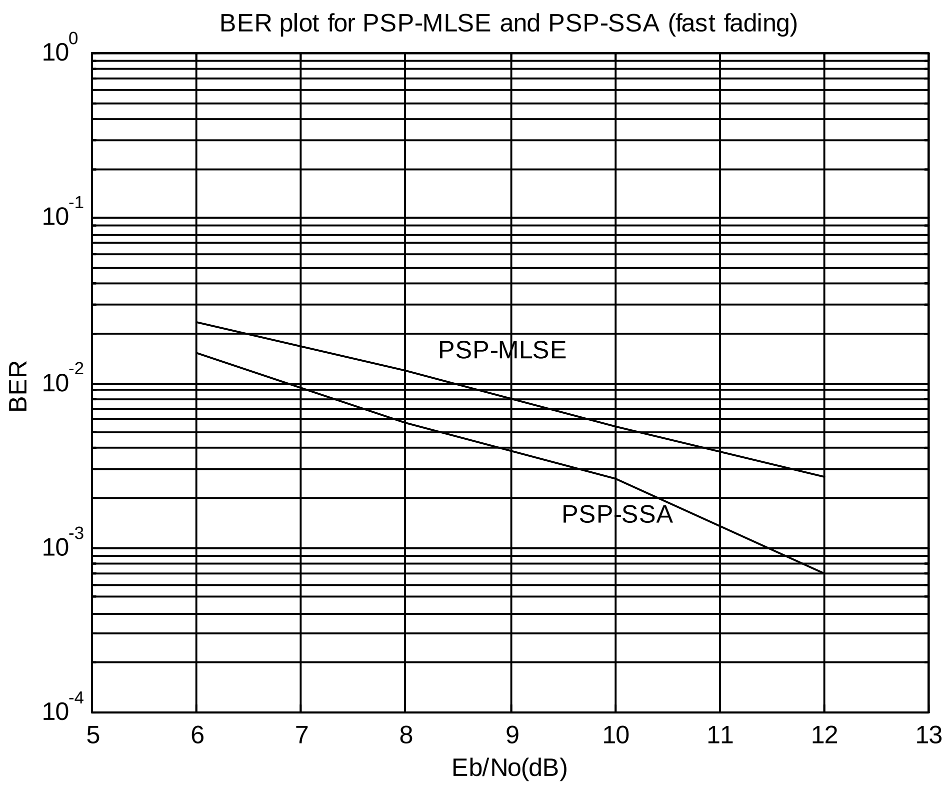
β. Κανάλι ακραίας δυναμικής $f_d T_s = 0.005$.



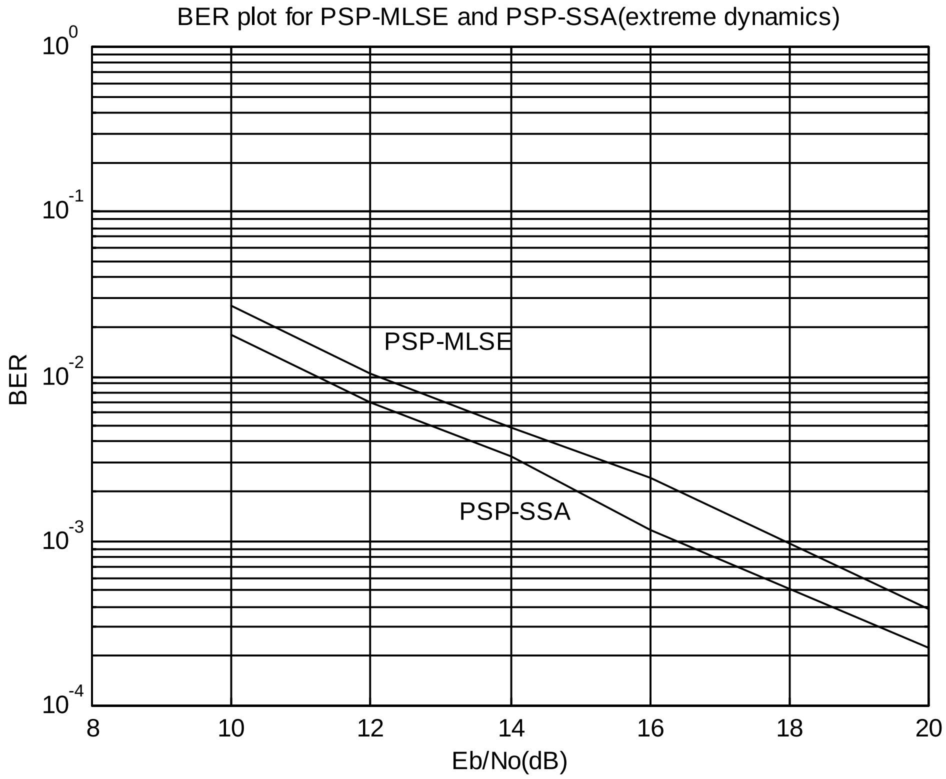
3.5 ΣΥΓΚΡΙΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

Η εργασία αυτή σκοπό έχει να συγκρίνει τους αλγόριθμους PSP-MLSE και PSP-SSA, ως προς την απόδοση τους, όταν αυτοί χρησιμοποιούνται σε ένα τηλεπικοινωνιακό σύστημα κάτω από δυσχερείς συνθήκες μετάδοσης. Οι παρακάτω γραφικές παραστάσεις παρουσιάζουν τα συγκριτικά αποτελέσματα για τους δύο αλγόριθμους που θα μας επιτρέψουν να εξάγουμε ορισμένα συμπεράσματα σχετικά με τη συμπεριφορά και τις σχετικές τους επιδόσεις.

Συγκριτικά αποτελέσματα για το fast fading κανάλι :



Συγκριτικά αποτελέσματα για το κανάλι ακραίας δυναμικής :



Εξετάζοντας τις γραφικές παραστάσεις μπορούμε να κάνουμε τις παρακάτω παρατηρήσεις :

α. Η απόδοση του PSP-SSA προκύπτει ότι είναι καλύτερη απ'αυτή του PSP-MLSE για οποιαδήποτε τιμή του λόγου $\frac{E_b}{N_0}$ και για τα δύο είδη καναλιού.

β. Η απόδοση και των δύο αλγορίθμων είναι αρκετά ικανοποιητική στην περίπτωση του fast fading καναλιού ακόμα και για σχετικά χαμηλές τιμές του λόγου $\frac{E_b}{N_0}$. Αντίθετα στη περίπτωση του καναλιού ακραίας δυναμικής και οι δύο αλγόριθμοι εμφανίζουν φτωχή απόδοση για μικρές τιμές του λόγου $\frac{E_b}{N_0}$.

Η απόδοση και των δύο αλγόριθμων είναι δυνατό να ερμηνευθεί σύμφωνα με το ακόλουθο σκεπτικό :

Ο χαμηλός λόγος $\frac{E_b}{N_0}$ σε συνδυασμό με τη γρήγορη μεταβολή του καναλιού οδηγούν τον αλγόριθμο RLS σε ανακριβή ή εντελώς εσφαλμένη εκτίμηση του καναλιού. Συνέπεια του παραπάνω γεγονότος είναι να μην επιβιώνουν στο επόμενο χρονικό βήμα οι επιθυμητές μεταβάσεις. Το παραπάνω φαινόμενο μπορεί να αποβεί καταστροφικό καθώς οδηγεί σε μια μετάδοση του σφάλματος και στις επόμενες μεταβάσεις «καταστρέφοντας» ουσιαστικά το μονοπάτι του διαγράμματος trellis που αντιστοιχεί στο μεταδιδόμενο πακέτο συμβόλων. Σε μια τέτοια περίπτωση εμφανίζεται ένας μεγάλος αριθμός σφαλμάτων και είναι απαραίτητη η μετάδοση της ακολουθίας εκπαίδευσης του επόμενου πακέτου ώστε να ανακτηθεί με ακρίβεια το κανάλι εκ νέου. Έτσι, όπως προέκυψε και κατά τη διάρκεια των προσομοιώσεων η αύξηση του BER οφείλεται κατά κύριο λόγο σε πακέτα που έχουν καταστραφεί σχεδόν ολοκληρωτικά, από αδυναμία του RLS να ανιχνεύσει μια μεταβολή που γίνεται στο κανάλι μετά τη διέλευση της ακολουθίας εκπαίδευσης και όχι σε πακέτα με μικρό αριθμό σφαλμάτων. Στη εξάλειψη πακέτων με μικρό αριθμό σφαλμάτων συμμετέχει αισθητά και το σύστημα συνελικτικής κωδικοποίησης – block interleaver που έχει τη δυνατότητα να διορθώσει αρκετά συνεχόμενα σφάλματα στην ακολουθία μετάδοσης.

Η αύξηση του λόγου $\frac{E_b}{N_0}$ οδηγεί σε ενίσχυση του σήματος έναντι του θορύβου και διευκολύνει αισθητά τον αλγόριθμο RLS να εκτιμήσει σωστά το κανάλι. Επιπλέον η PSP μορφή που έχουν και οι δύο αλγόριθμοι μειώνει την πιθανότητα αυξημένης εμφάνισης του παραπάνω φαινομένου (χωρίς όμως να μπορεί να την εξαλείψει).

Μπορεί να παρατηρήσει ακόμη κανείς ότι η απόδοση για το fast fading κανάλι δεν είναι δραματικά καλύτερη απ'αυτή για το κανάλι ακραίας δυναμικής και για τους δύο αλγόριθμους. Η παρατήρηση αυτή αιτιολογείται αν λάβουμε υπόψη μας ότι σε περίπτωση που το fast fading κανάλι βρεθεί σε κατάσταση βαθειάς εξασθένησης, θα μείνει σ'αυτή περισσότερο χρόνο από ότι το κανάλι ακραίας δυναμικής σε μια αντίστοιχη περίπτωση. Όσο όμως το κανάλι είναι σε κατάσταση deep fade τόσο πιο ευάλλωτος είναι ο δέκτης στο θόρυβο, γεγονός που αυξάνει σημαντικά την πιθανότητα σφάλματος. Από την άλλη πλευρά βέβαια το κανάλι ακραία δυναμικής μένει λιγότερο σε κατάσταση deep fade, αλλά εισέρχεται σ'αυτή τη κατάσταση πολύ πιο συχνά και με απότομες αλλαγές δυσκολεύοντας τη λειτουργία του δέκτη.

