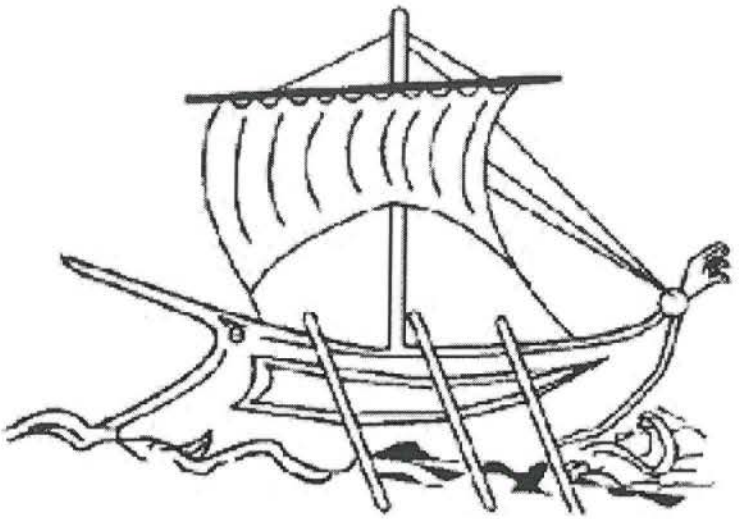


Τ.Ε.Ι. ΠΕΙΡΑΙΑ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΑΥΤΟΜΑΤΙΣΜΟΥ

«ΑΠΟΦΥΓΗ ΣΥΓΚΡΟΥΣΕΩΝ UAV ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ ΤΟΝ ΑΛΓΟΡΙΘΜΟ Α*»

ΚΑΛΟΓΕΡΑ ΕΥΤΥΧΙΑ , ΓΙΑΝΝΝΟΥΛΑΣ ΧΡΗΣΤΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΕΠΙΒΛΕΠΩΝ

ΤΣΕΛΕΣ ΔΗΜΗΤΡΙΟΣ

ΕΥΧΑΡΙΣΤΙΕΣ

Η πτυχιακή εργασία μας εκπονήθηκε στο τμήμα Αυτοματισμού στο Τ.Ε.Ι Πειραιά. Την βιώσαμε σαν μία κατακλείδα των σπουδών μας, κάτι σαν την κατάληξη ενός εποικοδομητικού ταξιδιού με ανατροπές.

Θα θέλαμε να ευχαριστήσουμε θερμά τον καθηγητή κ. Δημήτριο Τσελέ που μας έδωσε την ευκαιρία να ασχοληθούμε με το θέμα των αλγορίθμων και να μάθουμε ένα σωρό ενδιαφέροντα πράγματα μέσα από την ενασχόλησή μας μ' αυτούς. Η συνεχής καθοδήγηση, η αμέριστη υποστήριξη, οι ουσιώδεις συμβουλές, καθώς και η αδιάκοπη συμπαράσταση και ενθάρρυνση που μας παρείχε σε όλο αυτό το διάστημα υπήρξαν για μας πολύ σημαντικά και εποικοδομητικά.

Επίσης, θα θέλαμε να ευχαριστήσουμε τον κ. Χρήστο Δρόσο για την εποπτεία και τις χρήσιμες παρατηρήσεις και συμβουλές του. Ήταν πάντα διαθέσιμος να μας προσφέρει τις γνώσεις και την εμπειρία του.

Ένα ιδιαίτερο, μεγάλο «ευχαριστώ» στο φίλο και συνάδελφο Γιάννη Τρίχα, ο οποίος δε μας προσέφερε απλά πολύτιμες γνώσεις πάνω στο αντικείμενο του μηχανικού αυτοματιστή, αλλά μας βοήθησε να αναπτύξουμε ικανότητες που δεν πιστεύαμε πως έχουμε.

Τέλος, θέλουμε να ευχαριστήσουμε όλους εκείνους που μας έμαθαν να «προσπερνάμε» και μας βοήθησαν να γίνουν «ανεκτοί» οι συμβιβασμοί των τελευταίων χρόνων: τις οικογένειές μας, τους φίλους μας, τους συναδέλφους μας. Σε αυτούς, που με την καθημερινή τους συμπαράσταση, την υπομονή τους και την θετική τους σκέψη συνέβαλαν στην εκπλήρωση των στόχου μας, αφιερώνεται η εργασία αυτή.

Καλογερά Ευτυχία , Γιαννούλας Χρήστος

Ιανουάριος 2014

ΠΕΡΙΛΗΨΗ

Η αποφυγή της σύγκρουσης είναι η ουσιώδης προυπόθεση για τα μη επανδρώμενα αέρια οχήματα ώστε να καταστούν πλήρως αυτόνομα. Πολλοί αλγόριθμοι έχουν προταθεί για να αποτελέσουν τη βάση σχεδιασμού σε ένα προσομοιωμένο περιβάλλον, αλλά μόνο λίγοι μπορούν να επιβιώσουν αποτελεσματικά σε ένα περιβάλλον προσομοίωσης. Το θέμα εμποδίζει την χρήση UAVs σε εμπορικές και άλλες εφαρμογές διότι τα UAVs πετούν αυτόνομα , η αδυναμία τους να αποφύγουν άλλα αεροσκάφη στον αέρα, μπορούν να προκαλέσουν σοβαρούς κινδύνους.

Στην παρούσα εργασία εξετάζουμε πολλές προσεγγίσεις συμπεριλαμβανομένου του αλγορίθμου A* , τη συνολική προσέγγιση στον τομέα ανίχνευσης και τη διαδικασία Markov. Τότε, προτείνεται η τροποποίηση του αλγορίθμου A*. Τυπικά, ο αλγόριθμος A* υλοποιείται σε ένα σύστημα κινητών ρομπότ για το σχεδιασμό της βάσης σε ένα στατικό περιβάλλον. Εισάγουμε μερικές προσεγγίσεις που μας επιτρέπουν να χρησιμοποιήσουμε τον αλγόριθμο A* σε ένα δυναμικό περιβάλλον. Η εκτίμηση αυτού του αλγορίθμου βασίζεται στην εξομοίωση διαφορετικών σεναρίων, και η σύγκριση μεταξύ δυο ευρετικών συναρτήσεων θα αναλυθούν. Συζητάμε την εφαρμογή της προσέγγισής μας, τις κατάλληλες συνθήκες, την αξιόπιστη λειτουργία της και τι θέματα θα μπορούσαν να επηρεάσουν αυτή την εφαρμογή. Επίσης, ερευνούμε τους περιορισμούς της προσέγγισής μας στο ακραίο σενάριο να παρέχουμε χρήσιμες προτάσεις βελτίωσης.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Τα μη επανδρωμένα εναέρια οχήματα είναι αεροσκάφη χωρίς πιλότο μέσα σε αυτά. Είτε ελέγχονται από πιλότους ή προγραμματίζονται εκ των προτέρων να πετάνε αυτόνομα. Το UAV θα μπορούσε να είναι ένα αεροσκάφος σταθερών πτερυγίων, ένα ελικόπτερο, ένα ρομποτικό πουλί ή οποιοδήποτε όχημα κινείται στον αέρα. Αρχικά, τα UAVs σχεδιάστηκαν και χρησιμοποιήθηκαν για στρατιωτικούς σκοπούς, όπως για παράδειγμα η αναγνώριση και οι αποστολές επίθεσης. Σε πρόσφατες συγκρούσεις, τα UAVs πραγματοποίησαν πολυάριθμες αποστολές κατασκοπίας και μάχης. Οι αυξανόμενες δυνατότητες και επιτυχίες τους, κινούν όλο και περισσότερο το ενδιαφέρον για ενδεχόμενες εμπορικές εφαρμογές στη γεωργία, τη χωμετρία και τον έλεγχο αγωγών πετρελαίου μεταξύ άλλων.

Ωστόσο, δεν υπάρχει λόγος που τα UAVs αποτρέπονται από τη διαδεδόμενη εμπορική ανάπτυξη. Η Ομοσπονδιακή Διοίκηση Αεροπορίας (ΟΔΑ) έχει αυστηρές απαιτήσεις για την ενσωμάτωση των UAVs στον εναέριο χώρο, θα πρέπει να είναι τόσο επαρκή (ικανά να λειτουργούν με ασφάλεια) όσο ένας ισάξιος πιλότος χωρίς συνεργασία (όπως για παράδειγμα, εντολές από ελεγκτή ή πληροφορίες από γειτονικό αεροσκάφος). Όταν τα UAVs πετάξουν αυτόνομα, η αδυναμία τους να αποφύγουν άλλο αεροσκάφος, συνιστά κύριο μέλημα και τα αποκλείει από εμπορικές εφαρμογές. Τα UAVs πρέπει να αποφύγουν τις συγκρούσεις.

Υπάρχουν πολλές υπάρχουσες προσεγγίσεις για την επίλυση του προβλήματος που αφορά στην αποφυγή της σύγκρουσης ενός UAV. Για παράδειγμα, ο σταθμός εφαρμόζει την πλησιέστερη μέθοδο προσέγγισης για την επίτευξη της ανίχνευσης της σύγκρουσης ενός UAV και την επίλυση του προβλήματος της σύγκρουσης. Μερικές ερευνητικές ομάδες χρησιμοποίησαν προσεγγίσεις με πλέγμα για να επιτύχνουν την αποφυγή της σύγκρουσης. Ο Tooren εφάρμοσε τον αλγόριθμο A* για να αποφύγει τα εμπόδια. Ο αλγόριθμος A* (λεγόμενος και A star) είναι μια προσέγγιση με πλέγμα και σχεδιάστηκε αρχικά για επίγεια ρομπότ που μπορούν να επιταχύνουν, να επιβραδύνουν, να σταματούν, να στρίβουν αυτόνομα, ή ακόμα και να δημιουργήσουν αντίγραφα ασφαλείας κατά μήκος των σταθερών εμποδίων. Σ'αυτή τη μελέτη θα προσαρμόσουμε τον αλγόριθμο A* για το σχεδιασμό βάσης για τα UAVs ώστε να μπορούν να αποφύγουν άλλα UAVs.

Σ'αυτή την εργασία, υποθέτουμε ότι τα UAVs πετούν σε σταθερή ταχύτητα, δεν αλλάζουν υψόμετρο και γυρίζουν 22° το δευτερόλεπτο το πολύ. Η σταθερή ταχύτητα και το υψόμετρο βελτιστοποιούν την απόδοση ενέργειας. Αν και ο αλγόριθμος A* συχνά χρησιμοποιείται για την επίλυση του σχεδίου βάσης των UAVs, μόνο λίγοι ερευνητές τον εφαρμόζουν για να επιτύχουν την αποφυγή της σύγκρουσης. Αυτές οι εργασίες που χρησιμοποιούν τον αλγόριθμο A* για την αποφυγή της σύγκρουσης, δεν συζητούν πολύ την σχέση ανάμεσα στις ευρετικές και περιβαλλοντικές παραμέτρους (μέγεθος πεδίου, αριθμός των UAVs κλπ). Ερώτηση: Πώς η απόδοση διαφέρει με το χειρισμό του αλγόριθμου A * και των περιβαλλοντικών συνθηκών στη σύγκρουση UAVs πρόβλημα αποφυγής;

Σ'αυτή την εργασία, θέλουμε να απαντήσουμε σε αυτό το ερώτημα εκτελώντας προσομοιώσεις με πολλά σενάρια. Όλες οι πληροφορίες προσομοιώνονται σύμφωνα με τη C++ και το Matlab περιβάλλον πρτογραμματισμού. Συνεπώς, μπορούμε να προσδιορίσουμε τους περιορισμούς για τον αλγόριθμο A* χρησιμοποιώντας διαφορετικές ευρετικές λειτουργίες. Για παράδειγμα, θα δείξουμε πόσο πρέπει να αποκλίνουν τα UAVs ώστε να κρατηθούν στην ασφαλή διαδρομή και πόσα UAVs σε ορισμένο πεδίο θα προκαλέσουν αναπόφευκτη σύγκρουση.

Σε αυτή τη μελέτη, πρώτα θα εξετάσουμε τις συναφείς εργασίες με διαφορετικές προσεγγίσεις. Το μοντέλο μας τότε θα κλιθεί να αποδείξει πώς λύνουμε το πρόβλημα αποφυγής της σύγκρουσης. Έπειτα, θα εκτελέσουμε πολλές προσομοιώσεις σε διαφορετικά σενάρια για να αξιολογήσουμε την προσέγγισή μας. Τέλος, προτείνουμε το συμπέρασμά μας και την πρότασή μας για μελλοντική εργασία.

ΚΕΦΑΛΑΙΟ 2

ΑΝΑΣΚΟΠΗΣΗ ΤΗΣ ΛΟΓΟΤΕΧΝΙΑΣ

Έχουν προταθεί πολλές προσεγγίσεις με σκοπό την αποφυγή συγκρούσεων για τα UAVs. Θα εξετάσουμε τη γεωμετρική προσέγγιση, την στοχαστική, αυτή του γραμμικού προγραμματισμού, τα τεχνητά δυναμικά πεδία (ΤΔΠ) και τις προσεγγίσεις με πλέγμα.