ΚΕΦΑΛΑΙΟ 4^{ο}

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

4.1 ΣΥΜΠΕΡΑΣΜΑΤΑ

Στην εργασία αυτή εισάγεται η μέθοδος PSP στους είδη γνωστούς αλγόριθμους MLSE και SSA. Η επίδοση της νέας μορφής των αλγορίθμων αποκτάται με την προσομοίωση ενός τηλεπικοινωνιακού συστήματος με την εξής δομή: (α) QPSK modulation (β) Convolutional Coding-Block Interleaving. Οι δύο αλγόριθμοι δοκιμάζονται τόσο για ένα γρήγορα μεταβαλλόμενο κανάλι όσο και για ένα κανάλι ακραίας δυναμικής δείχνοντας τη δυνατότητα που έχουν να ανταπεξέρχονται σε δύσκολες συνθήκες μετάδοσης. Οι προσομοιώσεις ανέδειξαν τον αλγόριθμο PSP-SSA, που εδώ χρησιμοποιήθηκε στην πλέον απλή, αλλά και πιο πρακτική μορφή του, ως ικανότερο να αποδώσει παρουσία διασυμβολικής παρεμβολής και θορύβου σε ένα frequency selective Rayleigh fading κανάλι με μεγάλη τιμή της διασποράς Doppler f_d. Το χαρακτηριστικό της απλούστερης υλοποίησης σε συνδυασμό με την καλύτερη απόδοση καθιστούν τον αλγόριθμο PSP-SSA μια καλή επιλογή για την ανάκτηση της μεταδιδόμενης πληροφορίας στο δέκτη. Ο αλγόριθμος PSP-MLSE παρουσίασε ικανοποιητικά αποτελέσματα αλλά δεν κατόρθωσε να ξεπεράσει τον PSP-SSA, γεγονός που σε συνδυασμό με τα χαρακτηριστικά της υψηλότερης πολυπλοκότητας και τις μεγάλες απαιτήσεις σε μνήμη τον φέρνουν στη δεύτερη θέση.

Η επίδοση των αλγορίθμων PSP-MLSE και PSP-SSA συνδέεται άμεσα με την ικανότητα σύγκλισης του RLS στην κρουστική απόκριση του καναλιού. Ο RLS κατορθώνει εν γένη να προσεγγίζει ικανοποιητικά τα taps του καναλιού, εξασφαλίζοντας έτσι το σωστό υπολογισμό των μετρικών κόστους των δύο αλγορίθμων. Στις περιπτώσεις όμως εκείνες όπου ο RLS αποτυγχάνει να προσεγγίσει τις σωστές τιμές των taps το σύστημα οδηγείται σε μεγάλο αριθμό σφαλμάτων. Η ικανότητα ταχύτατης σύγκλισης του RLS, ιδιαίτερα κατά τη μετάδοση ακολουθίας εκπαίδευσης, συμβαδίζει με την υψηλή πολυπλοκότητα και τις μεγάλες απαιτήσεις σε μνήμη του αλγόριθμου.

Σημαντική είναι επίσης η συνεισφορά του συνδυασμού συνελικτικού κωδικοποιητή-block interleaver στη διόρθωση εσφαλμένων αποφάσεων του δέκτη. Η αποτελεσματικότητα του παραπάνω σχήματος κωδικοποιητή-interleaver έγκειται στην ικανοποιητική αντιμετώπιση «εκρήξεων» σφαλμάτων που εμφανίζονται κατά τη μετάδοση της πληροφορίας.

4.2 ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

Ο συνδυασμός της επεξεργασίας ανά επιβιώνουσα μετάβαση με τον υποβέλτιστο αλγόριθμο χαλαρών αποφάσεων και η σύγκριση με τον ήδη γνωστό PSP-MLSE αποτέλεσαν το κυρίως αντικείμενο αυτής της εργασίας. Προκειμένου να εξάγουμε συμπεράσματα σχετικά με την απόδοση των δύο αλγορίθμων, θεωρήσαμε την περίπτωση ενός Rayleigh fading καναλιού με γνωστή έκταση διασυμβολικής παρεμβολής. Η υπόθεση αυτή μας δίνει τη δυνατότητα να χρησιμοποιήσουμε ένα διάγραμμα trellis πλήρες σε καταστάσεις. Μια άμεση επέκταση της εργασίας αυτής θα ήταν η μνήμη του διαύλου να είναι και αυτή άγνωστη ή υπερβολικά μεγάλη οπότε να περιοριστούμε σε εκτίμηση ακολουθίας μειωμένων καταστάσεων (Reduced State Sequence Estimation). Σ'αυτή την περίπτωση ο αποκωδικοποιητής πρέπει να εκτιμήσει εκτός από τις παραμέτρους του διαύλου και την υπολειπόμενη διασυμβολική παρεμβολή ώστε να είναι δυνατός ο ακριβέστερος υπολογισμός των μετρικών μετάβασης.

Η σημαντική προσφορά του συνελικτικού κωδικοποιητή και του block interleaver στη συνολική απόδοση του συστήματος που προσομοιώθηκε δίνουν το έναυσμα για τη χρήση πιο ευέλικτων και αποτελεσματικών μεθόδων κωδικοποίησης όπως είναι η Turbo κωδικοποίηση. Η βασική αρχή αυτού του νέου σχήματος κωδικοποίησης/αποκωδικοποίησης είναι η χρήση μιας παράλληλης ακολουθίας από τουλάχιστον δύο κωδικοποιητές, με ένα interleaver ανάμεσα τους. Η αποκωδικοποίηση στηρίζεται στη διαδοχική αποκωδικοποίηση των στοιχειωδών κωδίκων και στη μετάδοση της λεγόμενης εξωτερικής πληροφορίας (extrinistic information) στο επόμενο στάδιο αποκωδικοποίησης. Ακόμα και όταν οι στοιχειώδεις κώδικες που χρησιμοποιούνται είναι απλοί, η Turbo κωδικοποίηση είναι ικανή να επιτύχει απόδοση κοντά στο όριο του Shannon -αν γίνει χρήση αρκετά μεγάλων interleavers- και ταυτόχρονα να επιτύχει ένα πολύ μικρό bit error rate. Θα παρουσίαζε ιδιαίτερο ενδιαφέρον ο προσδιορισμός της απόδοσης των αλγορίθμων PSP-MLSE και PSP-SSA όταν γίνεται χρήση της ισχυρής αυτής μεθόδου κωδικοποίησης.