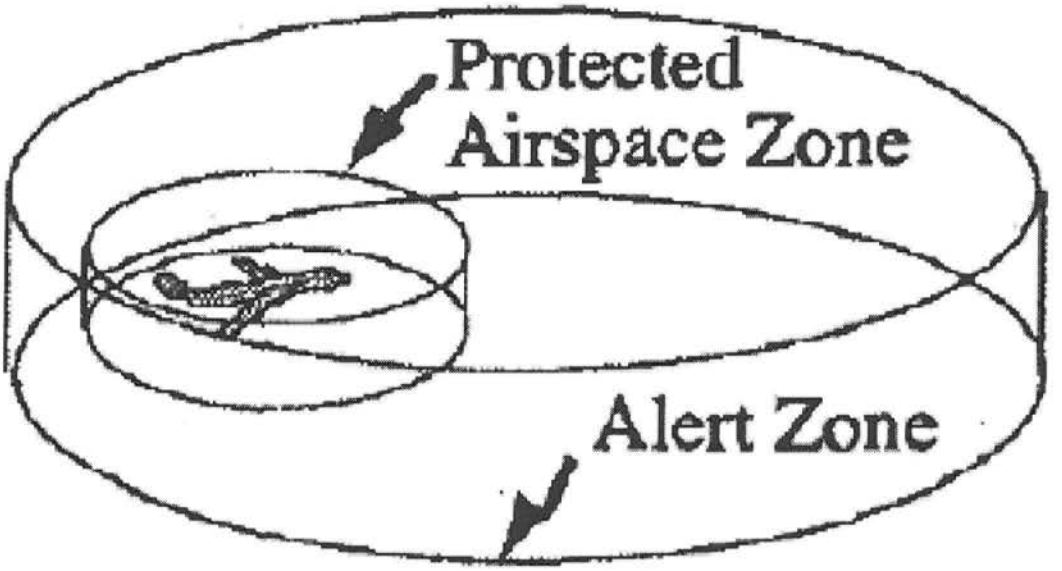
2.1 Γεωμετρική Προσέγγιση

Σ'αυτή την προσέγγιση, κάθε UAV θεωρείται σημειακή μάζα, η μελλοντική του διαδρομή μπορεί να προβλευθεί βασιζόμενη στην πιο πρόσφατη τροχιά του, ταχύτητα και συμπεριφορά του. Ένας αντιπροσωπευτικός αλγόριθμος είναι η προσέγγιση κοντινότερου σημείου. Δεδομένου τις προβλεπόμενες διαδρομές όλων των UAVs, η μέθοδος PCA υπολογίζει τις αποστάσεις μεταξύ των UAVs. Εάν η απόσταση είναι μικρότερη από κάποια απόσταση ασφαλείας, εκτελούνται στρατηγικές αποφυγής για να αποφευχθούν οι συγκρούσεις.

2.1.1 Ελεύθερη Πτήση

Το 1991, ο Διεθνής Οργανισμός Πολιτικής Αεροπορίας δημιούργησε το σύστημα μελλοντικής αεροναυτιγίας. Η γενική ιδέα της «καθορισμένης τροχιάς από το χρήστη » προτάθηκε και έγινε γνωστή ως ελεύθερη πτήση από τα μέσα της δεκαετίας του 1990. Η γενική ιδέα της ελεύθερης πτήσης είναι να αντικαταστήσει τις τρέχουσες μεθόδους διαχείρησης εναέριας κυκλοφορίας με τη χρήση της αναπτυσσόμενης τεχνολογίας της εποχής μας. Ο απώτερος στόχος της ελεύθερης πτήσης δεν είναι η χρήση κεντρικού ελέγχου (όπως για παράδειγμα κεντρικού σταθμού), αλλά η αυτόματη πτήση με κατανεμημένο τρόπο χρησιμοποιώντας την επικοινωνία μέσω υπολογιστή για να διασφαλίσει ότι η ζώνη συναγερμού των εναέριων πτήσεων δεν θα παραβιαστεί.

Μόλις παραβιασθεί αυτή η ζώνη, οι ελεγκτές εναέριας κυκλοφορίας θα παρέμβουν για να συμβάλλουν στην αποφυγή της σύγκρουσης να προλάβουν την παραβίαση της ζώνης του προστατευόμενου εναέριου χώρου. Η εικόνα 2.1 απεικονίζει την ζώνη συναγερμού και την ζώνη προστατευόμενου εναέριου χώρου. Αυτή η ιδέα κινητοποίησε πολλές ερευνητικές ομάδες να αναπτύξουν την αυτόνομη ανίχνευση σύγκρουσης και μεθόδουν ανάλυσης. Το 1997, ο Krozel και ο Peters πρότειναν ένα οικονομικό μοντέλο για την ανίχνευση και ανάλυση της σύγκρουσης χρησιμοποιώντας την μέθοδο PCA.



Εικόνα 2.1

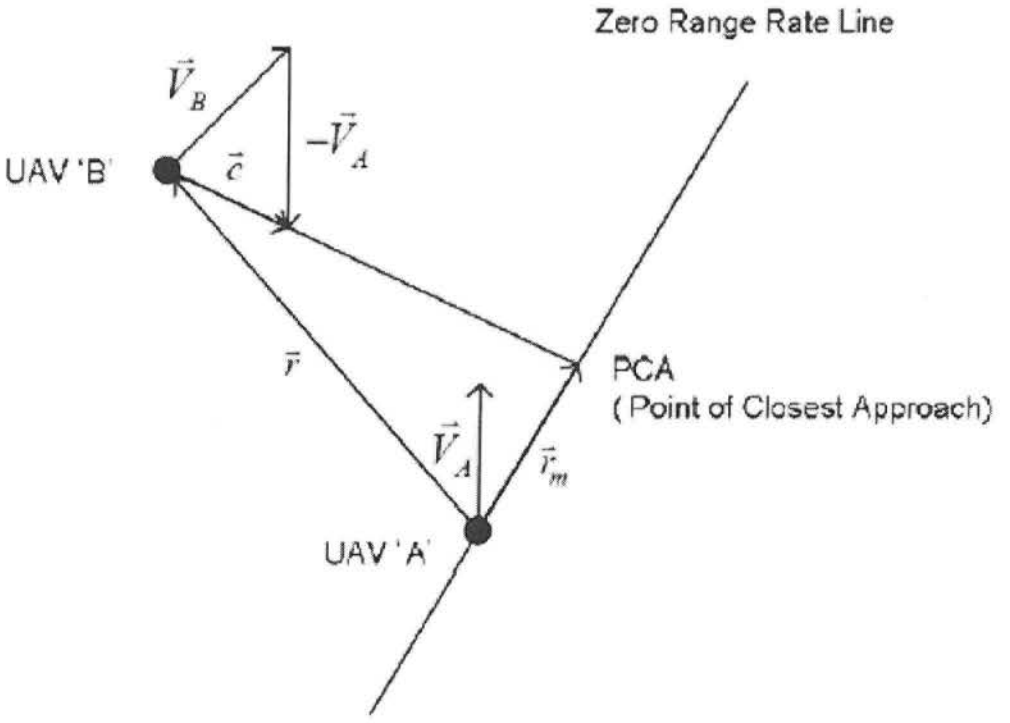
2.1.2 Σημείο Πλησιέστερης Προσέγγισης

Ο Krozel και ο Peters σκέφτηκαν δυο UAVs να πετάνε χωρίς πλαγιολίσθηση, τα οποία θα συναντηθούν μεταξύ τους όπως φαίνεται στην εικόνα 2.2. Προκειμένουν να προβλευθεί εάν μια σύγκρουση θα συμβεί ή όχι, χρησιμοποιούν τη γεωμετρική μέθοδο για να υπολογίσουν την μικρότερη μεταξύ αυτών των 2 UAVs απόσταση. Με άλλα λόγια, χρησιμοποιούν την μέθοδο PCA για να πιάσουν την χαμένη απόσταση.

Το διάνυσμα αυτής της απόστασης προσδιορίζεται: $\vec{r}_m = \hat{c} \times (\vec{r} \times \hat{c})$

Όπου $\vec{r}$ το διάνυσμα σχετικής απόστασης του UAV "B" σε σχέση με το UAV "A"

και $\hat{c}$ η μονάδα διανύσματος στην κατεύθυνση του διανύσματος σχετικής ταχύτητας.



Εικόνα 2.2

Το 1998, ο Morcel χρησιμοποίησε απλό φάσμα, διαιρούμενο στην αναλογία ρυθμού φάσματος και τον προβλεπόμενο χρόνο σύγκρουσης. Ο χρόνος πλησιέστερης προσέγγισής του διαφέρει απ'το κριτήριο του Morcel, ο χρόνος πλησιέστερης προσέγγισης είναι πιο αξιόπιστος και αποτελεσματικός για να κρίνει εάν μια σύγκρουση θα συμβεί ή όχι. Στο παραπάνω παράδειγμα, το υψόμετρο καθορίστηκε έτσι ώστε τα UAVs να μπορούν να πετάξουν μόνο σε οριζόντιο επίπεδο. Ο Krozel και ο Peters περιέγραψαν επίσης την πλήρως 3D ανίχνευση σύγκρουσης.

Ο Park απέδειξε ότι ο χρόνος πλησιέστερης προσέγγισης t θα είναι μια θετική τιμή όταν δυο από τα UAVs πλησιάζουν το ένα με το άλλο, και το t θα είναι αρνητικό όταν δυο απ'τα UAVs απομακρύνονται. Βάση αυτού του συμπεράσματος, ελέγχουν την πιθανότητα συγκρούσεων μόνο όταν το t είναι αρνητικό. Προκειμένου να καθορίσουν τα UAVs σε κατάσταση σύγκρουσης, θέτουν μια απόσταση ασφαλείας. Όταν το μέγεθος του $\vec{w}$ είναι μικρότερο απ'την απόσταση r_{safe} , υπάρχει προβλεπόμενη σύγκρουση μεταξύ δυο UAVs. Αφού συμβεί η σύγκρουση, θα πρέπει να επιτευχθεί η επίλυσή της. Μπορούμε να ξέρουμε διαισθητικά ποια κατεύθυνση θα διαλέξει κάθε UAV για να αποφύγει τις συγκρούσεις στην εικόνα 2.2 και να λυθεί το πρόβλημα.

2.2 Στοχαστική Προσέγγιση

Σύμφωνα με αυτή την προσέγγιση, το πρόβλημα αποφυγής της σύγκρουσης διατυπώνεται ως ο καλύτερος έλεγχος του στοχαστικού συστήματος. Η MDP αποτελεί αντιπροσωπευτική προσέγγιση τέτοιου είδους απ'τους Temizer, Kochenderfer and Kaelbling.

2.2.1 MDP

Η MDP πήρε την ονομασία της απ'το Ρώσσο μαθηματικό Andrey Markon που είναι γνωστός για τη συμβολή του στη θεωρία της στοχαστικής διαδικασίας. Την δεκαετία του 1950, ξεκίνησε να συζητιέται η MDP. Μετά τη δημοσίευση του βιβλίου του Howard "Dynamic Programming and Markou Processes" αναπαράχθηκαν πολλές έρευνες για την MDP.

Η MDP είναι προέκταση του Marcov Clain, εισάγει επιπλέον ιδέες, όπως δράσεις (επιτρέποντας την επιλογή) και τα οφέλη (δίνοντας κίνητρα). Η MDP είναι μια διακριτή στοχαστική διαδικασία προγραμματισμού όπου η κατάσταση του συστήματος αλλάζει τυχαία σύμφωνα με την τρέχουσα κατάσταση και δράση. Μια MDP προσδιορίζεται ως πλειάδα $(S, A, P^a_{ss'}, R^a_{ss'}, \gamma)$ όπου S είναι η κατάσταση και A η δράση. Παίρνοντας την δράση A απ'την κατάσταση S, υπάρχει πιθανότητα $P^a_{ss'}$ να διέλθει απ'την κατάσταση S και λαμβάνουν αμοιβή $R^a_{ss'}$. Τέλος, $\gamma \in [0,1]$ είναι ο συντελεστής που χρησιμοποιείται για να δώσει προτεραιότητα στις τωρινές ανταμοιβές έναντι μελλοντικών ανταμοιβές και η τιμή είναι συνήθως κοντά στο 1.

Η MDP επιδιώκει την αυτόματη δημιουργία αλγορίθμων αποφυγής συγκρούσεων δεδομένων των μοντέλων αεροσκαφών, την απόδοση αισθητήρα και τη συμπεριφορά εισβολέα. Διατυπώνοντας το πρόβλημα της αποφυγής συγκρούσεων ως MDP για τους αισθητήρες που παρέχουν ακριβή εντοπισμό του αεροσκάφους/εισβολέα, ή ως POMDP για τους αισθητήρες που έχουν αβέβαιη θέση ή περιορισμένο ορατό πεδίο, οι MDP/POMDP μπορούν να χρησιμοποιηθούν για να παράγουν στρατηγικές αποφυγής που βελτιώνουν μια συνάρτηση κόστους που εξισορροπεί την παρέκληση του σχεδίου πτήσης με τη σύγκρουση. Ωστόσο, ο υπόλογιστικός χρόνος μπορεί να είναι αρκετός. Η τρισδιάστατη κίνηση σε διακριτοποιημένη σύνθεση, θα λάβει το μέγεθος της κατάστασης πέρα από το φάσμα των υφιστάμενων λυτών. Άλλες αναπαραστάσεις για το χώρο κατάστασης και νέα είδη λυτών διευρενώνται επί του παρόντος.

2.3 Προσέγγιση Γραμμικού Προγραμματισμού

Ο γραμμικός προγραμματισμός είναι μια μαθηματική μέθοδος για την βελτίωση μιας γραμμικής αντικειμενικής λειτουργίας. Το 1939, ο Leonid Kantorovich ανέπτυξε το πρώτο πρόβλημα γραμμικού προγραμματισμού για να υπολογίσει τις δαπάνες κατά τη διάρκεια του Β' Παγκοσμίου Πολέμου. Τη δεκαετία του 1990, ο Bemporad και ο Paul πρότειναν την ίδια ιδέα, δηλαδή ότι το πρόβλημα του γραμμικού προγραμματισμού μπορεί να εφαρμοστεί ως γραμμικοί περιορισμοί σε ένα μείγμα συνεχόμενων και ακέραιων μεταβλητών.

2.3.1 Γραμμική Προγραμματιστική Προσέγγιση

Εδώ το πρόβλημα αποφυγής της σύγκρουσης διατυπώνεται ως πρόβλημα γραμμικού προγραμματισμού. Το MILP είναι μια αντιπροσωπευτική τεχνική για τη λύση του προβλήματος αποφυγής της σύγκρουσης ενός UAV χρησιμοποιώντας τον γραμμικό προγραμματισμό. Το MILP παρέχει το καλύτερο αποτέλεσμα για ένα συγκεκριμένο μαθηματικό μοντέλο περιέχοντας ένα σύνολο γραμμικών περιορισμών. Αρχικά, ένας αριθμός περιορισμών πρέπει να προσδιοριστεί για να δημιουργήσει την δυναμική κίνηση του UAV. Επιπλέον, πρέπει να παρασχεθούν περιορισμοί σχετικά με την αποφυγή της σύγκρουσης. Αυτοί οι περιορισμοί τροφοδοτούνται με εμπορικούς λύτες MILP όπως APML και MATLAB, οι οποίοι αναπτύσουν την καλύτερη διαδρομή για το UAV. Κάθε διαδρομή του αεροσκάφους βελτιώνεται βασιζόμενο στις αναμενόμενες διαδρομές των άλλων ανταγωνιστικών αλλά συνεργαζόμενων UAV. Γι'αυτό, ένα μη συνεργαζόμενο UAV μπορεί να σπείρει τον όλεθρο στο προστατευτικά σχεδιασμένο σύστημα.

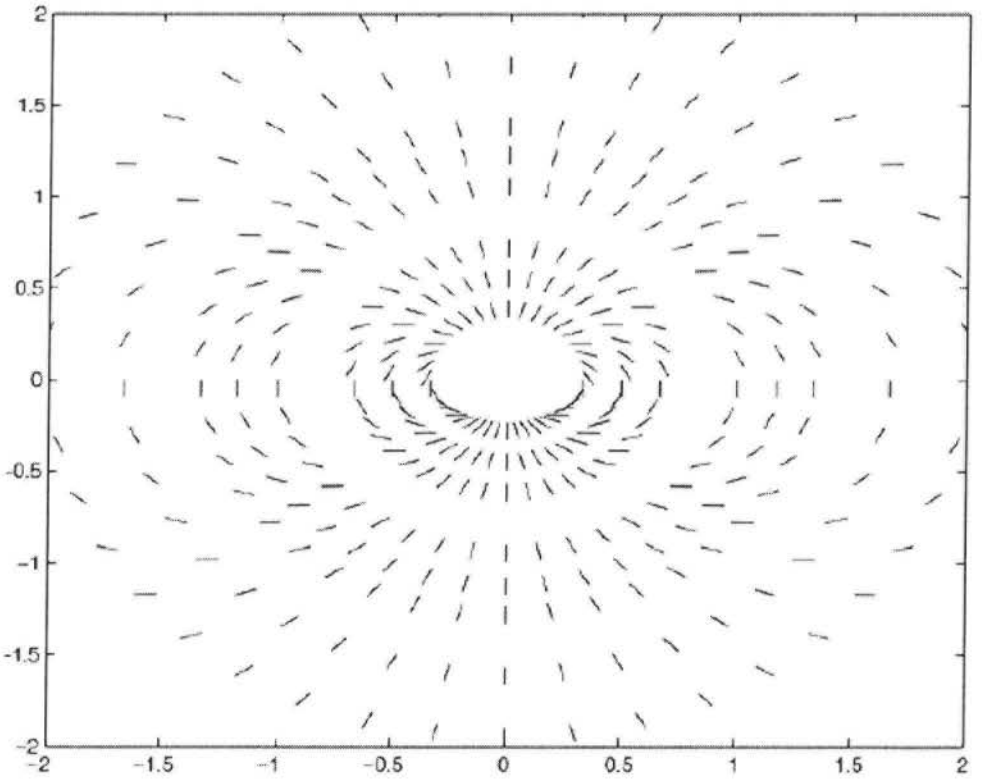
Το MILP είναι υπολογιστικά ακριβό. Εάν η βέλτιστη λύση παίρνει τόσο πολύ χρόνο, το MILP μπορεί να λύσει το πρόβλημα και να παράσχει λιγότερες βέλτιστες λύσεις στον προβλεπόμενο καθορισμένο χρόνο. Καθώς ο καθορισμένος χρόνος αυξάνεται, το MILP παρέχει όλο και καλύτερες βέλτιστες λύσεις. Για να μειωθεί το υπολογιστικό κόστος, χρησιμοποιείται μια τεχνική προερχόμενη από τον μοντέλο πρόβλεψης ελέγχου, γνωστή ως «receding horizon». Χρησιμοποιώντας την προσέγγιση της «receding horizon», η διαδρομή του UAV διασπάται σε μικρά κομμάτια όπου το μοντέλο MILP λύνεται για N χρονικά βήματα, τα οποία εξυπηρετούν ως δύναμη για τα επόμενα N χρονικά βήματα και ούτω καθ'εξής μέχρι να τελειώσει η διαδικασία. Ενώ αυτή η προσέγγιση ελαχιστοποιεί το υπολογιστικό βάρος, όταν το πρόγραμμα λύνεται σε μικρότερα blocks, εμπόδια που βρίσκονται εκτός χρονικών ορίων απ'το UAV δεν επηρεάζουν την διαδρομή του UAV. Γι'αυτό, όταν οι διαδρομές πρέπει να υπολογιστούν για το επόμενο χρονικό όριο, τα UAVs πρέπει να είναι τόσο κοντά ώστε να αποφύγουν τη σύγκρουση. Έτσι, χρησιμοποιώντας την προσέγγιση MILP, ο στόχος είναι να ελαχιστοποιηθεί ο χρόνος υπολογισμού ενώ αυξάνεται το μέγεθος των χρονικών blocks που χρησιμοποιούνται στον υπολογισμό.

2.4. Προσέγγιση δυναμικών πεδίων

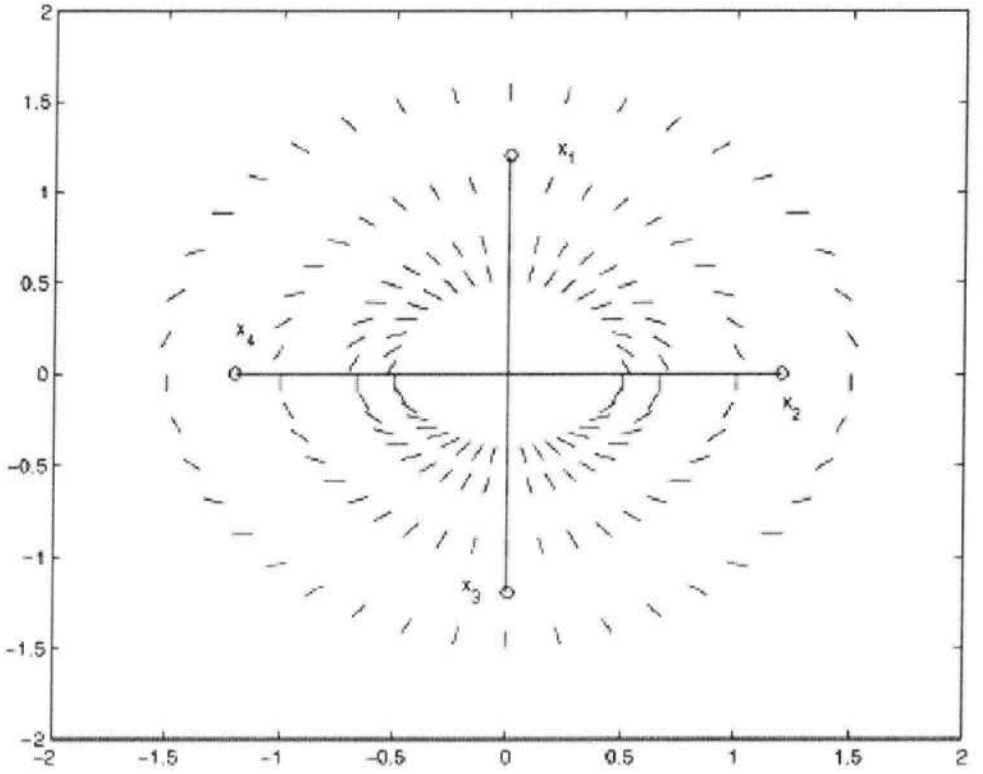
Αυτή η προσέγγιση προτάθηκε για πρώτη φορά το 1986 από τον Khatib. Αυτή η μέθοδος αναθέτει μαγνητικές ή ηλεκτρικές δαπάνες του ίδιου σήματος στα UAVs και τις αντίθετες δαπάνες στους προορισμούς, βασισμένη στους φυσικούς νόμους, σωματίδια με τις ίδιες δαπάνες μπορούν να αποκρούσουν το ένα το άλλο ενώ προσελκύονται απ'τους προορισμούς (αντίθετη δαπάνη).

2.4.1 Συνολικό πεδίο

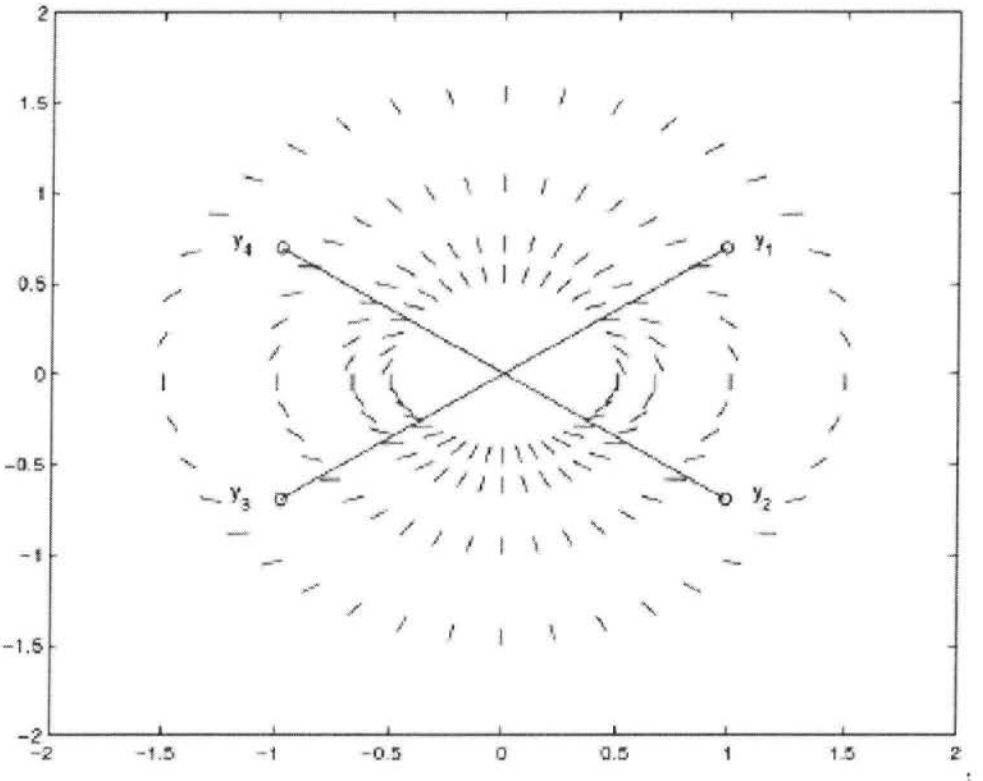
Ο Sigurd και ο How εξέτασαν εάν είναι εφικτό να παραχθούν δυναμικά πεδία χωρίς να είναι απαραίτητο να γνωρίζουν την θέση των άλλων αεροσκαφών. Η αποκρουστική δύναμη ανάμεσα στα μαγνητικά πεδία που παράγονται από κάθε UAV επιτρέπει σε αυτά να αποφεύγουν το ένα το άλλο αυθόρμητα. Ένας επιπλέον μαγνητικός αισθητήρας και μια τοπική μαγνητική γεννήτρια απαιτούνται στο σκάφος σύμφωνα με αυτή την προσέγγιση. Γι'αυτό, κάθε UAV μπορεί να συμπεριφερθεί ως μαγνητικό δίπολο. Η μαγνητική πυκνότητατη ροής B/|B|στο xy-plane γύρω από ένα μαγνητικό δίπολο φαίνεται στην εικόνα 2.3. Παράγοντας τοπικό πεδίο, μπορούν να βρουν μια σειρά γωνιών όπου το πεδίο του UAV είναι ορθογώνιο στον άξονα x ή στον άξονα y στο πλαίσιο αναφοράς του UAV. Αυτές οι γωνίες φαίνονται στην εικόνα 2.4.



Εικόνα 2.3



Εικόνα 2.4 (άξονας x)



Εικόνα 2.4 (άξονας y)

2.4.2 Δυναμικό πεδίο

Ο Liu, ο Wang και ο Dissanayakes παρουσιάζουν έρευνα σχετικά με τεχνητά δυναμικά πεδία στη ρομποτική αποφυγή συγκρούσεων. Έλυσαν το πρόβλημα εντοπισμού διαδρομής για μια κατηγορία ομοιοκατευθυντικών οχημάτων. Μοντελοποιώντας τα σημεία-στόχο ως θετικά φορτία και τα άλλα ρομπότ στο χώρο της άσκησης ως αρνητικά, βρέθηκαν βέλτιστες διαδρομές υπολογίζοντας μια συνολική δύναμη των ρομπότ προς την κατεύθυνσή τους και την απωστική δύναμη μακριά από τα άλλα οχήματα που κινούνται στην περιοχή. Μια σημαντική καινοτομία είναι η χρήση ελλειπτικών δυναμικών πεδίων η οποία καταλήγει σε μικρότερη προσπάθεια για το ρομπότ να συνεχίσει αφού το τεχνητό πεδίο που δημιουργείται από ένα ρομπότ εκτείνεται πιο μακριά στο χώρο μπροστά απ'αυτό. Μια άλλη σημαντική καινοτομία περιλαμβάνει την χρήση των επεκτατικών δυναμικών πεδίων. Καθώς ένα ρομπότ κινείται πιο κοντά στο στόχο του, το τεχνητό πεδίο αυξάνεται σε μέγεθος για να αποκρούσει άλλα ρομπότ στην περιοχή.

2.5 Προσέγγιση βασισμένη στο πλέγμα

Η προσέγγιση αυτή διακριτοποιεί τον αέρα του σε διακριτά τετράγωνα ή κυβικά κελιά ομοιόμορφου μεγέθους. Η συλλογή των κυττάρων αναπαριστάνεται από ένα σταθμισμένο γράφημα στο οποίο κορυφές είναι τα κελιά και οι άκρες συνδέουν γειτονικά κελιά. Σ'ένα αεροπλάνο, μια κορυφή έχει 8 άκρες: ορθωγώνια (βόρεια, ανατολικά, νότια, δυτικά) και διαγώνια (βορειοδυτικά, νοτιοανατολικά και βορειοανατολικά). Μεγάλα βάρη στις άκρες μπορούν να χρησιμοποιηθούν για να αποδείξουν ένα εμπόδιο. Αυτή η γραφική αναπαράσταση επιτρέπει την χρήση συντομότερης διαδρομής αλγορίθμων όπως του Dijkstra ή του Bell-Ford.

2.5.1 Ερευνητικά Προβλήματα

Το 1959, ο Dijkstra μελέτησε τα προβλήματα ελάχιστου κόστους σε συνδυασμό με τα γραφήματα. Λαμβάνοντας υπόψη n κώδικες, υπάρχουν κλάδοι συνδεδεμένοι με δεδομένα μήκη. Υποθέτοντας ότι ένας κόμβος έχει τουλάχιστον μια διαδρομή, η διαδρομή με το μικρότερο μήκος ανάμεσα στους δυο κόμβους P και Q μπορεί να βρεθεί με τον εξής τρόπο:

1. Δημιουργία τριών σετ για τους κλάδους: I, II, III και τριών σετ για τους κόμβους: A, B, C.