ΠΑΡΑΡΤΗΜΑ Α

ΚΩΔΙΚΕΣ ΠΡΟΣΟΜΟΙΩΣΕΩΝ ΣΕ C++

Στο παράρτημα αυτό παρουσιάζονται και σχολιάζονται οι κώδικες των προγραμματων που υλοποιήθηκαν στην γλώσσα C++. Οι κώδικες παρουσιάζονται σε αντιστοιχία με τα blocks του τηλεπικοινωνιακού συστήματος που προσομοιώθηκε.

```
class bit_source{
   enum { size = 228};
        double probability_1;   //probability to send 1
        int bit_stream[size];

public:

        bit_source(double p);
        ~bit_source();
        int * bit_generator();   //function that generates the bits
   int print_stream();      //prints the bits that enter the system

};

bit_source::bit_source(double p){
        probability_1=p;
}

bit_source::~bit_source(){}

int * bit_source::bit_generator(){

  srand( (unsigned)time( NULL ) );
  double temp=0;
  for(int i=0;i<(size-1);i++){
          temp=((double)rand())/((double)RAND_MAX);
          if(temp<=probability_1)
                  bit_stream[i]=1;
          else
                  bit_stream[i]=0;
  }
  bit_stream[(size-1)]=0;       //zero terminated sequence of bits
                         //..last value is always zero
 return bit_stream;
}

int bit_source::print_stream(){

          for(int i=0; i<size ;i++)
                  cout << bit_stream[i];

          cout << endl;
return 1;
}
```

```
class conv_enc {
		enum {size = 228};
		 int M[3];   //shift register
		 int v1;     //output 1
		 int v2;     //output 2
	 int output_data[size*2];

public:
		conv_enc();

		~conv_enc();

		friend int xor(int const & left,int const & right);
		//exclusive-OR function

	int * conv_code_generator(int const * input_data);
	//function that genetates the convolutional code
};

conv_enc::conv_enc(){

		for(int i=0;i<3;i++)
			M[i]=0;

		 v1=0;
		 v2=0;

		 for(int j=0;j<(2*size);j++)
			output_data[j]=0;


}

conv_enc::~conv_enc(){};

int xor(int const & left,int const & right){
  int result=0;;
		if ((left==0)&&(right==0))
			result=0;
		if ((left==0)&&(right==1))
			result=1;
		if ((left==1)&&(right==0))
			result=1;
		if ((left==1)&&(right==1))
			result=0;
		return result;
}

int * conv_enc::conv_code_generator(int  const * input_data){

		for(int k=0;k<3;k++)   //clean the encoder from the previous
			M[k]=0;           //sequence (return to zero state)

		int counter=0;

		for(int i=0;i<size;i++){
			M[2]=M[1];
```

```
			M[1]=M[0];
			M[0]=input_data[i];		//shifting of values
			v1=xor(M[0],M[2]);
			v2=xor(v1,M[1]);		//output calcualations
			output_data[counter]=v1;
			counter++;
	output_data[counter]=v2;
			counter++;
		}
return output_data;
}




//The following class performs the modulation of
//convolutional encoder symbols
class qpsk_modulator{

		complex<double> *signals; //this array holds the outpu signal(symbol)
					//that corresponds to the input encoder code word
public :

		qpsk_modulator();
		~qpsk_modulator();
		complex<double> * qpsk_modulate(int * data);//calculates the
							//symbols
};

qpsk_modulator::qpsk_modulator(){
		signals = new complex<double> [228];

		for(int i=0;i<228;i++){
			signals[i].real(0);
			signals[i].imag(0);
		}
}

qpsk_modulator::~qpsk_modulator(){}

complex<double> * qpsk_modulator::qpsk_modulate(int * data){
int data_idx = 0;
//we consider a normalized bit energy equal to 1
		for(int i=0;i<228;i++){
//if the most significant bit of the code word is 0 the symbol
//has a -1 real part elseif 1 the real part is 1
			if (data[data_idx]==0){
				signals[i].real(-1);}
			else{
				signals[i].real(1);}

			data_idx++;
//if the least significant bit of the code word is 0 the symbol
//has a -1 real part elseif 1 the real part is 1
			if (data[data_idx]==0){
				signals[i].imag(-1);}
			else{
```

```
                        signals[i].imag(1);}
    data_idx++ ;
        }
//the output of this function is the address of the array of complex signals
        return (complex<double> *) signals;
}






//the following class performs deinterleaving of the symbol sequence
//and also removes the training sequence

class Interleaver{

        complex<double> * input_signals;
        complex<double> ** interleaver;
        complex<double> * output_signals;
   complex<double> train_symbol;

public:
        Interleaver();
   ~Interleaver(){}
        complex<double>* interleave(complex<double>* input_signals);
};

Interleaver::Interleaver(){

        interleaver =new complex<double> *[19];
        for(int i=0;i<19;i++)
                interleaver[i]=new complex<double> [12];

        output_signals = new complex<double> [254];

        train_symbol.real(-1);
        train_symbol.imag(-1);
}



complex<double>* Interleaver::interleave(complex<double>* input_signals){
//put signals  to interleaver array
        int s_idx = 0;
        for(int i=0;i<19;i++){
                for(int j=0;j<12;j++){
                        interleaver[i][j]=input_signals[s_idx];
                        s_idx++;
                }
        }
//interleave signals

//the first 26 symbols are the training sequence
        int out_si_idx = 0;
        for(i=0;i<26;i++){
                output_signals[out_si_idx]=train_symbol;
                out_si_idx++;
        }
```

```
            for(int j=0;j<12;j++){
                        for(int i=0;i<19;i++){
                                    output_signals[out_si_idx]=interleaver[i][j];
                                    out_si_idx++;
                        }
            }
            return output_signals;
}




extern bool change_taps = true;
extern long int symbol_no=0;
//The following class implements a Rayleigh fading channel
class channel{
            enum {packet = 254};
            complex<double> memory[2]; //a memory of 2 symbols for the channel
            complex<double> h[3];      //the tap coefficients for the discrete channel model
            complex<double>* output_signals;//the corrupted signals packet
public:
            channel();
            ~channel(){}
//next function produces the output symbols of the channel-----------
   complex<double>* channel_output(complex<double> *input_signals,
                                             double No);
};

 channel::channel(){
   output_signals = new complex<double> [packet];
            complex<double> temp(0.0,0.0);
            memory[0]=temp;
   memory[1]=temp;
            h[0]=temp;
            h[1]=temp;
            h[2]=temp;

}

//read from the following files(read)-------------------------------------
 ifstream h0_realin("zh0real.txt");
 ifstream h0_imagin("zh0imag.txt");
 ifstream h1_realin("zh1real.txt");
 ifstream h1_imagin("zh1imag.txt");
 ifstream h2_realin("zh2real.txt");
 ifstream h2_imagin("zh2imag.txt");
//-------------------------------------------------------------------



complex<double>* channel::channel_output(complex<double> *input_signals,
                                                                                    double No){
double real_noise=0.0,imag_noise = 0.0;
            complex<double> noise(0.0,0.0);
//initialize output signals and memory to 0+0j
            complex<double> temp(0.0,0.0);
            memory[0]=temp;
   memory[1]=temp;
```

```
            for(int j=0;j<packet;j++)
        output_signals[j] = (0.0,0.0);
//---------------------------------------------------------
            double   AWGN_std_deviation = sqrt(No/2);
    double  trans=0.0;
            for(int i=0;i<packet;i++){
               if((symbol_no%1143000) == 0){

                        h0_realin.close();
          h0_imagin.close();
          h1_realin.close();
          h1_imagin.close();
          h2_realin.close();
          h2_imagin.close();

          h0_realin.open("zh0real.txt");
          h0_imagin.open("zh0imag.txt");
          h1_realin.open("zh1real.txt");
          h1_imagin.open("zh1imag.txt");
          h2_realin.open("zh2real.txt");
          h2_imagin.open("zh2imag.txt");
                       }

                       //check if the channel will change during the transmission of this symbol
                       if(change_taps==true){
          h0_realin >> trans;
                                 h[0].real(trans);
          h0_imagin >> trans;
          h[0].imag(trans);
          h1_realin >> trans;
                                 h[1].real(trans);
          h1_imagin >> trans;
          h[1].imag(trans);
          h2_realin >> trans;
                                 h[2].real(trans);
          h2_imagin >> trans;
          h[2].imag(trans);
/*                               cout << h[0]<<endl;
                                 cout << h[1]<<endl;
          cout << h[2]<<endl;
          cout << endl;*/
                                 change_taps=false;
                       }

real_noise = gaussian(0.0,AWGN_std_deviation);
imag_noise = gaussian(0.0,AWGN_std_deviation);
noise.real(real_noise);
noise.imag(imag_noise);
//out_noise << noise << "     ";
//calculate the output symbol according to the discrete channel model
output_signals[i]=(input_signals[i]*h[0]+memory[0]*h[1]+memory[1]*h[2]+noise);//+noise
//update the memory of the channel after the transmission of the new symbol
                       memory[1]=memory[0];
                       memory[0]=input_signals[i];

                       symbol_no++;
               if((symbol_no%1000) == 0)
                        change_taps = true;
            }
```

```
        return (complex<double>*)output_signals;
}




#define INFINITE 1.7e+308

//ofstream surv_P("xxsurv_P.txt");
//ofstream surv_h("xxsurv_h.txt");

class channel_state{
public:
        int    prev_state; //the survivor previous state
        double cost;       //accumulated cost
        complex<double>   receiver_output; //the output of the survivor for traceback that
                // corresponts to the transition prev->current state
   complex<double> *h;  //estimates of channel tap coefficients
        complex<double> **P;  //Inverse of correlation matrix
   complex<double> kal[3];
        complex<double> error;     //error between the input signal and the
                        // estimated value
        complex<double> *u;    //this vector holds the last 2 hypothetical
                        //channel inputs of a survivor
        channel_state();
        ~channel_state();
int set_state(double Cost,int Prev_state,complex<double> Receiver_output);
};


channel_state::channel_state(){
   cost = INFINITE;
        prev_state= -1;
        receiver_output=(0.0,0.0) ;
   h= new complex<double> [3];

        for(int i=0;i<3;i++)
                h[i] =(0.0,0.0) ;

        for(i=0;i<3;i++)
                kal[i] =(0.0,0.0) ;

//alloc mem for current and previous hypothetical symbols
        u = new complex<double> [3];
        for(i=0;i<3;i++)
                u[i] =(0.0,0.0) ;

//allocate memory for P
        P = new complex<double> * [3];
        for(i=0;i<3;i++)
                P[i] = new complex<double> [3];

        //Initialize matrix P diagonal elements---------------
        for(i=0;i<3;i++){
                for(int j=0;j<3;j++){
                        P[i][j]=(0.0,0.0);
                }
```

```
            }
    for(i=0;i<3;i++){
                        P[i][i].real(1000);
                         P[i][i].imag(0.0);
            }
//--------------------------------------------------
//initial  channel tap coefficients-------------------
                        h[0].real(0);h[0].imag(0);
                        h[1].real(0);h[1].imag(0);
                        h[2].real(0);h[2].imag(0);
//--------------------------------------------------
//initialize channel memory--------------------------
                        for(i=1;i<3;i++)
                        u[i]=(0.0,0.0);
//--------------------------------------------------
}
channel_state::~channel_state(){
delete [] h;
//delete [] Kal;
delete [] u;

for(int i=0;i<3;i++)
  delete [] P[i];
     delete [] P ;

}

int channel_state::set_state(double Cost,int Prev_state,complex<double> Receiver_output){