2. Τοποθέτηση όλων των κόμβων στο C και όλων των κλάδων στο III.

3. Μετακίνηση του κόμβου P στο A και προσδιορισμό αυτού του κόμβου ως τρέχοντα.

4. Εξετάζοντας όλους τους κλάδους r που συνδέονται με τον τρέχον κόμβο με κόμβο R_i μεταξύ του σετ B, θεωρούμε ότι κάνει χρήση του κλάδου r_i και παρέχει μικρότερη απόσταση από τη χρήση των αντίστοιχων κλάδων στο σετ II. Αν συμβαίνει αυτό, ο κλάδος r_i αντικαθιστά τον κλάδο στο σετ II. Διαφορετικά, ο κλάδος r_i απορρίπτεται και μεταφέρεται στο σετ III.

5. Αν το i_{th} του κόμβου R_i δεν ανήκει στο σετ B, μεταφέρουμε τον κόμβο R_i στο σετ B και τον κλάδο r_i στο σετ II.

6. Εάν ακολουθήσουμε τον κανόνα που πρέπει να χρησημοποιήσουμε τους κλάδους από το σετ I και έναν από το II, τότε κάθε κόμβος στο B μπορεί να συνδεθει στον κόμβο P με μόνο έναν τρόπο. Γι'αυτό, μπορούμε να υπολογίσουμε την απόσταση μεταξύ του κόμβου P και κάθε κόμβου στο B. Μεταφέρουμε τον κόμβο με την ελάχιστη απόσταση από τον P στον A και το αντίστοιχο κλάδο στο I. Κάνουμε αυτό τον κόμβο τρέχοντα κόμβο.

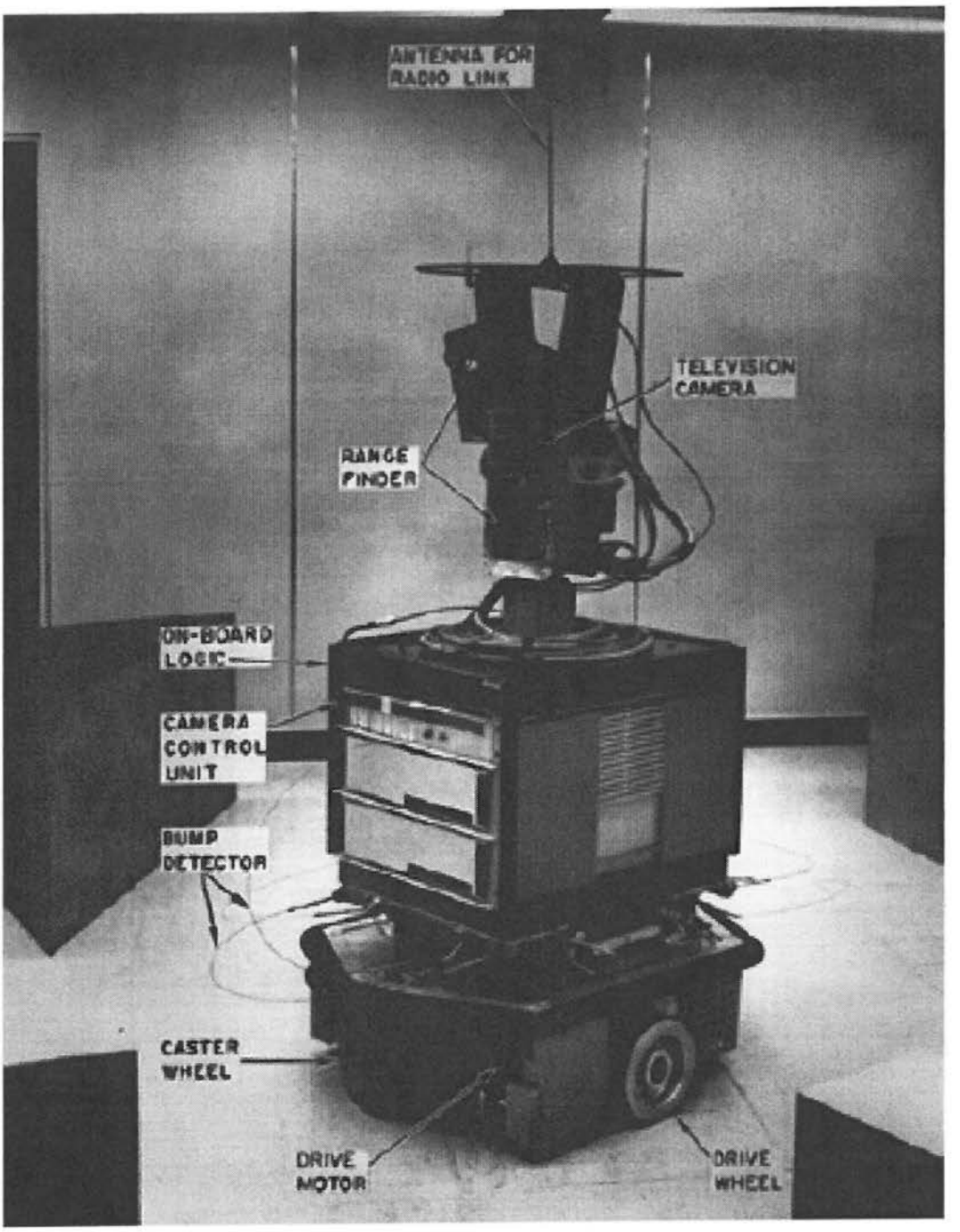
7. Επιστρέφουμε στο βήμα 4 και επαναλαμβάνουμε τη διαδικασίας μέχρι ο κόμβος Q να μεταφερθεί στο I. Τότε οι κόμβοι στο A είναι η λύση.

Ο αλόριγμος του Dijkstra μπορεί να λύσει προβλήματα τα οποία μπορούν να μειωθούν στην ελάχιστη διαδρομή ενός σταθμισμένου γραφήματος. Το γράφημα δεν είναι απαραίτητο να προσδίδει το ίδιο βάρος με τους κλάδους. Επιπλέον, το βάρος μπορεί να ποικίλει ανάλογα με την κατεύθυνση που ακολουθεί. Ένας περιορισμός για τον αλγόριθμο του Dijkstra είναι ότι δεν δουλεύει σε γράφημα με αρνητικά βάρη. Ένας αλγόριθμος με όμοια δομή, ο οποίος λέγεται Bell-Ford, εφαρμόζεται όταν υπάρχουν αρνητικά βάρη στα γραφήματα.
Παρόλο που ο αλγόριθμος του Dijkstra μπορεί να λύσει το πρόβλημα του ελάχιστου κόστους σε μεγάλο βαθμό, δεδομένου του μεγάλου αριθμού των κόμβων, ο υπολογισμός μπορεί να είναι απαγορευτικός. Λίγα χρόνια αργότερα, μια γενίκευση του αλγόριθμου Dijkstra πρωτάθηκε για να μειώσει τους κόμβους, η οποία πρέπει να διερευνηθεί.

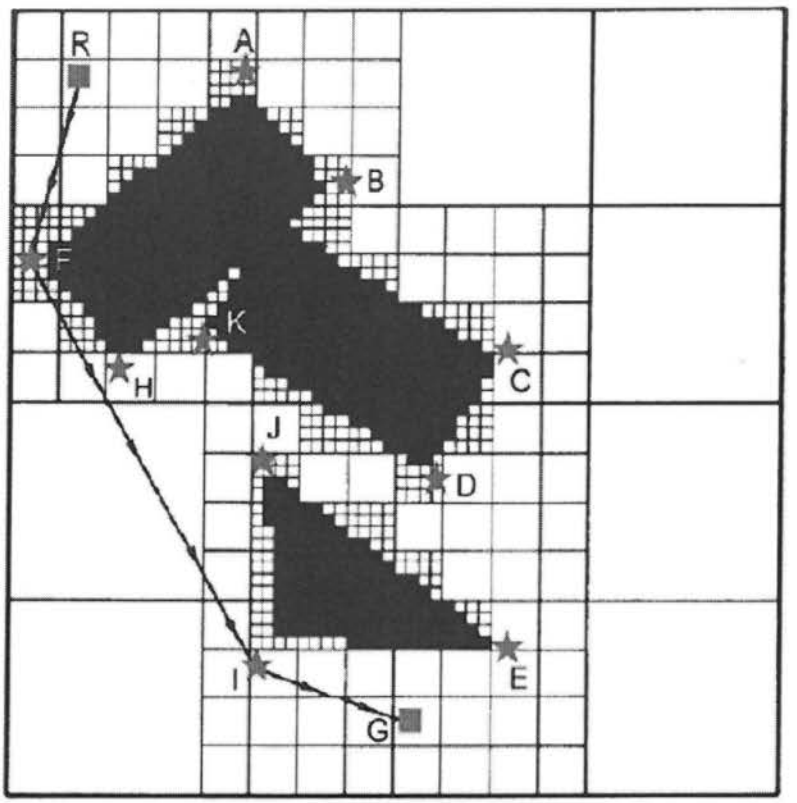
2.5.2 Ευρετικές Λειτουργίες

Το 1966, διεξήχθει η έρευνα για την κατασκευή ενός κινητού ρομπότ ονίματι «Shakey» από το SRI (τότε Ινστιντούτο Έρευνας Stanford). Η εικόνα 2.5 απεικονίζει το Shakey, υπάρχει μια κάμερα πάνω σε αυτό, ώστε να μπορεί να πιάνει εικόνες γύρω του για να ανιχνεύει τα εμπόδια. Η μέθοδος πλοήγησης είναι ο σχεδιασμός μιας αλληλουχίας σημείων ώστε το Shakey να μπορεί να το ακολουθήσει απ'το ένα μέρος στο άλλο.

Το Shakey αποθήκευε τις θέσεις των εμποδίων και τη δική του σε ένα μοντέλο βασισμένο σε πλέγμα, όπως φαίνεται στην εικόνα 2.6. Χρησιμοποιούσαν μικρότερα κελιά κοντά στα εμπόδια ώστε να περιγράφουν τις θέσεις με μεγαλύτερη ακρίβεια. Ο Nilsson, μέλος της ερευνητικής ομάδας του Shakey, δήλωσε: «Αυτή θα μπορούσε να είναι μια από τις πρώτες εφαρμογές της αποσύνθεσης κελιού στη ρομποτική κίνηση και τώρα είναι μια συνηθισμένη τεχνική». Τα προβλήματα πλοήγησης του Shakey, μπορούν να μειωθούν σε ένα ερευνητικό πρόβλημα, σε ένα σταθμισμένο γράφημα και σε πολλές προσεγγίσεις για τα ερευνητικά γραφήματα που πρωτάθηκαν εκείνη την περίοδο, συμπεριλαμβανομένου του αλγόριθμου του Dijkstra και του αλγόριθμου του Bell-Ford που προαναφέραμε. Αυτές οι προσεγγίσεις μπορούσαν να λύσουν το πρόβλημα πλοήγησης του Shakey το οποίο εγγυόταν την πιο σύντομη διαδρομή. Ωστόσο, μπορεί να ήταν αναποτελεσματικές για τον υπολογισμό με πολύπλοκα προβλήματα όπως ο μεγάλος αριθμός κόμβων.


ANTENNA FOR RADIO LINK
TELEVISION CAMERA
RANGE FINDER
ON-BOARD LOGIC
CAMERA CONTROL UNIT
BUMP DETECTOR
CASTER WHEEL
DRIVE MOTOR
DRIVE WHEEL

Εικόνα 2.5



Εικόνα 2.6

Εμπνευσμένος από τα πειράματα των Doran και Michie, με το πρόγραμμα Graph Traverser, όπου προσδιορίζουν ευρετικές λειτουργίες για να προσδιορίσουν κόστη με βάση την εκτιμόμενη δυσκολία του να φτάσεις τον προορισμό για να ασχοληθείς με το πρόβλημα του παζλ συρταρωτών πλακιδίων , ο Nilsson έκρινε ότι για ένα πρόβλημα πλοήγησης κινητού ρομπότ, μια καλή ευρετική λειτουργία για να εκτιμήσεις τη δυσκολία του να φτάσεις τον προορισμό (πριν αναζητήσεις κόμβους) θα μπορούσε να είναι η απόσταση αερογραμμής, που σημαίνει το μήκος της διαδρομής αγνοώντας κάθε εμπόδιο για να φτάσεις στον προορισμό. Ο Nilsson πρότεινε την χρήση αυτής της ευρετικής λειτουργίας ως τα βάρη των αντίστοιχων κόμβων στην έρευνα.

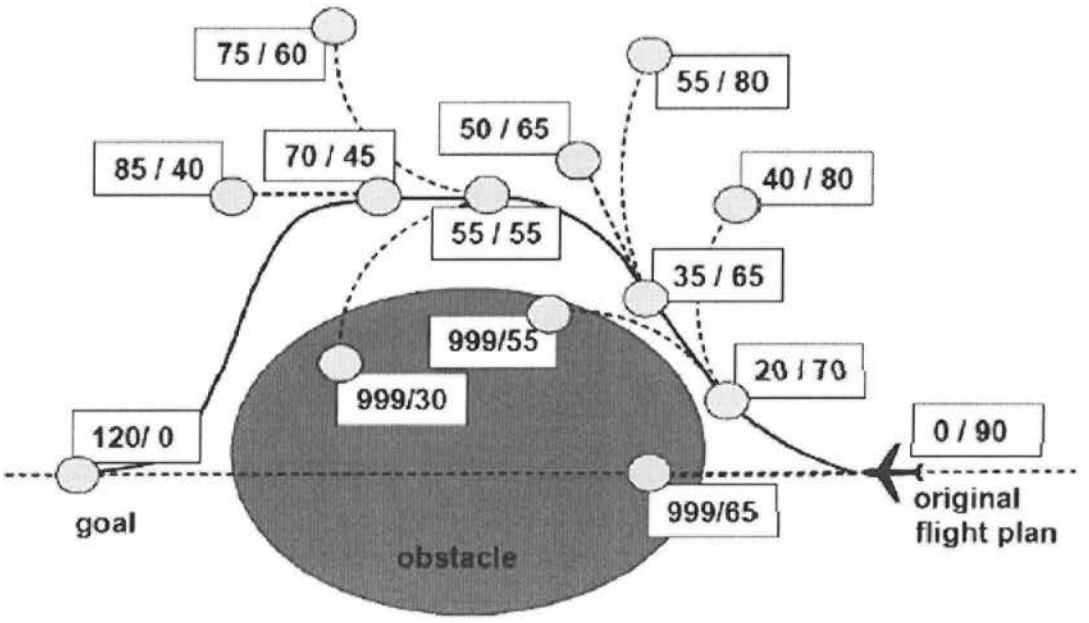
Ο Raphael βελτίωσε την ιδέα του Nilsson για το πρόβλημα πλοήγησης του Shackey. Εκτός από την αποκλειστική χρήση της ευρετικής λειτουργίας ως τα βάρη των αντίστοιχων κόμβων στην έρευνα, ο Raphael πρότεινε τον συνδιασμό της ευρετικής λειτουργίας που πρότεινε ο Nilsson με το απομακρυσμένο ρομπότ που ταξίδευε μέχρι τώρα από την αρχική θέση ως το συνολικό βάρος των κόμβων.