            cost=Cost;
            prev_state=Prev_state;
            receiver_output=Receiver_output;
    return 1;
}

//class branch implements the branch of transition from a state to another

class channel_branch {
public:
            int begin_state;
            int ending_state;
            complex<double> hypoth_channel_input; //the hypothetical input that can cause that transition
    complex<double> Uk_L;
            channel_branch();
            ~channel_branch(){};
int set_branch(int Begin_state,int Ending_state,
                                    complex<double> hypoth_Channel_input,complex<double> Uk_l);
};

channel_branch::channel_branch(){

             begin_state=-1;
             ending_state=-1;
             hypoth_channel_input=(0.0,0.0);

}

int channel_branch::set_branch(int Begin_state,int Ending_state,
                                    complex<double> hypoth_Channel_input,complex<double> Uk_l){
              begin_state = Begin_state;
              ending_state = Ending_state;
```

```
		hypoth_channel_input = hypoth_Channel_input;
	Uk_L=Uk_l;

 return 1;
}

//----------------------------------------------------------
class channel_trellis_diagram{
public:
	enum {no_states=16,L=2,max_no_transitions=64,
		no_possible_trans=4,packet = 254};

		channel_state ** trellis;//[no_states][packet+1];
		channel_branch  * tran_matrx;//[max_no_transitions];
		complex<double> * train_seq;//[26];
	complex<double> * psp_out;//[packet];

		channel_trellis_diagram();
		~channel_trellis_diagram();
		complex<double>* MLSE_receive(complex<double> *Symbol);
	complex<double>* SSA_receive(complex<double> *Symbol);
};


//SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
//SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAA

complex<double>* channel_trellis_diagram::SSA_receive(complex<double>* Symbol){

//variable definition for Viterby----------------------------------------------------

 complex<double> * symbol = Symbol;
 complex<double> y(0.0,0.0);
 complex<double>* temp_conj_h = new complex<double> [3];

 double trans_cost = 0.0,tempabs = 0.0;//the cost for a transition
 {//short namespace
	for(int i=0;i<no_states;i++){
		for(int j=0;j<(packet+1);j++)
			trellis[i][j].cost = INFINITE;//initialize cost for the packets after the first
	}
 }
//-----------------------------------------------------------------------
//variable definition for RLS---------------------------------------------
int st = 0;
complex<double>  err(0.0,0.0);
complex<double> par1(0.0,0.0);

double lambda =0.95;
double lambda_rec=1.0/lambda;

complex<double>* temp1  = new complex<double> [3];
complex<double>* conj_u = new complex<double> [3];
complex<double>* temp_h = new complex<double> [3];
complex<double>* temp_Kal = new complex<double> [3];
complex<double>* conj_h = new complex<double> [3];

complex<double>** temp3 = new complex<double> *[3];
```

```
  for(int ig=0;ig<3;ig++)
    temp3[ig]= new complex<double> [3];

complex<double>** temp4 = new complex<double> *[3];
  for(ig=0;ig<3;ig++)
            temp4[ig]= new complex<double> [3];

complex<double>** temp_P = new complex<double> *[3];
  for(ig=0;ig<3;ig++)
            temp_P[ig]= new complex<double> [3];
//--------------------------------------------------
/////////////////////////////////////////////////////////////////////////
//symbol index
int si =0;
// train_seq_index  0=<tr_idx<26
int tr_idx = 0;
complex<double> train_symbol(-1,-1);

//initialize t=0 costs to zero
for(int ix=0;ix<no_states;ix++)
    trellis[ix][0].cost=0;

//Parse the trellis columns-------------------------------------------
  for(int t=0;t<(packet);t++){
//for every state in a column ---------------------------------------
    for(int state=0;state<no_states;state++){
//check transition matrix to get the ending states of the state-------
      for(int j=0;j<max_no_transitions;j++){
        if(state == tran_matrx[j].begin_state){
//compute the conjugate vector of h-----------------------------------
        for(int indc=0;indc<3;indc++){
                              temp_conj_h[indc].real(trellis[state][t].h[indc].real());
          temp_conj_h[indc].imag(-trellis[state][t].h[indc].imag());
                        }
//compute output estimation-------------------------------------------
                        trellis[state][t].u[0]=tran_matrx[j].hypoth_channel_input;
        y=vec_mult(trellis[state][t].u,temp_conj_h);
//--------------------------------------------------------------------
//compute transition cost metric--------------------------------------
    if((t<26)&&(trellis[state][t].u[0]==train_symbol)){
             trans_cost = 0;        //the transition cost for a train symbol is zero
            }                          //because we know them a priori(only the sequence of
    else{                         //has total cost 0 so it will be the survivor sequence)
                           tempabs = 0;trans_cost = 0;
                                          tempabs = abs(symbol[si] - y);
             trans_cost= pow(tempabs,2);
            }
//-----------------------------------------------------
//save under condition in trellis[next_state for 1 or 0 input][t+1] cost,previous state and decoder otput
                        if(trellis[tran_matrx[j].ending_state][t+1].cost==INFINITE){
                          //no other branch has ended in that state before
              //save accumulated cost
                        trellis[tran_matrx[j].ending_state][t+1].cost =
                                  trellis[tran_matrx[j].begin_state][t].cost+trans_cost;
                        //update previous state
                        trellis[tran_matrx[j].ending_state][t+1].prev_state=state;
                //update receiver output info
                        trellis[tran_matrx[j].ending_state][t+1].receiver_output =
                                  tran_matrx[j].hypoth_channel_input;
        //update u vector
```

```
                        trellis[tran_matrx[j].ending_state][t+1].u[1]=
                                trellis[tran_matrx[j].begin_state][t].u[0];
                   trellis[tran_matrx[j].ending_state][t+1].u[2]=
                                trellis[tran_matrx[j].begin_state][t].u[1];
                                }
                        else{
     //other branches have reached this state and we have to find the survivor
                        double canditate_cost=trellis[tran_matrx[j].begin_state][t].cost+
                                trans_cost;
             //if current branch produces a lower accumulated metric update state info
           if(canditate_cost < trellis[tran_matrx[j].ending_state][t+1].cost){
                        //save accumulated cost
                          trellis[tran_matrx[j].ending_state][t+1].cost=canditate_cost;
          //update previous state
                          trellis[tran_matrx[j].ending_state][t+1].prev_state=state;
                //update receiver output info
                          trellis[tran_matrx[j].ending_state][t+1].receiver_output =
                          tran_matrx[j].hypoth_channel_input;
          //update u vector
                          trellis[tran_matrx[j].ending_state][t+1].u[1]=
                                        trellis[tran_matrx[j].begin_state][t].u[0];
            trellis[tran_matrx[j].ending_state][t+1].u[2]=
                                    trellis[tran_matrx[j].begin_state][t].u[1];
                               }
                        }
                        }
             }
            }

// RLS algorithm for updating channel coefficients
            for(int idx=0;idx<no_states;idx++){
//-------------------------------------------------------------
//for every state of t+1 find the previous state and Hypothetical channel input
                       st=trellis[idx][t+1].prev_state;
              trellis[st][t].u[0] = trellis[idx][t+1].u[1];
//-------------------------------------------------------------
//compute the conjugate vector of h-----------------------------
        for(int ind=0;ind<3;ind++){
                             conj_h[ind].real(trellis[st][t].h[ind].real());
           conj_h[ind].imag(-trellis[st][t].h[ind].imag());
                       }
//-------------------------------------------------------------
//compute output estimation again-------------------------------
                       y=vec_mult(trellis[st][t].u,conj_h);
//--------------------------------------------------------------
//compute error-----------------------------------------------
                             err =symbol[si] - y;
//--------------------------------------------------------------
//produce Kalman gain vector-------------------------------------
                       for(int g=0;g<3;g++){   //conjg of signal vector u
                             conj_u[g].real(trellis[st][t].u[g].real());
           conj_u[g].imag(-trellis[st][t].u[g].imag());
                       }
        vec_mult_matr(conj_u,trellis[st][t].P,temp1);//u_transpose*P
                       par1 = vec_mult(temp1,trellis[st][t].u);//u_transpose*P*u
                       matr_mult_vec(trellis[st][t].P,trellis[st][t].u,temp_Kal);
//P*u_transpose
                       for(g=0;g<3;g++){
                             temp_Kal[g]=temp_Kal[g]/(lambda+par1);
           trellis[st][t].kal[g]=temp_Kal[g];
```

```
                        }
//-----------------------------------------------------------------
//update matrix P-------------------------------------------------
       for(int ii=0;ii<3;ii++){
                               for(int jj=0;jj<3;jj++){
                                        temp_P[ii][jj]=trellis[st][t].P[ii][jj];
                               }
                        }
                        vecs_to_matr(temp_Kal,conj_u,temp3);//temp3=kal*u
                        matr_to_matr(temp3,temp_P,temp4);//temp4 = temp3*P
       matr_differ(temp_P,temp4);//P=P-temp4
                        const_matr(temp_P,lambda_rec);//P=P*lambda_rec->reciprocal of lambda
//-----------------------------------------------------------------
//update tap coefficients------------------------------------------
       complex<double> conj_err(0.0,0.0);
                        conj_err.real(err.real());
                        conj_err.imag(-err.imag());
                        const_vec(temp_Kal,conj_err);//kal = Kal*conj_err
        for(int ifx=0;ifx<3;ifx++)
                               temp_h[ifx]=trellis[st][t].h[ifx];
//!!!!!!!!!!!!TWO ENDING STATES CAN HAVE THE SAME PREVIOUS STATE
//!!!!!!!!!!!!FOR NO REASON WE ARE ALLOWED TO CHANGE INFO ABOUT
//!!!!!!!!!!!!THE TAPS :SO NO h=h+kal*err BUT temp_h = h + kal*err; BECAUSE
//WE WILL USE VECTOR h OF trellis[st][t] AGAIN as a previous state later same for P
                          add_vecs(temp_h,temp_Kal);//h=h+kal
//-----------------------------------------------------------------
//update survivors ending state info-------------------------------
//update  ending state P matrix
                        for(int gx=0;gx<3;gx++){
                               for(int gs=0;gs<3;gs++)
              trellis[idx][t+1].P[gx][gs]=temp_P[gx][gs];
                        }
//update  ending state tap coefficient vector
                               for(int igs=0;igs<3;igs++)
              trellis[idx][t+1].h[igs]=temp_h[igs];

//-----------------------------------------------------------------
}//END RLS

//Find state with the least augmented metric(wich equals for D=L the
//additive branch metrics)i.e. it is the survivor with the least
//accumulated metric
    if(t>=29){
                        int state_idx1=0;
                        int min_aug_cost_state=0;
                        double min_cost =trellis[state_idx1][t].cost;
                        for(state_idx1=1;state_idx1<no_states;state_idx1++){
                               if(min_cost>trellis[state_idx1][t].cost){
                                        min_cost = trellis[state_idx1][t].cost;
                                        min_aug_cost_state=state_idx1;
                               }
                        }

//Find Uk_L symbol using the state with the minimum augmented metric
//and its previous state
   for(int tr_matr_idx1=0;tr_matr_idx1<max_no_transitions;tr_matr_idx1++){
     if((trellis[min_aug_cost_state][t].prev_state==tran_matrx[tr_matr_idx1].begin_state)&&
        (min_aug_cost_state==tran_matrx[tr_matr_idx1].ending_state)){
         psp_out[t-(L+1)]=tran_matrx[tr_matr_idx1].Uk_L;
                break;
```

```
             }
   }

           }//END if(t<=29)
//read the next  symbol---------------------------------------------
  si++;
//------------------------------------------------------------------
 }
//-----------------deallocate memory of arrays---------------------

  delete [] temp1;
  delete [] conj_u;
  delete [] temp_h;
  delete [] conj_h;
  delete [] temp_conj_h;
for(int im=0;im<3;im++)
  delete [] temp3[im];
     delete [] temp3 ;
for(im=0;im<3;im++)
  delete [] temp4[im];
     delete [] temp4 ;
for(im=0;im<3;im++)
  delete [] temp_P[im];
     delete [] temp_P ;
//set the first 26 symbols of psp_out equal to train_symbol
             for(int out_idx1=0;out_idx1<26;out_idx1++)
                         psp_out[out_idx1]=train_symbol;
//return inner receiver output
             return (complex<double>*) psp_out;
}//end of SSA_receiver-------------------------------------------------




//MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMMMMMMMMMMMMM
//LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
L
//SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
//EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
E



//The decode function performs decoding of the 228 convolutional code
//symbols(2 bits each).The algorithm finds the path that minimizes the accumulated
//metric for the whole sequence of symbols(it doesn't use the suboptimum
//approach of decoding a symbol after  proceding 5*L+1 stages in the
//trellis diagram)
complex<double> *channel_trellis_diagram::MLSE_receive(complex<double> * Symbol){

//variable definition for Viterby-----------------------------------------------------


 complex<double> * symbol = Symbol;
 complex<double> y(0.0,0.0);
 complex<double>* temp_conj_h = new complex<double> [3];

 double trans_cost = 0.0,tempabs = 0.0;//the cost for a transition
```

```
 {//short namespace
            for(int i=0;i<no_states;i++){
                        for(int j=0;j<(packet+1);j++)
                                    trellis[i][j].cost = INFINITE;//initialize cost for the packets after the first
            }
 }
//-------------------------------------------------------------------------
//variable definition for RLS---------------------------------------------
int st = 0;
complex<double>  err(0.0,0.0);
complex<double> par1(0.0,0.0);

double lambda =0.95;
double lambda_rec=1.0/lambda;

complex<double>* temp1  = new complex<double> [3];
complex<double>* conj_u = new complex<double> [3];
complex<double>* temp_h = new complex<double> [3];
complex<double>* temp_Kal = new complex<double> [3];
complex<double>* conj_h = new complex<double> [3];

complex<double>** temp3 = new complex<double> *[3];
  for(int ig=0;ig<3;ig++)
    temp3[ig]= new complex<double> [3];

complex<double>** temp4 = new complex<double> *[3];
  for(ig=0;ig<3;ig++)
            temp4[ig]= new complex<double> [3];

complex<double>** temp_P = new complex<double> *[3];
  for(ig=0;ig<3;ig++)
            temp_P[ig]= new complex<double> [3];
/*
ofstream out_error("xout_error.txt");
ofstream out_symbols("xout_symbols.txt");
ofstream h0real("xout_h0real.txt");
ofstream h0imag("xout_h0imag.txt");
ofstream h1real("xout_h1real.txt");
ofstream h1imag("xout_h1imag.txt");
ofstream h2real("xout_h2real.txt");
ofstream h2imag("xout_h2imag.txt");
ofstream out_cost("xout_cost.txt");
*/
//---------------------------------------------------
//////////////////////////////////////////////////////////////////////////
//symbol index
int si =0;
// train_seq_index  0=<tr_idx<26
int tr_idx = 0;
complex<double> train_symbol(-1,-1);

for(int ix=0;ix<no_states;ix++)//initialize t=0 costs to zero
   trellis[ix][0].cost=0;

//Parse the trellis columns-------------------------------------------
  for(int t=0;t<(packet);t++){
//for every state in a column
   for(int state=0;state<no_states;state++){
//check transition matrix to get the ending states of the state
     for(int j=0;j<max_no_transitions;j++){
```

```
        if(state == tran_matrx[j].begin_state){
//compute the conjugate vector of h------------------------------
       for(int indc=0;indc<3;indc++){
                                temp_conj_h[indc].real(trellis[state][t].h[indc].real());
         temp_conj_h[indc].imag(-trellis[state][t].h[indc].imag());
                        }
//compute output estimation-----------------------------------
                        trellis[state][t].u[0]=tran_matrx[j].hypoth_channel_input;
        y=vec_mult(trellis[state][t].u,temp_conj_h);
//-------------------------------------------------------
//compute transition cost metric--------------------------------
    if((t<26)&&(trellis[state][t].u[0]==train_symbol)){
            trans_cost = 0;        //the transition cost is for a train symbol is zero
           }                            //because we know them a priori(only the sequence of
    else{                           //has total cost 0 so it will be the survivor sequence)
                          tempabs = 0;trans_cost = 0;
                    tempabs = abs(symbol[si] - y);
            trans_cost= pow(tempabs,2);
           }
//-------------------------------------------------------
//save under condition in trellis[next_state for 1 or 0 input][t+1] cost,previous state and decoder otput
                        if(trellis[tran_matrx[j].ending_state][t+1].cost==INFINITE){
                          //no other branch has ended in that state before
              //save accumulated cost
                         trellis[tran_matrx[j].ending_state][t+1].cost =
                                   trellis[tran_matrx[j].begin_state][t].cost+trans_cost;
                         //update previous state
                         trellis[tran_matrx[j].ending_state][t+1].prev_state=state;
                 //update receiver output info
                         trellis[tran_matrx[j].ending_state][t+1].receiver_output =
                                   tran_matrx[j].hypoth_channel_input;
         //update u vector
                         trellis[tran_matrx[j].ending_state][t+1].u[1]=
                                     trellis[tran_matrx[j].begin_state][t].u[0];
                trellis[tran_matrx[j].ending_state][t+1].u[2]=
                                   trellis[tran_matrx[j].begin_state][t].u[1];
                                    }
                         else{
     //other branches have reached this state and we have to find the survivor
                         double canditate_cost=trellis[tran_matrx[j].begin_state][t].cost+
                                    trans_cost;
             //if current branch produces a lower accumulated metric update state info
          if(canditate_cost < trellis[tran_matrx[j].ending_state][t+1].cost){
                          //save accumulated cost
                             trellis[tran_matrx[j].ending_state][t+1].cost=canditate_cost;
         //update previous state
                             trellis[tran_matrx[j].ending_state][t+1].prev_state=state;
                 //update receiver output info
                             trellis[tran_matrx[j].ending_state][t+1].receiver_output =
                             tran_matrx[j].hypoth_channel_input;
         //update u vector
                             trellis[tran_matrx[j].ending_state][t+1].u[1]=
                                                   trellis[tran_matrx[j].begin_state][t].u[0];
           trellis[tran_matrx[j].ending_state][t+1].u[2]=
                                        trellis[tran_matrx[j].begin_state][t].u[1];
                              }
                         }
                         }
            }
           }
```

```
// RLS algorithm for updating channel coefficients
            for(int idx=0;idx<no_states;idx++){
//-------------------------------------------------------------
//for every state of t+1 find the previous state and Hypothetical channel input
                        st=trellis[idx][t+1].prev_state;
                trellis[st][t].u[0] = trellis[idx][t+1].u[1];
//-------------------------------------------------------------
//compute the conjugate vector of h-----------------------------
       for(int ind=0;ind<3;ind++){
                                conj_h[ind].real(trellis[st][t].h[ind].real());
          conj_h[ind].imag(-trellis[st][t].h[ind].imag());
                        }
//-------------------------------------------------------------
//compute output estimation again--------------------------------
                        y=vec_mult(trellis[st][t].u,conj_h);
//--------------------------------------------------------------
//compute error------------------------------------------------
                                err =symbol[si] - y;
          trellis[idx][t+1].error=err;
//--------------------------------------------------------------
//produce Kalman gain vector--------------------------------------
                        for(int g=0;g<3;g++){   //conjg of signal vector u
                                conj_u[g].real(trellis[st][t].u[g].real());
          conj_u[g].imag(-trellis[st][t].u[g].imag());
                        }
       vec_mult_matr(conj_u,trellis[st][t].P,temp1);//u_transpose*P
                        par1 = vec_mult(temp1,trellis[st][t].u);//u_transpose*P*u
                        matr_mult_vec(trellis[st][t].P,trellis[st][t].u,temp_Kal);
//P*u_transpose
                        for(g=0;g<3;g++){
                                temp_Kal[g]=temp_Kal[g]/(lambda+par1);
          trellis[st][t].kal[g]=temp_Kal[g];
                         }
//---------------------------------------------------------------
//update matrix P------------------------------------------------
       for(int ii=0;ii<3;ii++){
                                 for(int jj=0;jj<3;jj++){
                                         temp_P[ii][jj]=trellis[st][t].P[ii][jj];
                                 }
                         }
                        vecs_to_matr(temp_Kal,conj_u,temp3);//temp3=kal*u
                        matr_to_matr(temp3,temp_P,temp4);//temp4 = temp3*P
       matr_differ(temp_P,temp4);//P=P-temp4
                        const_matr(temp_P,lambda_rec);//P=P*lambda_rec->reciprocal of lambda
//----------------------------------------------------------------
//update tap coefficients------------------------------------------
       complex<double> conj_err(0.0,0.0);
                        conj_err.real(err.real());
                        conj_err.imag(-err.imag());
                        const_vec(temp_Kal,conj_err);//kal = Kal*conj_err
        for(int ifx=0;ifx<3;ifx++)
                                 temp_h[ifx]=trellis[st][t].h[ifx];
//!!!!!!!!!!!!TWO ENDING STATES CAN HAVE THE SAME PREVIOUS STATE
//!!!!!!!!!!!!FOR NO REASON WE ARE ALLOWED TO CHANGE INFO ABOUT
//!!!!!!!!!!!!THE TAPS :SO NO h=h+kal*err BUT temp_h = h + kal*err; BECAUSE
//WE WILL USE VECTOR h OF trellis[st][t] AGAIN as a previous state later same for P
                            add_vecs(temp_h,temp_Kal);//h=h+kal
//----------------------------------------------------------------
//update survivors ending state info-------------------------------
```

```