Ο Hart, ένα μέλος της ομάδας των Nilsson και Raphael στο SRI, μελέτησε τον αλγόριθμο του Nilsson και Raphael και παρατήρησε ότι η εναπομείνουσα απόσταση που η ευρετική λειτουργία δεν ήταν ποτέ μεγαλύτερη, εκτίμησε απ'την πραγματική εναπομείνουσα απόσταση, τότε η διαδρομή που βρήκαν οι αλγόριθμοί τους θα μπορούσε να είναι η καλύτερη λύση στο πρόβλημα αυτό. Επίσης ο Hart πίστευε ότι οι αλγόριθμοί τους θα ήταν οι πιο αποτελεσματικές διαδικασίες στην έρευνα του προβλήματος συγκριτικά με άλλες υπάρχουσες προσεγγίσεις που εγγυήθηκαν να δώσουν την καλύτερη λύση.

2.5.3 Αλγόριθμος A*

Το 1968, ο Hart, ο Nilsson και ο Raphael πρότειναν μια αποδεδειγμένη βελτίωση του αλγόριθμου του Dijkstra ονόματι A*. Ο A* είναι ένας ενημερωμένος αλγόριθμος: αυτός διερεύνησε πρώτος τις διαδρομές που φαίνονταν να οδηγούν στον προορισμό. Γι'αυτό, ο A* χρησιμοποιεί ευρετικό κόστος και απόσταση για να προσδιορίσει τη σειρά στην οποία επισκέπτονται πιθανούς κόμβους της διαδρομής. Ο A* βελτιώνει το χρόνο του αλγορίθμου του Dijkstra και δουλεύει καλά τον σχεδιασμό διαδρομής για ρομπότ που κυκλοφορούν γύρω από τα εμπόδια.

Για την αποφυγή της σύγκρουσης, άλλα UAVs είναι τα κινούμενα εμπόδια. Οι Tooren, Heni, Knoll και Beck υιοθέτησαν και εφάρμοσαν τον A* για τα UAVs ως έρευνα πέρα από τους κόμβους για αρχέτυπα κίνησης που είναι μικρά τμήματα τροχιάς. Κάθε ένα απ'αυτά τα αρχέτυπα κίνησης επιλέγεται ώστε να παράγει πτητική, ομαλή τροχιά κατάλληλη για την κατάσταση του αεροσκάφους. Η εικόνα 2.7 απεικονίζει κάποιους διευρυμένους κόμβους που προκύπτουν απ'τα τυπικά αρχέτυπα κίνησης με το κόστος τους που σημειώνονται στα κουτιά όπου G είναι το κόστος απ'το σημείο έναρξης μέχρι το τρέχον σημείο και H είναι το αντιληπτό ευρετικό κόστος απ'το τρέχον σημείο, στον προορισμό. Να σημειωθεί ότι για την απλούστευση, δεν φαίνονται όλα τα πιθανά τμήματα και η ελάχιστη διαδρομή έχει σχεδιαστεί ως μία συνεχής γραμμή. Το σύνολο των πιθανόν σημείων περιλαμβάνει τυπικά στοιχεία πτήσης όπως γραμμικά τμήματα, καμπυλωτά τμήματα αλλά και πιο πολύπλοκα στοιχεία όπως Dubins. Ο υπολογιστικός χρόνος ποικίλει καθώς ο εναέριος χώρος αλλάζει. Όσο ο εναέριος χώρος αυξάνεται, τόσο αυξάνεται η υπολογιστική πολυπλοκότητα. Απ'την άλλη, εάν ο εναέριος χώρος διασπάται σε λίγότερο μεγάλα κελιά, μπορεί να μην μπορούν να σχεδιαστούν αποτελεσματικά μη συγκρουόμενες διαδρομές.



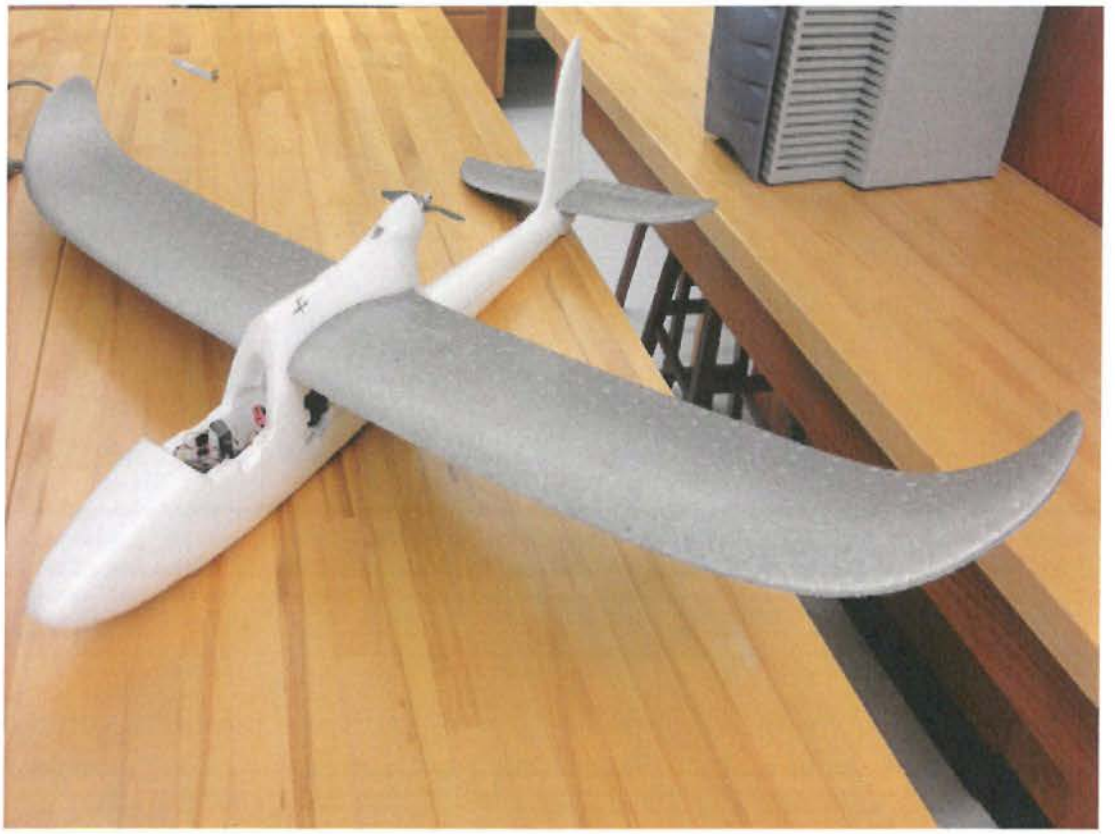
Εικόνα 2.7

2.6 Συμπέρασμα

Σε ένα δυναμικό περιβάλλον, ο υπολογιστικός χρόνος είναι κρίσιμος. Ο υπολογιστικός χρόνος του A* ποικίλει όσο ποικίλει και ο εναέριος χώρος. Η περιπλοκότητα του A* μπορεί να μειωθεί σχεδιάζοντας μια καλή ευρετική λειτουργία έτσι ώστε ο A* να έχει μεγαλύτερη ευελιξία να προσαρμόσει διάφορα σενάρια. Το πλεονέκτημα του A* είναι ότι μπορεί να είναι ευέλικτος επιτρέποντας σε διαφορετικές ευρετικές λειτουργίες να χειριστούν διαφορετικές καταστάσεις. Ο A* είναι ένας καλός υποψήφιος για την εκτέλεση διαδρομής σχεδιασμένη σε πραγματικό χρόνο.

ΚΕΦΑΛΑΙΟ 3

ΑΠΟΦΥΓΗ ΣΥΓΚΡΟΥΣΗΣ ΤΩΝ UAVs



Εικόνα 3.1

3.1 Easy Star

Η προσέγγιση που θα προτείνουμε σ'αυτό το κεφάλαιο θα εφαρμοστεί στην πλατφόρμα του UAV που ονομάζεται EASY STAR (βλ. εικόνα 3.1). Το Easy Star είναι ένα UAV σταθερών πτερυγίων, το οποίο αποτελείται από σώμα, ένα πτερύγιο, ένα πηδάλιο, έναν ανελκυστήρα, έναν κινητήρα και μια ανάγλυφη διάταξει που βρίσκονται τα ηλεκτρονικά, όπως το Arduino επί του σκάφους. Όπως ο κινητήρας ελέγχει την ταχύτητα του Easy Star, το πηδάλιο, ο ανελκυστήρας και το πτερύγιο ελέγχουν την κατεύθυνση της πτήσης. Το Arduino μπορεί να μεταδόσει και να λάβει πληροφορίες απ'τον επίγειο σταθμό (όπως ένας φορητός υπολογιστής) ή από άλλα UAVs.

3.2 Ανίχνευση Σύγκρουσης

Μια σύγκρουση μπορεί να οριστεί ως μια προβλεπόμενη παράβαση του προτύπου διασφάλισης του διαχωρισμού. Υπό αυτή την έννοια, σύγκρουση υφίσταται όταν δύο αεροσκάφη συναντηθούν εντός της ασφαλούς απόστασης κάποια στιγμή στο μέλλον. Με την προσέγγισή μας, προβλέπουμε τις θέσεις των UAVs στο προσεχές μέλλον και χρησιμοποιούμε αυτές τις πληροφορίες για να αποφευχθούν συγκρούσεις. Εφαρμόζοντάς τες στην ευρετική λειτουργία του αλγορίθμου A*, μπορούμε να πετύχουμε την ευαισθητοποίηση του εισβολέα.

3.2.1 Η ευαισθητοποίηση του εισβολέα

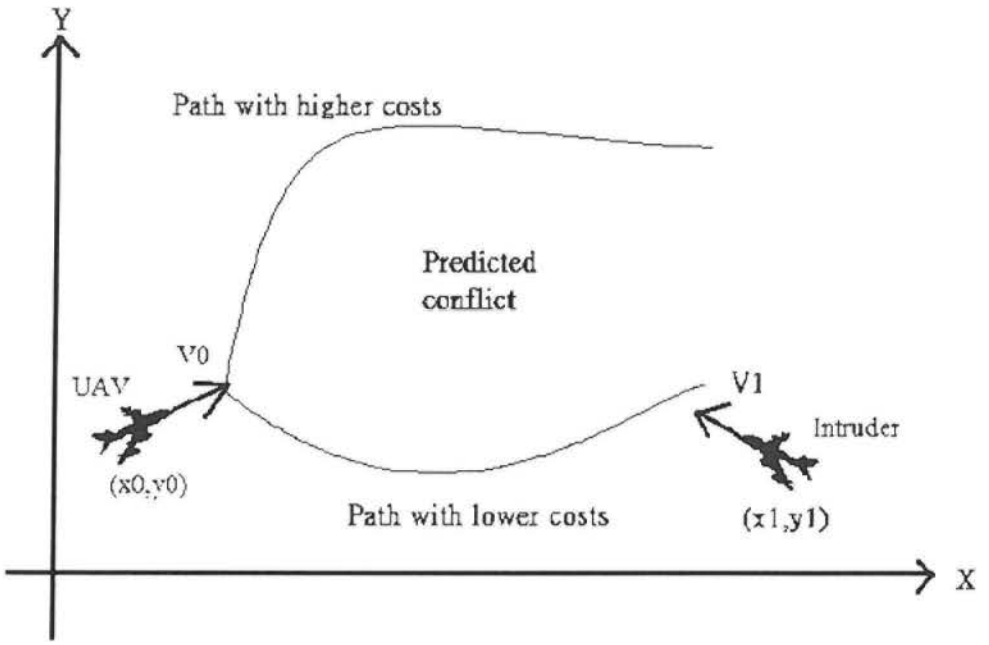
Αποκαλούμε ευαισθητοποίηση εισβολέα τον εκτιμώμενο κίνδυνο που υπολογίζουμε στην ευρετική λειτουργία. Η εικόνα 3.2 απεικονίζει πώς θα πετύχουμε την ευαισθητοποίηση εισβολέα.

- Πρώτα υπολογίζουμε την διαφορά των συντεταγμένων x του UAV και του εισβολέα: $x_{01}=x_{12}-x_0$, όπου το x_{12} μας λέει εάν ο εισβολέας βρίσκεται δεξιά ή αριστερά του UAV.

- Δευτέρον, η ταχύτητα των UAVs μπορεί να υπολογιστεί διαφοροποιώντας τις κινήσεις. Υπολογίζοντας την ταχύτητα, μπορούμε να πούμε αν ο εισβολέας πλησιάζει ή απομακρύνεται κατά μήκος του άξονα x ελέγχοντας το πρόσημα της ταχύτητας του άξονα x. Τέλος, πολλαπλασιάζουμε την σχετική θέση στον άξονα x με την ταχύτητα. Εάν το γινόμενο είναι αρνητικό, ο εισβολέας πλησιάζει κατά μήκος τον άξομα x. Με άλλα λόγια, το UAV είναι μπροστά απ'τον εισβολέα. Εφαρμόζοντας την ίδια διαδικασία στην κατεύθυνση y, μπορούμε να πούμε από ποια κατεύθυνση πλησιάζει ο εισβολέας.