//update  ending state P matrix
                         for(int gx=0;gx<3;gx++){
                                     for(int gs=0;gs<3;gs++)
              trellis[idx][t+1].P[gx][gs]=temp_P[gx][gs];
                         }
//update  ending state tap coefficient vector
                                     for(int igs=0;igs<3;igs++)
              trellis[idx][t+1].h[igs]=temp_h[igs];

//-----------------------------------------------------------------
            }//END RLS
//read the next  symbol-------------------------------------------
   si++;
//-----------------------------------------------------------------
  }
//-----------------deallocate memory of arrays---------------------

  delete [] temp1;
  delete [] conj_u;
  delete [] temp_h;
  delete [] conj_h;
  delete [] temp_conj_h;
for(int im=0;im<3;im++)
  delete [] temp3[im];
    delete [] temp3 ;
for(im=0;im<3;im++)
  delete [] temp4[im];
    delete [] temp4 ;
for(im=0;im<3;im++)
  delete [] temp_P[im];
    delete [] temp_P ;
//----------------------------------------------------------------

//***********************TRACE BACK************************
//find the state that has the least accumulated cost at the end of the trellis
int st_ =0;
int trace_back_state=0;
double min_cost = trellis[st_][packet].cost; //packet is the last column of the array because arrays start
at 0

for(st_=1;st_<no_states;st_++){
 if( min_cost > trellis[st_][packet].cost ){
  min_cost=trellis[st_][packet].cost;
  trace_back_state=st_;
 }
}
//-------------------------------------------------------------------
//=============================output cost============================
/*{
            for(int i=0;i<no_states;i++){
                        out_cost << trellis[i][150].cost<<"   "<< trellis[i][151].cost <<endl;
            }
}*/
//initialize receiver output------------------------------------------

    for(int k=0;k<packet;k++)
     psp_out[k] =(0.0,0.0);
int st_index=trace_back_state; //starting position for backtrace
//-------------------------------------------------------------------
//fill the receiver output array--------------------------------------
```

```
 for(int m=(packet);m>=1;m--){
    psp_out[m-1]=trellis[st_index][m].receiver_output;
//====================print taps========================================
/*    out_error << trellis[st_index][m].error << "  ";
                        if(m == 150)
                                    out_error << "??????";
            h0real <<  trellis[st_index][m].h[0].real() <<"  ";
            if(m == 150)
                                    h0real << "??????";
            h0imag <<  trellis[st_index][m].h[0].imag() <<"  ";
   h1real <<  trellis[st_index][m].h[1].real() <<"  ";
   h1imag <<  trellis[st_index][m].h[1].imag() <<"  ";
   h2real <<  trellis[st_index][m].h[2].real() <<"  ";
   h2imag <<  trellis[st_index][m].h[2].imag() <<"  ";
*/
//=================================================================

            st_index=trellis[st_index][m].prev_state;
 }
//-------------------------------------------------------------------

//====================print symbols=====================================
/* for(int out_s=0;out_s<254;out_s++){
          out_symbols << psp_out[out_s] << "  ";


 }*/
//=================================================================

  return (complex<double>*) psp_out;
}//end of decode----------------------------------------------------



//-------------------------------------------------------------------
//the following constructor initializes the transition matrix and
//specifies the training sequence
//-------------------------------------------------------------------

channel_trellis_diagram:: channel_trellis_diagram(){
//define trellis diagram
            trellis = new channel_state *[no_states];
            for(int it=0;it<no_states;it++)
                        trellis[it] = new channel_state[packet+1];

//definition of transition matrix and ...
             tran_matrx = new channel_branch[max_no_transitions];
      train_seq=new complex<double> [26];
      psp_out = new complex<double> [packet];
//the starting state is defined by the last two symbols of the training sequence
complex<double> a(-1,-1);
complex<double> b(-1,1);
complex<double> c(1,-1);
complex<double> d(1,1);
//initialization of the channel transition matrix
//------------------------------------
tran_matrx[0].set_branch(0,0,a,a);
tran_matrx[1].set_branch(0,4,b,a);
```

```
tran_matrx[2].set_branch(0,8,c,a);
tran_matrx[3].set_branch(0,12,d,a);
//------------------------------------
tran_matrx[4].set_branch(1,0,a,b);
tran_matrx[5].set_branch(1,4,b,b);
tran_matrx[6].set_branch(1,8,c,b);
tran_matrx[7].set_branch(1,12,d,b);
//------------------------------------
tran_matrx[8].set_branch(2,0,a,c);
tran_matrx[9].set_branch(2,4,b,c);
tran_matrx[10].set_branch(2,8,c,c);
tran_matrx[11].set_branch(2,12,d,c);
//------------------------------------
tran_matrx[12].set_branch(3,0,a,d);
tran_matrx[13].set_branch(3,4,b,d);
tran_matrx[14].set_branch(3,8,c,d);
tran_matrx[15].set_branch(3,12,d,d);
//------------------------------------
tran_matrx[16].set_branch(4,1,a,a);
tran_matrx[17].set_branch(4,5,b,a);
tran_matrx[18].set_branch(4,9,c,a);
tran_matrx[19].set_branch(4,13,d,a);
//------------------------------------
tran_matrx[20].set_branch(5,1,a,b);
tran_matrx[21].set_branch(5,5,b,b);
tran_matrx[22].set_branch(5,9,c,b);
tran_matrx[23].set_branch(5,13,d,b);
//------------------------------------
tran_matrx[24].set_branch(6,1,a,c);
tran_matrx[25].set_branch(6,5,b,c);
tran_matrx[26].set_branch(6,9,c,c);
tran_matrx[27].set_branch(6,13,d,c);
//------------------------------------
tran_matrx[28].set_branch(7,1,a,d);
tran_matrx[29].set_branch(7,5,b,d);
tran_matrx[30].set_branch(7,9,c,d);
tran_matrx[31].set_branch(7,13,d,d);
//------------------------------------
tran_matrx[32].set_branch(8,2,a,a);
tran_matrx[33].set_branch(8,6,b,a);
tran_matrx[34].set_branch(8,10,c,a);
tran_matrx[35].set_branch(8,14,d,a);
//------------------------------------
tran_matrx[36].set_branch(9,2,a,b);
tran_matrx[37].set_branch(9,6,b,b);
tran_matrx[38].set_branch(9,10,c,b);
tran_matrx[39].set_branch(9,14,d,b);
//------------------------------------
tran_matrx[40].set_branch(10,2,a,c);
tran_matrx[41].set_branch(10,6,b,c);
tran_matrx[42].set_branch(10,10,c,c);
tran_matrx[43].set_branch(10,14,d,c);
//------------------------------------
tran_matrx[44].set_branch(11,2,a,d);
tran_matrx[45].set_branch(11,6,b,d);
tran_matrx[46].set_branch(11,10,c,d);
tran_matrx[47].set_branch(11,14,d,d);
//------------------------------------
tran_matrx[48].set_branch(12,3,a,a);
tran_matrx[49].set_branch(12,7,b,a);
```

```
tran_matrx[50].set_branch(12,11,c,a);
tran_matrx[51].set_branch(12,15,d,a);
//-------------------------------------
tran_matrx[52].set_branch(13,3,a,b);
tran_matrx[53].set_branch(13,7,b,b);
tran_matrx[54].set_branch(13,11,c,b);
tran_matrx[55].set_branch(13,15,d,b);
//-------------------------------------
tran_matrx[56].set_branch(14,3,a,c);
tran_matrx[57].set_branch(14,7,b,c);
tran_matrx[58].set_branch(14,11,c,c);
tran_matrx[59].set_branch(14,15,d,c);
//-------------------------------------
tran_matrx[60].set_branch(15,3,a,d);
tran_matrx[61].set_branch(15,7,b,d);
tran_matrx[62].set_branch(15,11,c,d);
tran_matrx[63].set_branch(15,15,d,d);

//creation of a training sequence--------------------------------
                    for(int i=0;i<26;i++){
                              train_seq[i].real(-1);
                              train_seq[i].imag(-1);
                    }
}

channel_trellis_diagram::~channel_trellis_diagram(){}





//the following class performs deinterleaving of the symbol sequence
//and also removes the training sequence

class DEInterleaver{

          complex<double> * input_signals;
          complex<double> ** deinterleaver;
          complex<double> * output_signals;

public:
          DEInterleaver();
   ~DEInterleaver(){}
          complex<double>* deinterleave(complex<double>* input_signals);
};

DEInterleaver::DEInterleaver(){
          deinterleaver =new complex<double> *[19];
          for(int i=0;i<19;i++)
                    deinterleaver[i]=new complex<double> [12];

          output_signals = new complex<double> [228];
}



complex<double>* DEInterleaver::deinterleave(complex<double>* input_signals){
//put signals back to deinterleaver array except for the 26 first symbols
          int s_idx = 26;//the first 26 symbols are the training sequence
```

```
	for(int j=0;j<12;j++){
		for(int i=0;i<19;i++){
			deinterleaver[i][j]=input_signals[s_idx];
			s_idx++;
		}
	}
//deinterleave signals
	int out_sy_idx = 0;
	for(int i=0;i<19;i++){
		for(int j=0;j<12;j++){
			output_signals[out_sy_idx]=deinterleaver[i][j];
			out_sy_idx++;
		}
	}
	return output_signals;
}
```

```
#define INFINITE 1.7e+308
class state{
public:
	double cost;	//accumulated cost
	int	prev_state; //the survivor previous state
	int	decoder_output; //the output of the survivor for traceback that
		// corresponts to the transition prev->current state

	state();
	~state(){};
int set_state(double Cost,int Prev_state,int Decoder_output);

};


state::state(){
	cost = INFINITE;
	prev_state= -1;
	decoder_output= -1;
}

int state::set_state(double Cost,int Prev_state,int Decoder_output){
	cost=Cost;
	prev_state=Prev_state;
	decoder_output=Decoder_output;
   return 1;
}

//class branch implements the branch of transition from a state to another

class branch {
public:
	int begin_state;
	int ending_state;
	complex<double> encoder_output; //the output of the encoder for that
				//transition after Qpsk modulation
	int encoder_input;	//the input that can cause that transition
```

```
    branch();
            ~branch(){};
int set_branch(int Begin_state,int Ending_state,complex<double> Encoder_output,
                                int Encoder_input);
};

branch::branch(){

            begin_state=-1;
            ending_state=-1;
            encoder_input=-1;

}

int branch::set_branch(int Begin_state,int Ending_state,complex<double> Encoder_output
                                                    ,int Encoder_input){

             begin_state = Begin_state;
             ending_state = Ending_state;
             encoder_output = Encoder_output;
             encoder_input = Encoder_input;
return 1;
}

class trellis_diagram{
            enum {no_states=4,L=3,max_no_transitions=8,
            no_possible_trans=2,packet = 228};
    state trellis[no_states][packet+1];
    branch tran_matrx[max_no_transitions];
            int start_state;       //the state from which the trellis begins
    int dec_out[packet];    //array that contains decoder output bit stream

public :

             trellis_diagram();
             ~trellis_diagram(){};
int*   decode(complex<double> * symbols);

};

trellis_diagram::trellis_diagram(){
       start_state = 0;

complex<double> outpa(-1,-1);
complex<double> outpb(-1,1);
complex<double> outpc(1,-1);
complex<double> outpd(1,1);

                    tran_matrx[0].set_branch(0,0,outpa,0);
                    tran_matrx[1].set_branch(0,2,outpd,1);
       tran_matrx[2].set_branch(1,0,outpd,0);
                    tran_matrx[3].set_branch(1,2,outpa,1);
       tran_matrx[4].set_branch(2,1,outpb,0);
                    tran_matrx[5].set_branch(2,3,outpc,1);
                    tran_matrx[6].set_branch(3,1,outpc,0);
                    tran_matrx[7].set_branch(3,3,outpb,1);

    for(int k=0;k<packet;k++)
     dec_out[k] = 0;
```

```
}

//The decode function performs decoding of the 228 convolutional code
//symbols(2 bits each).The algorithm finds the path that minimizes the accumulated
//metric for the whole sequence of symbols(it doesn't use the suboptimum
//approach of decoding a symbol after  proceding 5*L+1 stages in the
//trellis diagram)

int* trellis_diagram::decode(complex<double> * symbols){