- Τρίτον, με δεδομένα τις προβλεπόμενες θέσεις του UAV και τον αριθμό n των εισβολέων, μπορούμε να εκτιμήσουμε τον κίνδυνο στο μέλλον με την εξίσωση:

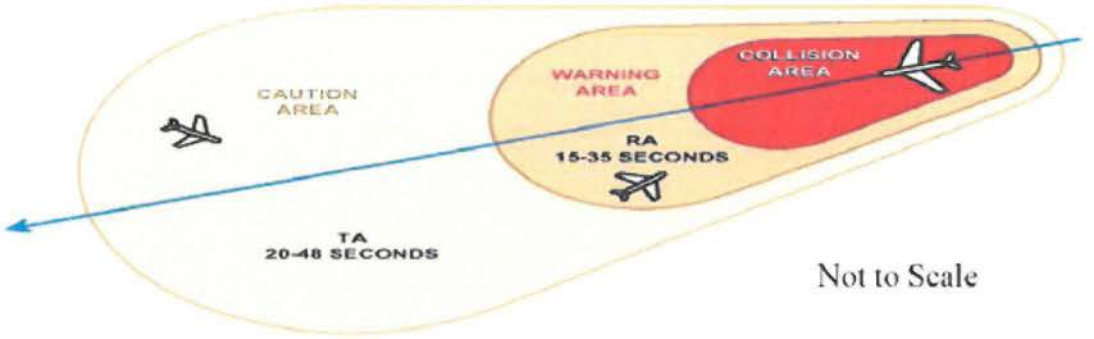
$$\sum_{i=1}^{n} |(X_i - X_0)| + |(Y_i - Y_0)|$$



Εικόνα 3.2

3.2.2 Προσδιορισμός της σύγκρουσης

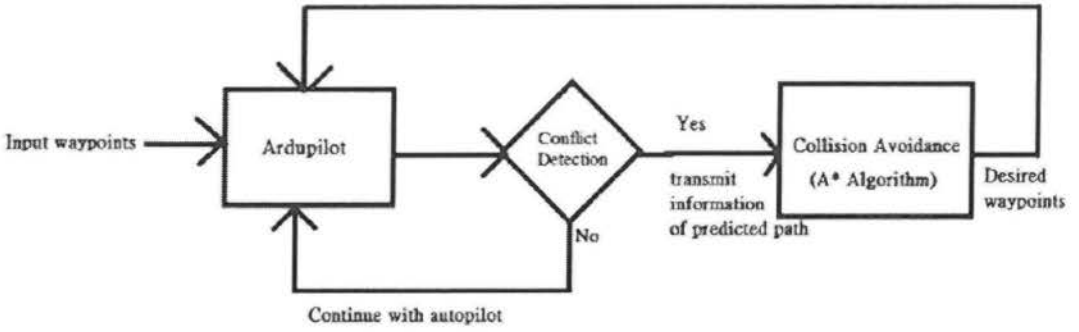
Για να ανιχνεύσουμε τις συγκρούσεις και να συμβουλεύσουμε τον πιλότο ώστε να αποφύγει ενδεχόμενη σύγκρουση, Σύστημα Αποφυγής Συγκρούσεων και Συναγερμού Κίνησης χρησιμοποιείται συνήθως στη διαχείρηση της εναέριας κυκλοφορίας συμπεριλαμβανομένου FAA. Ένα σύστημα TCAS προσδιορίζει διαφορετικό επίπεδο ζωνών πιθανών συγκρούσεων που θα έπρεπε να προστατευθούν γύρω απ'το αεροσκάφος όπως φαίνεται στην εικόνα 3.3. Στην περίπτωσή μας για το Easy Star, αναφερόμαστε στον προσδιορισμό του TCAS και στην απόσταση που τα UAVs μπορούν να καλύψουν σε 35 δευτερόλεπτα, απόσταση που ενεργοποιεί τη λειτουργία αποφυγής της σύγκρουσης. Στην προσομοίωσή μας, υποθέτουμε ότι τα UAVs πετούν 5 μέτρα ανά δευτερόλεπτο, έτσι η απόσταση ασφαλείας είναι 175 μέτρα.



Εικόνα 3.3

3.3 Αποφυγή σύγκρουσης

Μόλις ανιχνευτεί η ενδεχόμενη σύγκρουση, ενεργοποιούμε τον αλγόριθμο A* για να σχεδιάσουμε μια ασφαλή διαδρομή ώστε να αποφύγουμε τους εισβολείς. Η εικόνα 3.4 απεικονίζει την αρχιτεκτονική του μοντέλου αποφυγής της σύγκρουσης. Το Ardu Pilot είναι ένα λογισμικό άνευ κωδικού που φροντίζει για την πτήση του UAV. Δεδομένου ενός σημείου, το Ardu Pilot πετάει το AVL σε αυτό. Η ομάδα μας σχεδίασε λογισμικό μεταφόρτωσης στην πτήση νέων σημείων προς το Ardu Pilot. Η λειτουργία ανίχνευσης της σύγκρουσης συνεχώς παρακολουθεί όλα τα UAVs για να ανιχνεύσει επικείμενη σύγκρουση. Δεδομένων των θέσεων των UAVs αυτή η λειτουργία λαμβάνει την ταχύτητα όλων των UAVs και μπορεί να προβλέψει μια ενδεχόμενη σύγκρουση. Εάν προβλευθεί σύγκρουση, η λειτουργία ανίχνευσης της σύγκρουσης ενεργοποιεί τον A* και του παρέχει τις θέσεις των UAVs που μπορεί να συγκρουστούν. Οι μελλοντικές θέσεις όλων των UAVs υπολογίζονται 35 δευτερόλεπτα πριν. Τα 35 δευτερόλεπτα αντιστοιχούν στο χρόνο που προτείνεται TCAS. Όμως, ο χρόνος μπορεί να διαφέρει ανάλογα με την ταχύτητα των UAVs.



Εικόνα 3.4

Όταν ο A* ξεκινά, η λειτουργία ανίχνευσης της σύγκρουσης σταματάει για 35 δευτερόλεπτα. Μετά τα 35 δευτερόλεπτα, η λειτουργία ανίχνευσης της σύγκρουσης ενεργοποιείται ξανά για να αναβαθμίσει τις θέσεις για να εξαλείψει τις ασυμφωνίες μεταξύ του A* και του Ardu Pilot. Μόλις τα UAVs φτάσουν τόσο κοντά (σε απόσταση που τα UAVs μπορούν να φτάσουν σε 35 δευτερόλεπτα), ο A* παραμένει ενεργός. Βασισμένος στην πιο πρόσφατη αναβάθμιση, ο A* υπολογίζει τη νέα διαδρομή για να αποφύγει συγκρούσεις.

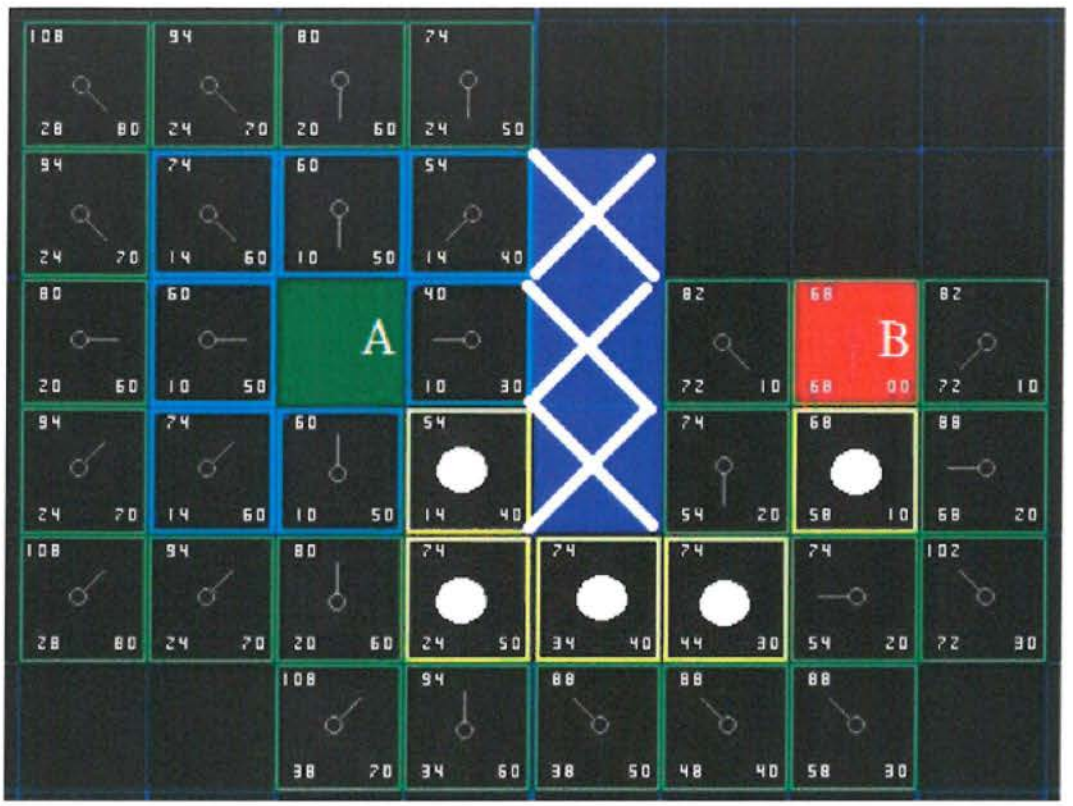
3.3.1 Αλγόριθμος A*

Ένα χαρακτηριστικό του αλγορίθμου A* είναι η ευκαμψία του κόστους λειτουργίας. Η εξίσωση του συνολικού κόστους είναι F=G+H όπου

- G είναι το κόστος απ'το σημείο έναρξης μέχρι το τρέχον (επισκεπτόμενο) σημείο και

- H, η ευρετική λειτουργία που υπολογίζει το κόστος απ'το τρέχον (επισκεπτόμενο) σημείο, στο σημείο προορισμού.

Όταν υπάρχει εμπόδιο που κάνει την τρέχουσα διαδρομή να κοστίζει περισσότερο από άλλες διαθέσιμες διαδρομές, ο A* θα επιστρέψει στον καινούριο κόμβο του χαμηλότερου κόστους. Κατόπιν επαναλήψεως της διαδικασίας, ο A* θα βρει τελικά μια διαδρομή που κοστίζει λιγότερο.

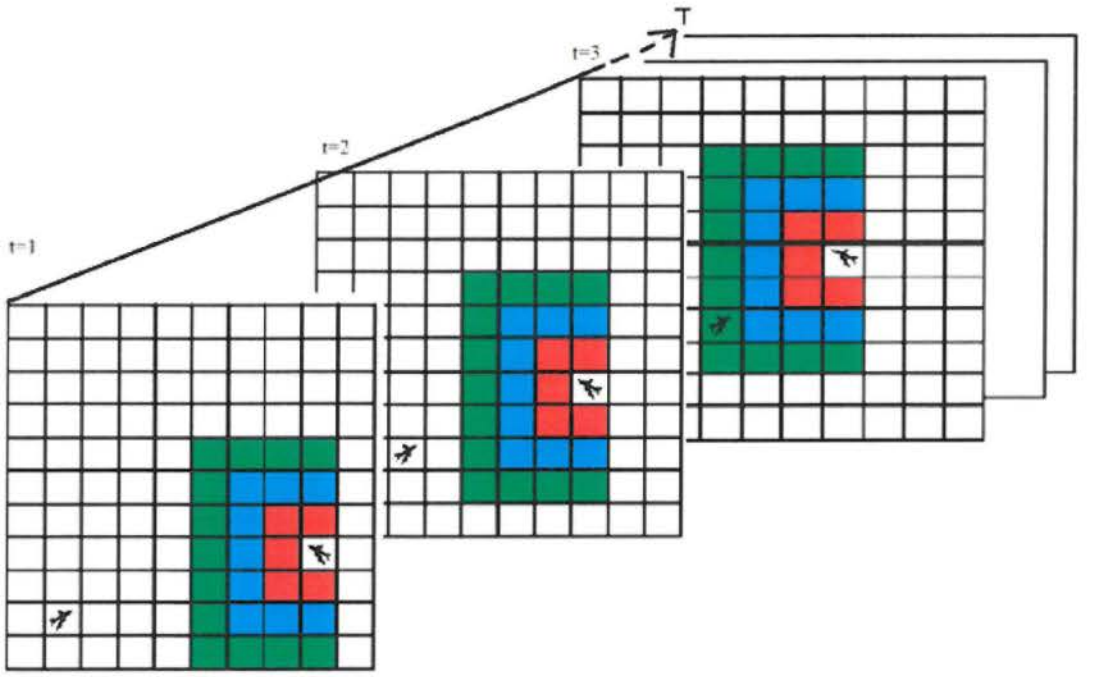


Εικόνα 3.5

Όπως φαίνεται στην εικόνα 3.5, ο A* ξεκινάει να εξαπλώνει κόμβους απ'το A στο B και τα X είναι τα εμπόδια. Στα πλέγματα περιλαμβάνονται οι εξής πληροφορίες: το F πάνω αριστερά, το G κάτω αριστερά , το H κάτω δεξιά και το βέλος δείχνει την πηγή του. Σ'αυτό το παράδειγμα, η ευρετική λειτουργία είναι η μέθοδος Manhattan. Η μέθοδος αυτή υπολογίζει τη συνολική κάθετη και οριζόντια απόσταση ανάμεσα στο τρέχον κελί και την απόσταση, αγνοώντας τα εμπόδια και απορρίπτοντας κάθε διαγώνια κίνηση. Στην πραγματικότητα, αυτό το χρησιμοποιήσαμε για να εκτιμήσουμε τον κίνδυνο στην παράγραφο 3.2.1.

3.3.2 Επικίνδυνα πλέγματα

Η έννοια των επικίνδυνων πλεγμάτων έχει στόχο να προβλέπει τις θέσεις των UAVs και να ανακατασκευάζει το χάρτη σε ένα ευδιάκριτο χρονικό σύστημα. Στην εικόνα 3.6, το UAV κι ο εισβολέας πλησιάζουν το ένα το άλλο. Αυτά τα έγχρωμα πλέγματα δείχνουν ότι τα επιπλέον κόστη της ευρετικής λειτουργίας θα προσδιοριστούν αυτή τη χρονική στιγμή όταν τα έγχρωμα πλέγματα εξαπλωθούν απ'τον A*. Έτσι, ο A* μπορεί να κατασκευάσει τον χάρτη με τα δυναμικά εμπόδια και να αποφύγει αυτά τα πλέγματα με υψηλότερο κόστος.