//initialize trellis diagram for multiple packer transmition
            {
                        for(int i=0;i<no_states;i++){
                          for(int j=0;j<(packet+1);j++){
                                    trellis[i][j].cost=INFINITE;
                                    trellis[i][j].prev_state=-1;
                                    trellis[i][j].decoder_output=-1;
                          }
                        }
            }
//variable declaration
 int si =0;
 int active_states[no_states];
 int temp_active_states[no_states];
 double trans_cost = 0;
 double temp_diff  = 0;
 for(int i=0;i<no_states;i++){
            active_states[i]=0;
   temp_active_states[i]=0;
 }

 active_states[start_state]=1;  //at first only starting state is active
 trellis[start_state][0].cost=0;
 trellis[start_state][0].prev_state=-1;

//Parse the trellis columns
 for(int t=0;t<(packet);t++){
   //for every state in a column
   for(int state=0;state<no_states;state++){
    //if it is an active state( a state that is either the starting or an ending
            //state for a branch)--helps during transient part of the trelis.
    if(active_states[state]==1){
     //check transition matrix for the states it activates next instant
             for(int j=0;j<max_no_transitions;j++){
     if(state == tran_matrx[j].begin_state){
              //activate the states that will be active at next step
       temp_active_states[(tran_matrx[j].ending_state)]=1;
//calculate cost of transition eukleidian distance
       temp_diff = abs(symbols[si]-tran_matrx[j].encoder_output);
              trans_cost= pow(temp_diff,2);
//save under condition in trellis[next_state for 1 or 0 input][t+1] cost,previous state and decoder otput
              if(trellis[tran_matrx[j].ending_state][t+1].cost==INFINITE){
                         //no other branch has ended in that state before
         trellis[tran_matrx[j].ending_state][t+1].cost =
                        trellis[tran_matrx[j].begin_state][t].cost+trans_cost;
                        trellis[tran_matrx[j].ending_state][t+1].prev_state=state;
                        trellis[tran_matrx[j].ending_state][t+1].decoder_output =
                                    tran_matrx[j].encoder_input;}
       else{
```

```
      //other branches have reached this state and we have to find the survivor
                          double canditate_cost=trellis[tran_matrx[j].begin_state][t].cost+
                                        trans_cost;
              //if current branch produces a lower accumulated metric update state info
                          if(canditate_cost < trellis[tran_matrx[j].ending_state][t+1].cost){
                                      trellis[tran_matrx[j].ending_state][t+1].cost=canditate_cost;
                              trellis[tran_matrx[j].ending_state][t+1].prev_state=state;
                                      trellis[tran_matrx[j].ending_state][t+1].decoder_output =
                                                                  tran_matrx[j].encoder_input;

                          }
              }
            }
            }
   }
  }
 //read the next symbol
  si++;
 //update active states for next interval
 for(int l=0;l<no_states;l++)
   active_states[l]=temp_active_states[l];

 }

//**********TRACE BACK*****************
//find the state that has the least accumulated cost at the end of the trellis

int st =0;
int trace_back_state=0;
double min_cost = trellis[st][packet].cost; //packet is the last column of the array because arrays start at
0

for(st=1;st<no_states;st++){
 if( min_cost > trellis[st][packet].cost ){
  min_cost=trellis[st][packet].cost;
  trace_back_state=st;
 }
}

int st_index=trace_back_state; //starting position for backtrace

 for(int m=(packet);m>=1;m--){
    dec_out[m-1]=trellis[st_index][m].decoder_output;
    st_index=trellis[st_index][m].prev_state;
 }


 return (int*)dec_out;
}//end of decode
```

```
//Once in the program a call to srand function is needed  srand( (unsigned)time( NULL ) );
#define pi  3.1415926535;

//this a zero mean uniform random number
//from -0.5 to 0.5
```

```
double uniform(){
    return (double)(rand()&RAND_MAX)/(double)RAND_MAX-0.5;
}

//Next function returns a gaussian random number with mean=mean
//and variance = std_deviation^2.Uses the Box-Muller
//transformation of two uniform random numbers to gaussian
//random numbers

double gaussian(double mean,double std_deviation){
        static int ready =0;  //flag to indicated stored value
        static double gstore;  //place to store other value
        double v1,v2,r,fac,gaus;

        //make two numbers if none stored
        if(ready ==0){
                do {
                        v1=2.*uniform();
                        v2=2.*uniform();
                        r=v1*v1+v2*v2;
                }
                while (r>1.0);//make radius less than 1

                fac=sqrt((-2.*log(r))/r);
                gstore=v1*fac;
                gaus=v2*fac;
                ready=1;
        }
        else
        {
                ready=0;     //reset ready flag for next pair
                gaus=gstore; //return the stored one.The variable gaus
                  //is  a zero mean unit variance gaussian random variable
        }
        return std_deviation*gaus+mean;
        //Use of the relation Y=a*X+b produces a gaussian random
        //variable N(b,a*a)
}


int Packet_Bit_Errors(int* Source,int* Output,int packet){
        int errors = 0;

        for(int i=0;i<packet;i++){
                if(Source[i]!=Output[i])
                        errors++;
        }
return errors;
}




//the following function returns the product of a 1xM vector with a Mx1
//vector
complex<double> vec_mult(complex<double> *vec1,complex<double> *vec2){

          return(vec1[0]*vec2[0]+vec1[1]*vec2[1]+vec1[2]*vec2[2]);
```

```
}
//the following function returns the product of a Mx1 vector with a 1xM
//vector resulting in a MxM array

int  vecs_to_matr(complex<double> *vec1,complex<double> *vec2,
                                        complex<double> **result){
        for(int i=0;i<3;i++){
                for(int j=0;j<3;j++){
                        result[i][j] = vec1[i]*vec2[j];
                }
        }
        return EXIT_SUCCESS;
}
//the following returns the product of a MxM array with a MxM array
//resulting in a MxM array

int  matr_to_matr(complex<double> **M1,complex<double> **M2,
                                        complex<double> **result){

        for(int i=0;i<3;i++){
                for(int j=0;j<3;j++){
       result[i][j]=M1[i][0]*M2[0][j]+M1[i][1]*M2[1][j]+
                                       M1[i][2]*M2[2][j];
                }
        }
  return EXIT_SUCCESS;
}

//the following function returns the product of a NxN matrix to a Nx1
//vector and produces a Nx1 vector
int matr_mult_vec(complex<double>** matr,complex<double> *vec,
                                                complex<double> *result){
   result[0] = matr[0][0]*vec[0]+matr[0][1]*vec[1]+matr[0][2]*vec[2];
   result[1] = matr[1][0]*vec[0]+matr[1][1]*vec[1]+matr[1][2]*vec[2];
        result[2] = matr[2][0]*vec[0]+matr[2][1]*vec[1]+matr[2][2]*vec[2];

        return EXIT_SUCCESS;
}

//the following function returns the product of  a 1xN
//vector with a NxN matrix and produces a 1xN vector

int vec_mult_matr(complex<double> *vec,complex<double> **matr,
                                       complex<double> *result){

        result[0] = matr[0][0]*vec[0]+matr[1][0]*vec[1]+matr[2][0]*vec[2];
   result[1] = matr[0][1]*vec[0]+matr[1][1]*vec[1]+matr[2][1]*vec[2];
        result[2] = matr[0][2]*vec[0]+matr[1][2]*vec[1]+matr[2][2]*vec[2];

        return EXIT_SUCCESS;
}
//this function calculates the difference between two matrices
int matr_differ(complex<double> **M1,complex<double> **M2){

    for(int i=0;i<3;i++){
                for(int j=0;j<3;j++)
      M1[i][j]=M1[i][j]-M2[i][j];
         }
  return EXIT_SUCCESS;
}
```

```
//the following function calculare the multiplication of a matrix with a constant
int const_matr(complex<double> **M1,double par){
             for(int i=0;i<3;i++){
                        for(int j=0;j<3;j++)
          M1[i][j]=M1[i][j]*par;
             }
  return EXIT_SUCCESS;
}

//this function calculate the multiplication of a vector with a complex
//value
int const_vec(complex<double> *vec,complex<double> par){

            for(int i=0;i<3;i++)
            vec[i]=vec[i]*par;
            return EXIT_SUCCESS;
}

//this function calculate the addition of two vectors
int add_vecs(complex<double> *vec1,complex<double>* vec2){

            for(int i=0;i<3;i++)
            vec1[i]=vec1[i]+vec2[i];
            return EXIT_SUCCESS;
}


//the following function returns the product of a 1xM vector with a Mx1
//vector
complex<double> vec_mult(complex<double> *vec1,complex<double> *vec2){

              return(vec1[0]*vec2[0]+vec1[1]*vec2[1]+vec1[2]*vec2[2]);

}
//the following function returns the product of a Mx1 vector with a 1xM
//vector resulting in a MxM array

int  vecs_to_matr(complex<double> *vec1,complex<double> *vec2,
                                               complex<double> **result){
            for(int i=0;i<3;i++){
                        for(int j=0;j<3;j++){
                                    result[i][j] = vec1[i]*vec2[j];
                        }
            }
            return EXIT_SUCCESS;
}
//the following returns the product of a MxM array with a MxM array
//resulting in a MxM array

int  matr_to_matr(complex<double> **M1,complex<double> **M2,
                                               complex<double> **result){

            for(int i=0;i<3;i++){
                        for(int j=0;j<3;j++){
          result[i][j]=M1[i][0]*M2[0][j]+M1[i][1]*M2[1][j]+
                                             M1[i][2]*M2[2][j];
                        }
            }
  return EXIT_SUCCESS;
}
```

```
//the following function returns the product of a NxN matrix to a Nx1
//vector and produces a Nx1 vector
int matr_mult_vec(complex<double>** matr,complex<double> *vec,
                                        complex<double> *result){
   result[0] = matr[0][0]*vec[0]+matr[0][1]*vec[1]+matr[0][2]*vec[2];
   result[1] = matr[1][0]*vec[0]+matr[1][1]*vec[1]+matr[1][2]*vec[2];
        result[2] = matr[2][0]*vec[0]+matr[2][1]*vec[1]+matr[2][2]*vec[2];

        return EXIT_SUCCESS;
}

//the following function returns the product of  a 1xN
//vector with a NxN matrix and produces a 1xN vector

int vec_mult_matr(complex<double> *vec,complex<double> **matr,
                              complex<double> *result){

        result[0] = matr[0][0]*vec[0]+matr[1][0]*vec[1]+matr[2][0]*vec[2];
   result[1] = matr[0][1]*vec[0]+matr[1][1]*vec[1]+matr[2][1]*vec[2];
        result[2] = matr[0][2]*vec[0]+matr[1][2]*vec[1]+matr[2][2]*vec[2];

        return EXIT_SUCCESS;
}
//this function calculates the difference between two matrices
int matr_differ(complex<double> **M1,complex<double> **M2){

    for(int i=0;i<3;i++){
              for(int j=0;j<3;j++)
        M1[i][j]=M1[i][j]-M2[i][j];
          }
  return EXIT_SUCCESS;
}
//the following function calculare the multiplication of a matrix with a constant
int const_matr(complex<double> **M1,double par){
         for(int i=0;i<3;i++){
              for(int j=0;j<3;j++)
        M1[i][j]=M1[i][j]*par;
          }
  return EXIT_SUCCESS;
}

//this function calculate the multiplication of a vector with a complex
//value
int const_vec(complex<double> *vec,complex<double> par){

        for(int i=0;i<3;i++)
        vec[i]=vec[i]*par;
        return EXIT_SUCCESS;
}

//this function calculate the addition of two vectors
int add_vecs(complex<double> *vec1,complex<double>* vec2){

        for(int i=0;i<3;i++)
        vec1[i]=vec1[i]+vec2[i];
        return EXIT_SUCCESS;
}
```

ΠΑΡΑΡΤΗΜΑ Β

ΚΩΔΙΚΕΣ ΠΡΟΣΟΜΟΙΩΣΕΩΝ ΣΕ MATLAB

Στο παράρτημα αυτό παρουσιάζονται και σχολιάζονται οι κώδικες των προγραμμάτων που υλοποιήθηκαν σε MATLAB και αφορούν την προσομοίωση των δύο τύπων –fast και extreme fading- του Rayleigh fading channel.

```
function [ri,env,angle] = lowfilter(n,Wn,samples,tap_no)

[b,a]=butter(n,Wn);

w1 = normrnd(0,2,1,samples);
w2 = normrnd(0,2,1,samples);

ri = filter(b,a,w1);
rq = filter(b,a,w2);
ri = ri(100:samples);
rq = rq(100:samples);

env = sqrt(ri.^2+rq.^2);
angle = 2*atan(ri./rq);

if tap_no==0
  fid0 = fopen('c:\george\diplomatikh\conv_Viterby\zh0real.txt','w');
   fid1 = fopen('c:\george\diplomatikh\conv_Viterby\zh0imag.txt','w');
   fprintf(fid0,'g \n',ri);
   fprintf(fid1,'g \n',rq);
   fclose(fid0);
   fclose(fid1);
elseif tap_no == 1
   fid0 = fopen('c:\george\diplomatikh\conv_Viterby\zh1real.txt','w');
   fid1 = fopen('c:\george\diplomatikh\conv_Viterby\zh1imag.txt','w');
   fprintf(fid0,'g \n',ri);
   fprintf(fid1,'g \n',rq);
   fclose(fid0);
   fclose(fid1);
else
   fid0 = fopen('c:\george\diplomatikh\conv_Viterby\zh2real.txt','w');
   fid1 = fopen('c:\george\diplomatikh\conv_Viterby\zh2imag.txt','w');
   fprintf(fid0,'g \n',ri);
   fprintf(fid1,'g \n',rq);
   fclose(fid0);
   fclose(fid1);
end
```

```
%symbol period
T = 3.69e-6;
%simulation step size in symbol periods
Tstep =1000*T;
%doppler frequency spread
fd = 0.001/T;
%---------------------------------------------------------------------
%calculate the theoretical autocorrelation between the low-pass filtered white noise
%(isotropic scattering
%number of simulation steps
M = 500;
%plot steps
m = 400;
%divide each step to smaller for accuracy in plotting the Bessel function
n = 0:0.2:M;
%calculate the Bessel function of fist kind zero order
j = besselj(0,2*pi*fd*n*Tstep);
plot(n(1:(m/0.2)),j(1:(m/0.2)),'r');
xlabel('number of steps(step = 200Ts)');
title('Theoretical autocorrelation of fr');
figure;
%---------------------------------------------------------
%filter order
n=10;
%cutoff frequency (1 stands for half the Nyquist frequency)
Wn=1/8;
%no of filter taps to be generated
samples = 4000;
%low-pass filter the gaussian random variables
[ri0,env0,angle0] = lowfilter(n,Wn,samples,0);
[ri1,env1,angle1] = lowfilter(n,Wn,samples,1);
[ri2,env2,angle2] = lowfilter(n,Wn,samples,2);
%calculate the autocorelation of the filtered white gaussian random variables
[x,lags] = xcorr(ri0,M,'coeff');
%plot(lags(M+1:2*M+1),x(M+1:2*M+1)) %the positive part of lags is the last half of lag
vector
plot(lags(M+1:M+1+m),x(M+1:M+1+m))
xlabel('number of steps(step = 1000T)');
title('normalized autocorrelation of fr');
```

```
%symbol period
T = 3.69e-6;
%simulation step size in symbol periods
Tstep =200*T;
%doppler frequency spread
fd = 0.005/T;
%---------------------------------------------------------------------
%calculate the theoretical autocorrelation between the low-pass filtered white noise
%(isotropic scattering
%number of simulation steps
```

```
M = 500;
%plot steps
m = 400;
%divide each step to smaller for accuracy in plotting the Bessel function
n = 0:0.2:M;
%calculate the Bessel function of fist kind zero order
j = besselj(0,2*pi*fd*n*Tstep);
plot(n(1:(m/0.2)),j(1:(m/0.2)),'k');
xlabel('number of steps(step = 200Ts)');
title('Theoretical autocorrelation of fr');
figure;
%--------------------------------------------------------
%filter order
n=10;
%cutoff frequency (1 stands for half the Nyquist frequency
Wn=1/8;
%no of filter taps to be generated
samples = 12000;
%low-pass filter the gaussian random variables
[ri0,env0,angle0] = lowfilter(n,Wn,samples,0);
[ri1,env1,angle1] = lowfilter(n,Wn,samples,1);
[ri2,env2,angle2] = lowfilter(n,Wn,samples,2);
%calculate the autocorelation of the filtered white gaussian random variables
[x,lags] = xcorr(ri0,M,'coeff');
%plot(lags(M+1:2*M+1),x(M+1:2*M+1)) %the positive part of lags is the last half of lag
vector
plot(lags(M+1:M+1+m),x(M+1:M+1+m),'k')
xlabel('number of steps(step = 200Ts)');
title('normalized autocorrelation of fr');


EbNo = [10 12 14 16 20];
BER_mlse = [0.0268913 0.0105531 0.00485819 0.00240887 0.000381306];
semilogy(EbNo,BER_mlse,'-.k');
hold on;
Ber_ssa = [0.0178514 0.00691131 0.00323635 0.00118372 0.000224156];
semilogy(EbNo,Ber_ssa,'k');
grid;
axis([8 20 1e-4 1e0]);
title('BER plot for PSP-MLSE and PSP-SSA(extreme dynamics)');
ylabel('BER');
xlabel('Eb/No(dB)');
hold off;
gtext('PSP-MLSE');
gtext('PSP-SSA');


EbNo = [6 8 10 12];
BER_mlse = [0.0234123 0.0117943 0.00548441 0.00273099];
semilogy(EbNo,BER_mlse,'-.k');
hold on;
Ber_ssa = [0.0153244 0.00578771 0.00264473 0.000692976];
semilogy(EbNo,Ber_ssa,'k');
```

```
grid;
axis([5 13 1e-4 1e0]);
title('BER plot for PSP-MLSE and PSP-SSA (fast fading)');
ylabel('BER');
xlabel('Eb/No(dB)');
hold off;
gtext('PSP-MLSE');
gtext('PSP-SSA');
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] G. Forney, "MLSE of Digital Sequences in the Presence of Intersymbol Interferecne", IEEE Trans. Inform. Theory, IT-18, pp.363-378, May 1972.

[2] J. Proakis, Digital Communications, 3rd Ed., McGraw-Hill, New York, 1995.

[3] R.Blahut, Digital Transmission of Information, 1st Ed, Addison-Wesley, New York, 1990.

[4] R.Price, "Optimum Detection of Random Signals in Noise, with Aplication to Scatter-Multipath Communication, I", IRE Trans. Inform. Theory, IT-2, pp.125-135, Dec. 1956.

[5] P.Embree and B.Kimble, C Language Algorithms for Digital Signal Processing, 1st Ed, Prentice-Hall, New York,1991.

[6] T.Kailath, "Optimum Receivers for Randomly Varying Channels," Proc. 4th Symposium Inform. Theory, Butterworth Scientific Press, London, pp. 109-122, 1961.

[7] T.Kailath, "A General Likelihood-Ratio Formula for Random Signals in Gaussian Noise," IEEE Trans. Inform. Theory, IT-15, pp.350-361, May 1969.

[8] R. Steele, Mobile Radio Communications, IEEE Press, New Jersey, USA, 1st Ed., 1994.

[9] A Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," IEEE Trans. Inform.Theory, IT-13, pp.259-260,Apr. 1967.

[10]G. Forney, "The Viterbi Algorithm," Proc. of IEEE, vol. 61, pp.268-278, March 1973.

[11] Y. Li, B. Vucetic, Y. Sato:"Optimum soft-output detection for channels with intersymbol interference".IEEE Trans. Inform. Theory, IT-41, pp.704-713, May 1995.

[12] G.Ungerboeck, "Adaptive ML Receiver for Carrier-Modulated Data-Transmission Systems," IEEE Trans. Commun.,COM-22, pp.624-636, May 1974.

[13] A. Polydoros and D. Kazakos, "MLSE in the Presence of Infinite ISI," Proc. ICC, Boston, MA, pp.25.2.1-25.2.5, June 1989

[14] A. Duel-Hallen and C.Heegard, "Delayed Decision Feedback Estimation," IEEE Trans. Commun.,COM-37,pp.428-436, Jan. 1988.

[15] A. Polydoros and R. Raheli, “The Principle of PSP : A General Approach to Approximate and Adaptive MLSE,” Communication Sciences Institute, University of Southern California, Report no. CSI-90-07-05, July 1990.

[16] M. Ghosh, “An Optimal Approach to Blind Equalization,” Ph.D. Thesis, USC, Dec. 1991.

[17] R.E. Morley, Jr. and D.L. Snyder, “Maximum Likelihood Sequence Estimation for Randomly Dispersive Channels,” IEEE Trans. Commun., vol. COM-27,No. 6, pp.883-839, June 1979.

[18] C-K. Tzou, R.Raheli and A. Polydoros, “Applications of PSP to mobile Digital Communications,” Proc. Globecom, Dec.1993.

[19] A.D’Andrea, U. Mengali and G. Vitetta, “Aproximate ML Decoding of Coded PSK with No Explicit Carrier Phase Reference,” IEEE Trans. Commun.,COM-42, pp.1033-1040, No. 2/3/4, 1994.

[20] H.Kubo, K. Murakami and T.Fujino, “An Adaptive MLSE for Fast Time Varying ISI Channels,” IEEE Trans. Commun., COM-42, pp.1872-1880, No. 2/3/4, 1994.

[21] R.Raheli, A. Polydoros and C-K. Tzou, “PSP : A General Aproach to MLSE in Uncertain Enviroments,” IEEE Trans. Commun., COM-43, pp.354-364, No. 2/3/4 1995.

[22] R. Raheli, G. Marino and P.Castoldi, “PSP and Tentative Decisions: What is in Between?,” IEEE Trans. Commun., COM-44, pp.127-129, Feb.1996.

[23] K.M Chugg and A. Polydoros, “MLSE for an Unknown Channel-Part I: Optimality Considerations,” IEEE Trans. Commun., COM-44, pp.836-846, Jul.1996.

[24] K.M Chugg and A. Polydoros, “MLSE for an Unknown Channel-Part II: Tracking Performance,” IEEE Trans. Commun., vol. 44, pp.949-958, Aug.1996.

[25] A. Polydoros and N. Lay, “Channel estimation and Blind Equalization Using Minimum State PSP,” Proc.MILCOM’95,pp. 993-997,Nov.1995.

[26] A. Anastasopoulos and A. Polydoros, “Soft Decisions Per-Survivor Processing for Mobile Fading Channels,”Proc. VTC’97,Phoenix, AZ, May 4-7, 1997 .

[27] A. Anastasopoulos and A. Polydoros, “Adaptive Soft Decision Algoritms for Mobile Fading Channels,” submitted to European Trans. Telecomm., April 1997.

[28] C.-K. Tzou, “Per-Survivor Processing : A general Approach to MLSE in Uncertain Environments,” Ph.D. Dissertation, University of Southern California, December 1993.

[29] G. Paparisto and K.M. Chung, "PSP Array Processing for Multipath Fading Channels," submitted to IEEE Trans.Commun., June 1997

[30] M. Eyuboglou and S. Qureshi, "Reduced-State Sequence Estimation with Set Partioning and Decision Feedback," IEEE Trans. Commun.,COM-36, pp.13-20, July 1988.

[31] P. Chevillat and E. Elefteriou, "Decoding of Trellis-Encoded Signals in the Presence of ISI and Noise," IEEE Trans. Commun.,COM-36, pp.669-676, July 1989.

[32] N. Seshadri, "Joint Data and Channel Equalization Using Fast Blind Trellis Search Techniques," Proc. Globecom, pp.1659-1663, Dec. 1990

[33] R. Iltis, "A Bayesian MLSE Algorithm for a priori Unknown Channels and Symbol Timing," IEEE Journ. Select. Areas Commun.,JSAC-10,pp.579-588,April 1992.

[34]J.Lodge and M. Moher, "ML Estimation of CPM Signals Transmitted over Rayleigh Flat Fading Channels," IEEE Trans. Commun.,COM-38,pp.787-794, June 1990.

[35] X.Yu and S. Pasupathy, "Innovations-Based MLSE for Rayleigh Fading Channels," IEEE Trans. Commun.,COM-43, pp.1534-1544, No. 2/3/4 1995.

[36] A. Reichman and R. Scholtz, " Joint Phase Estimation and Data Decoding for TCM Systems," Proc. FISCTA, Scottland, Sept. 1991.

[37] J. Lin, F. Ling and J. Proakis, "Joint Data and Channel Estimation for TDMA Mobile Channels," Proc. PIMRC92, pp.235-239, Oct.1992.

[38] Z.Xie, C.Rushforth, R. Short and T. Moon, "Joint Signal Detection and Parameter Estimation in Multiuser Communications," IEEE Trans. Commun., COM-41, pp.1208-1216, Aug. 1993.

[39] A. D'Andrea, U. Mengali and G. Vitetta, "Multiple Phase Synchronization in Continuous Phase Modulation," Digital Signal Processing: A Review Journal, A. Polydoros, Ed., Vol. 3, pp.188-198, July 1993.

[40] K.M. Chugg and A. Polydoros, "On the Existence and Uniqueness of Joint Channel and Data Estimates," IEEE IT Symposium, Sept. 1995.

[41] Godon L. Stuber, "Principles of Mobile Communications", Kluwer Academic Publishers, Boston, USA, 1996.