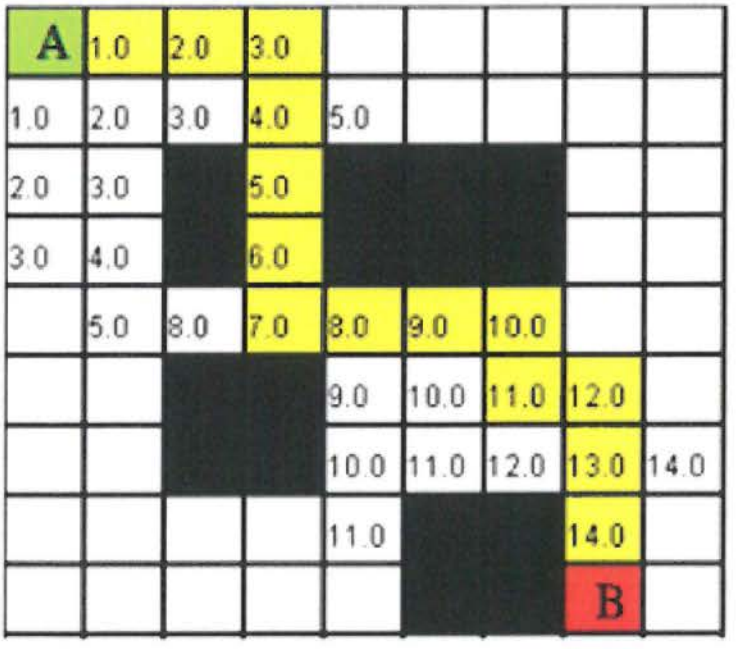


Εικόνα 3.6

3.3.3 Ευρετικές Λειτουργίες

Η ευρετική λειτουργία H είναι σημαντική για τον προσδιορισμό της εκτέλεσης του αλγορίθμου A*. Χρησιμοποιώντας διαφορετικές ευρετικές λειτουργίες, το αποτέλεσμα μπορεί να είναι τελείως διαφορετικό. Η εικόνα 3.7 δείχνει πως οι διαδρομές διαφέρουν χρησιμοποιώντας δυο διαφορετικές ευρετικές λειτουργίες.

Εικόνα 3.7 (α)



Εικόνα 3.7 (β)

Για να εφαρμοστεί ο αλγόριθμος A* στο πρόβλημα σχεδιασμού διαδρομής των UAVs, κάνουμε μια τροποποίηση.

- Πρώτον, κανονικά δεν θα πρέπει να υπάρχουν εμπόδια στον ουρανό έτσι ώστε κάθε πλέγμα να είναι θεωρητικά βατό στο πρόβλημά μας. Αλλιώς έχουμε άλλα αεροσκάφη στον αέρα ως κινούμενα εμπόδια.

- Δεύτερον, εκτός από τη μέθοδο Manhattan, η ευρετική λειτουργία πρέπει να βελτιωθεί για να υπολογίζει τα κόστη στις διαδρομές που βρίσκονται κοντά στους εισβολείς. Γι'αυτό, σ'αυτή τη μελέτη, η εκτίμηση του κόστους περιλαμβάνει την απόσταση Manhattan και την ευαισθητοποίηση του εισβολέα. Αυτή η μελέτη εισάγει δυο διαφορετικά είδη ευρετικών λειτουργιών.

  - Το πρώτο είναι οι συντηρητικές ευρετικές : σέβεται τους εναέριους κανόνες για την εξασφάλιση της ασφάλειας του αεροσκάφους.
  - Το δεύτερο είναι επιθετικές ευρετικές : παίρνει το ρίσκο να περάσει ανάμεσα απ'τους εισβολείς για να φτάσει πιο γρήγορα στον προορισμό.

Και οι δυο λειτουργίες βασίζονται στο συνδιασμό της μεθόδου Manhattan και της ευαισθητοποίησης του εισβολές.

Στη συντηρητική λειτουργία, όταν π.χ. μια σύγκρουση ανιχνεύεται, η διαδρομή προσέγγισης των εισβολέων θα κοστίσει περισσότερο απ'τη διαδρομή απομάκρυσνης απ'τους εισβολείς. Στην περίπτωση των ευαισθητοποιημένων εισβολέων, χρησιμοποιήσαμε μια μαθηματική μέθοδο για να κρίνουμε εάν ένας εισβολέας πλησιάζει ή απομακρύνεται και από ποια κατεύθυνση προέρχεται. Τώρα, μπορούμε να εφαρμόσουμε αυτή τη μέθοδο στην συντηρητική λειτουργία:

$$H(t) = \sum_{i=1}^{n} \alpha_1(|X_0 - X_g| + |Y_0 - Y_g|) + \beta_1(\lambda - |(X_i - X_0)| + |(Y_i - Y_0)|)$$

Όπου $(X_0\text{-}X_\alpha)+(Y_0\text{-}Y_\alpha)$ είναι η μέθοδος Manhattan, οι α, β και λ είναι σταθερές. Το τρέχον πλέγμα είναι τα X_0, Y_0 και η απόσταση είναι τα X_α, Y_α. Τέλος, τα X1, Y1 είναι ο εισβολέας.

Εάν ο εισβολέας πλησιάζει, δίνουμε μεγαλύτερη βαρύτητα στα πλέγματα που κινούνται προς τον εισβολέα αυξάνοντας την τιμή των α,β τότε το κόστος είναι μεγαλύτερο. Αλλιώς, εάν ο εισβολέας πλησιάζει αλλά το τρέχον πλέγμα απομακρύνεται απ'αυτόν, δίνουμε μικρότερη βαρύτητα. Άλλες συνθήκες επικρατούν διαμορφώνοντας κατάλληλα τα α, β. Ανεξάρτητα της κατάστασης, το λ παραμένει το ίδιο έτσι ώστε όταν ο εισβολέας πλησιάζει, όπου $|(X_1-X_0)|+|(Y_1-Y_0)|$ μικραίνει, το συνολικό κόστος μεγαλώνει. Η τιμή του α θα είναι μεγαλύτερη απ'του β διότι η μέθοδος Manhattan συμβάλλει στο εναπομείναν μήκος της διαδρομής, και θα το σταθμίσουμε για να επιβεβαιώσουμε ότι ο A* κινείται προς τον προορισμό. Αλλιώς, η διαδρομή μπορεί να παγιδευτεί όταν υπάρχουν τόσοι πολλοί εισβολείς στο δρόμο για τον προορισμό.

Στην επιθετική λειτουργία, προσπαθούμε να βρούμε ποια διαδρομή είναι η πιο αποτελεσματική και ίσως λιγότερο επικίνδυνη. Γι'αυτό λαμβάνουμε υπόψη την απόσταση ανάμεσα στα UAVs για να διασφαλίσουμε την ελάχιστη ασφάλεια.

$$H(t) = 2 * fieldsize - \sum_{i=1}^{n} |(X_i - X_0)| + |(Y_i - Y_0)| + (|X_0 - X_g| + |Y_0 - Y_g|)$$

όπου X_0 και Y_0 είναι η θέση των UAVs,

X_1 και Y_1 οι θέσεις των εισβολέων.

Ομοίως, όταν η απόσταση ανάμεσα στα UAVs μικραίνει, το άθροισμα της εξίσωσης μικραίνει και το κόστος λειτουργίας μεγαλώνει. Το κόστος της λειτουργίας αυξάνεται όταν η τιμή των ευαισθητοποιημένων εισβολέων μειώνεται. Γι'αυτό, οι ευαισθητοποιημένοι εισβολείς παρέχουν πληροφορίες για το πόσο μακριά είναι οι εισβολείς και για το αν πλησιάζουν ή απομακρύνονται.

Σημειωτέον ότι για την συντηρητική λειτουργία, λαμβάνουμε υπόψη κάθε εισβολέα ανεξάρτητα και υπολογίζουμε το κόστος που αντιστοιχεί σε κάθε εισβολέα. Τότε αθροίζουμε τα κόστη. Μ'αυτό τον τρόπο, αντιλαμβανόμαστε τη θέση και την κίνηση κάθε εισβολέα.

ΚΕΦΑΛΑΙΟ 4

ΠΡΟΣΟΜΟΙΩΣΗ

Σ'αυτό το κεφάλαιο θα εξετάσουμε την εκτέλεση των 2 λειτουργιών (συντηρητική και επιθετική) του αλγορίθμου A*. Θα εξετάσουμε τα αποτελέσματα της προσομοίωσης και τις επιπτώσεις χρησιμοποιώντας διάφορες ευρετικές λειτουργίες.

4.1 Η έναρξη της προσομοίωσης

a) Παρέκκλιση= $\frac{\text{Πραγματική απόσταση}}{\text{Ελάχιστη Εφικτή Απόσταση}}$

Η ελάχιστη εφικτή απόσταση είναι το μήκος της πιο μικρής διαδρομής που τα UAVs μπορούν να ταξιδέψουν όταν δεν υπάρχει εμπόδιο στο δρόμο τους. Με άλλα λόγια, η πιο μικρή διαδρομή είναι η ευθεία ανάμεσα στο σημείο έναρξης και του προορισμού. Η πραγματική απόσταση είναι το συνολικό μήκος της διαδρομής που ο A* προτείνει για την αποφυγή εμποδίων στον αέρα.

b) Βαθμός συγκρούσεων είναι ο αριθμός των ζευγαριών των αεροπλάνων εντός 6 πλεγμάτων. Μετράμε τις συγκρούσεις για να δούμε για πόσο τα UAVs βρίσκονται στην απόσταση σύγκρουσης. Γι'αυτό, εάν τα δυο UAVs εξακολουθούν να πετάνε μέσα σε 6 πλέγματα, ο αριθμός των συγκρούσεων θα εξακολουθεί να υπολογίζεται μέχρι η σύγκρουση να τελειώσει.

c) Αριθμός Συγκρούσεων. Στην προσομοίωσή μας λέμε οτι υπάρχει σύγκρουση όταν 2 ή περισσότερες διαδρομές των UAVs βρίσκονται στο ίδιο πλέγμα την ίδια χρονική στιγμή. Όταν συμβαίνει μια σύγκρουση, τα UAVs που συμμετέχουν σ'αυτή, εξαλείφονται και η προσομοίωση θα συνεχιστεί με τα εναπομείναντα UAVs.

Υποθέτουμε ότι ένα UAV χρειάζεται ένα δευτερόλεπτο για να διασχίσει ένα πλέγμα 5m*5m. Μια σύγκρουση συμβαίνει όταν 2 UAVs βρίσκονται το ένα απ'το άλλο σε απόσταση 6 δευτερολέπτων. Γι'αυτό έχουμε τουλάχιστον 3 δευτερόλεπτα για να αποφύγουμε μια σύγκρουση.

Για να εκτιμήσουμε τον A*, λαμβάνουμε υπόψη διάφορες διαμορφώσεις. Μια διαμόρφωση αποτελείται από έναν αριθμό UAVs (2,4,8 ή 16) και το μέγεθος του πεδίου 500*500 και 1000*1000. Τυχαία δημιουργούμε 100 διαφορετικά σενάρια. Κάθε σενάριο αποτελείται από 4 τυχαία επιλεγμένα σημεία για κάθε UAV. Κάθε UAV πρέπει να επισκευθεί αυτά τα 4 σημεία σε μια προκαθορισμένη σειρά. Για να δημιουργήσουμε τυχαία σημεία, επιλέγουμε ένα τυχαίο σημείο στο πεδίο ως σημείο έναρξης, τότε χρησιμοποιούμε μια συγκεκριμένη ακτίνα 50 μέτρων και 7 τυχαίες κατευθύνσεις για να δημιουργήσουμε το επόμενο σημείο. Μ'αυτό τον τρόπο εξασφαλίζουμε ότι κάθε UAV θα ταξιδέψει τουλάχιστον 200 μέτρα στις προσομοιώσεις.

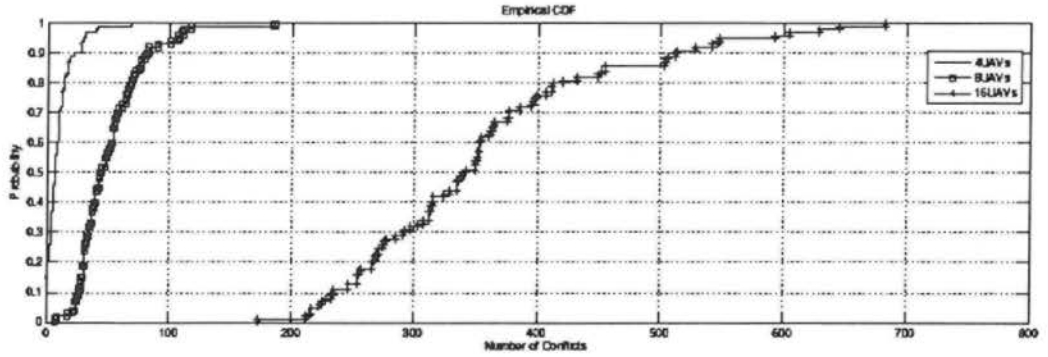
Με βάση το ίδιο σενάριο, το εξομοιώνουμε σε διαφορετική διαμόρφωση για να εκτιμήσουμε τον αλγόριθμο A*. Συγκρίνουμε 2 ευρετικές λειτουργίες: την συντηρητική και την επιθετική. Η συντηρητική λειτουργία θέτει επιπλέον κόστος όταν η διαδρομή του UAV αποκόπτει τους εισβολείς έτσι ώστε αναμένουμε ότι η διαδρομή θα είναι περιστροφική ώστε να κρατήσει τους εισβολείς σε απόσταση. Η επιθετική λειτουργία λαμβάνει υπόψη μόνο το μήκος της διαδρομής και της θέσης των εισβολέων, έτσι αναμένουμε ότι θα βρει μια διαδρομή γύρω από τους εισβολείς έτσι ώστε να φτάσει γρηγορότερα στον προορισμό.

4.2 Προσομοίωση σε πεδίο 500*500

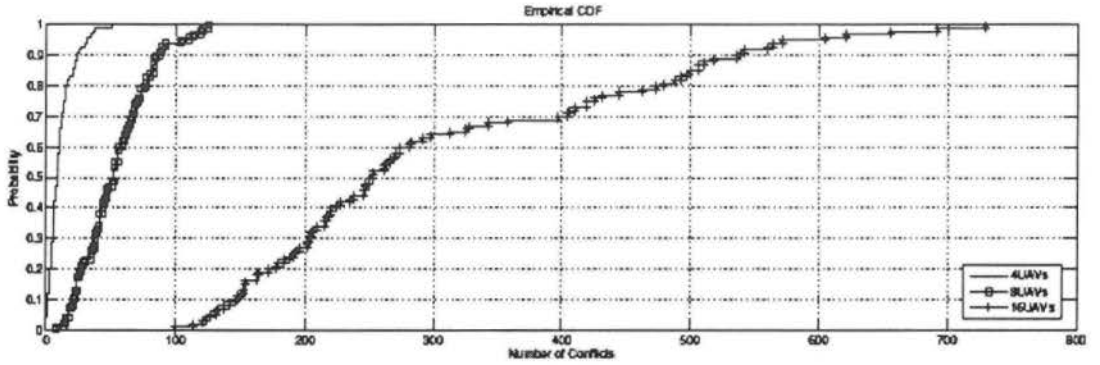
Όπως φαίνεται στην εικόνα 4.1 :

a. Η αθροιστική συνάρτηση διανομής περιγράφει την πιθανότητα κατά την οποία ο αριθμός συγκρούσεων ισοδυναμεί ή είναι μικρότερος από μια συγκεκριμένη τιμή. Για παράδειγμα, εάν θεωρήσουμε μια ευθεία γραμμή όπου ο αριθμός συγκρούσεων ισοδυναμεί με 100, μπορούμε να πούμε ότι η πιθανότητα των 100 συγκρούσεων ή λιγότερων είναι 100% για 4 UAVs, περίπου 96% για 8 UAVs και μόνο 1% για 16 UAVs.

b. Η αθροιστική συνάρτηση διανομής είναι όμοια για 4 UAVs και 8 UAVs. Σύμφωνα με αυτή την παρατήρηση, θεωρούμε ότι οι δυο αυτές ευρετικές συναρτήσεις έχουν σχεδόν τις ίδιες δυνατότητες αποφυγής των συγκρούσεων όταν ο αριθμός των UAVs είναι μικρότερος από 8. Στην εκδοχή των 16 UAVs, παρατηρούμε ότι η επιθετική συνάρτηση συναντά περισσότερες συγκρούσεις απ'την συντηρητική εάν συγκρίνουμε την αθροιστική συνάρτηση διανομής όταν ο αριθμός των συγκρούσεων ισοδυναμεί με 200.

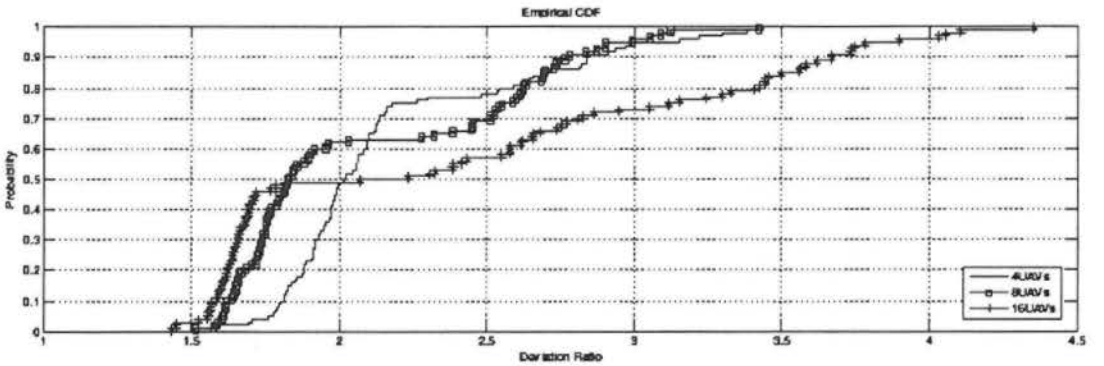


Εικόνα 4.1 (α)

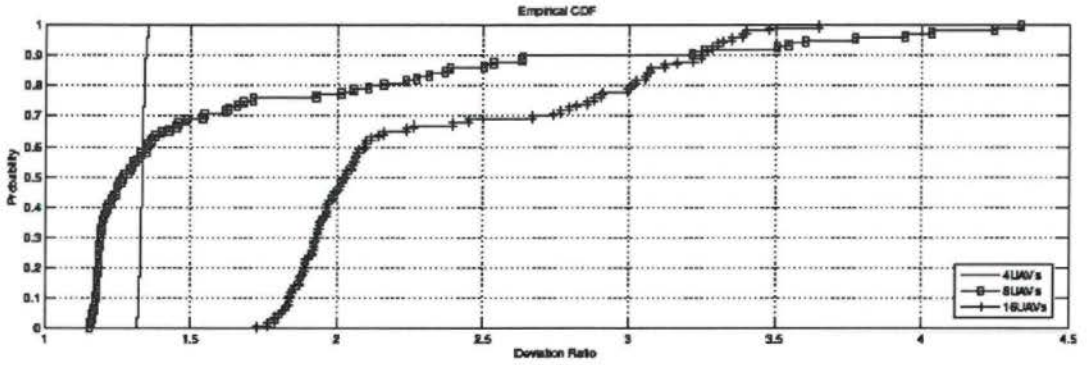


Εικόνα 4.1 (β)

Κατά τις προσδοκίες μας, η αναλογία απόκλισης της επιθετικής συνάρτησης θα πρέπει να είναι μικρότερη από αυτή της συντηρητικής, δεδομένου ότι θα βρει μια πιο σύντομη διαδρομή. Στην εικόνα 4.2, μπορούμε να δούμε ότι οι αναλογίες απόκλισης της επιθετικής συνάρτησης είναι περίπου 1,5 φορά μικρότερες από αυτές της συντηρητικής όταν πετούν 4 ή 8 UAVs. Για την περίπτωση των 16 UAVs, όταν η πιθανότητα φτάνει το 99%, η αναλογία απόκλισης ισοδυναμεί με 3,5 όταν χρησιμοποιείται η επιθετική συνάρτηση και 4,1 όταν χρησιμοποιείται η συντηρητική. Κάνοντας την σύγκριση ανάμεσα σε 8 UAVs που πετάνε και σε 16 UAVs, όταν πετάνε UAVs σε πεδίο με μεγαλύτερη πυκνότητα, το πλεονέκτημα της επιθετικής συνάρτησης με μικρότερη αναλογία απόκλισης καθιστάται αφανές.

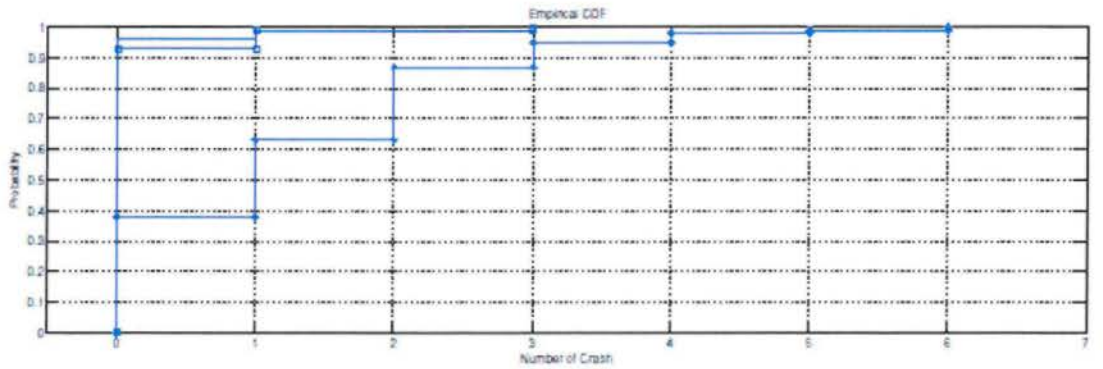


Εικόνα 4.2 (α)

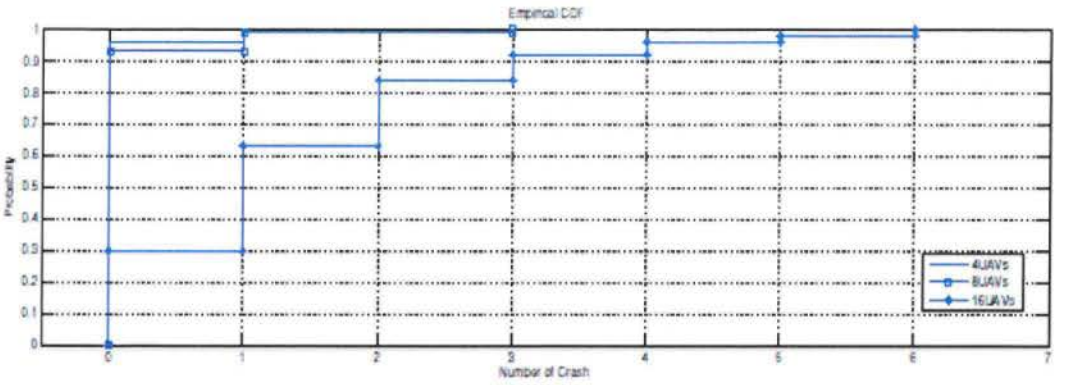


Εικόνα 4.2 (β)

Αν και η επιθετική συνάρτηση έχει μεγαλύτερη πιθανότητα να συναντήσει συγκρούσεις, συνήθως έχει μεγαλύτερη ικανότητα να βρει μια πιο σύντομη διαδρομή. Η επιθετική ευρετική μπορεί να μειώσει τις συγκρούσεις μειώνοντας τη διάρκεια της πτήσης των UAVs. Επίσης παρατηρούμε στην εικόνα 4.2 ότι ο μέσος όρος του ποσοστού απόκλισης της επιθετικής ευρετικής είναι μικρότερος σ'αυτή τη διαμόρφωση. Όμως, στην περίπτωση της πτήσης 8 UAVs, η μέγιστη αναλογία απόκλισης της επιθετικής ευρετικής είναι σχεδόν 1,5 φορά η αναλογία της συντηρητικής. Αυτό σημαίνει ότι σε ορισμένες ακραίες περιπτώσεις η διαδρομή της επιθετικής ευρετικής απ'τη διαδρομή της συντηρητικής. Θα αναλύσουμε περαιτέρω τους λόγους που προκαλούν αυτό το φαινόμενο παρακάτω.



Εικόνα 4.3 (α)

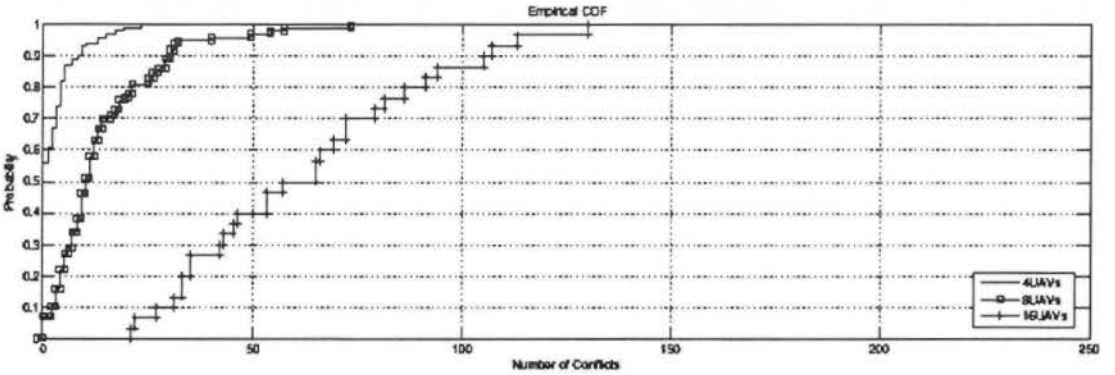


Εικόνα 4.3 (β)

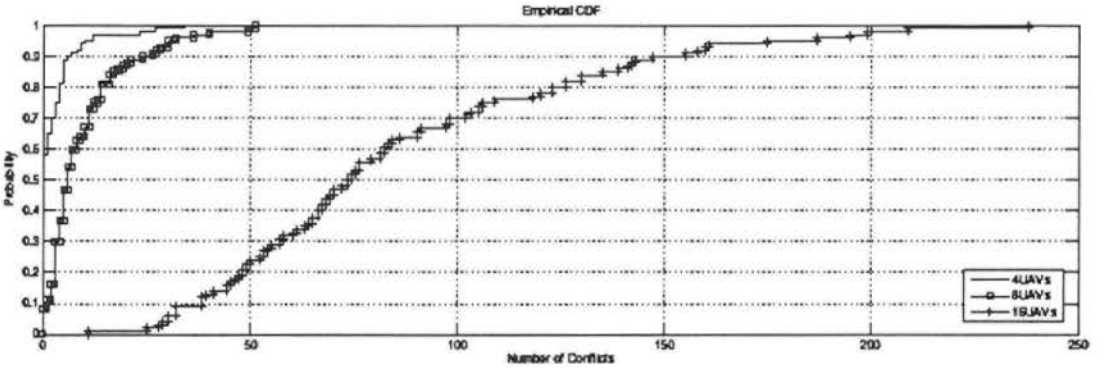
Η εικόνα 4.3 δείχνει ότι η ικανότητα αποφυγής της σύγκρουσης είναι όμοια με την αποφυγή των CONTLICB. Όταν ο αριθμός των UAVs είναι μικρότερος από 8, οι συγκρούσεις είναι λιγότερες από 4 σε 100 προσομοιώσεις. Πραγματικά, περισσότερες απ'τις μισές προσομοιώσεις δεν καταλήγουν σε σύγκρουση. Όμως, όταν ο αριθμός των UAVs είναι μεγαλύτερος από 8, η σύγκρουση αυξάνεται σταδιακά. Στην χειρότερη περίπτωση «επιζούν» μόνο 3 UAVs. Οι παρουσιάσεις των συντηρητικών και των επιθετικών ευρετικών είναι όμοιες όταν ο αριθμός των UAVs είναι μικρότερος του 8. Εάν ο αριθμός των UAVs είναι μεγαλύτερος του 8, η επιθετική τείνει να προκαλέσει περισσότερες συγκρούσεις σε αυτή τη διαμόρφωση.

4.3 Προσομοίωση σε πεδίο 1000*1000

Εδώ χρησιμοποιούμε ένα τετράγωνο πεδίο με πλέγματα 200*200 για να χρησιμοποιήσουμε την πτήση 2, 4, 8 ή 16 UAVs ταυτόχρονα. Συνεπώς, η παρουσίαση της αποφυγής συγκρούσεων είναι καλύτερη απ'την προσομοίωση σε ένα πεδίο 500*500 , κυρίως όταν ο αριθμός των UAVs είναι 16 όπως φαίνεται στην εικόνα 4.4. Όταν πετούν 16 UAVs, οι συγκρούσεις μειώνονται σημαντικά σε αυτή τη διαμόρφωση και δεν έχει σημασία ποια ευρετική χρησιμοποιούμε διότι παρέχει στα UAVs περισσότερο χώρο για να αποφύγουν τις συγκρούσεις.

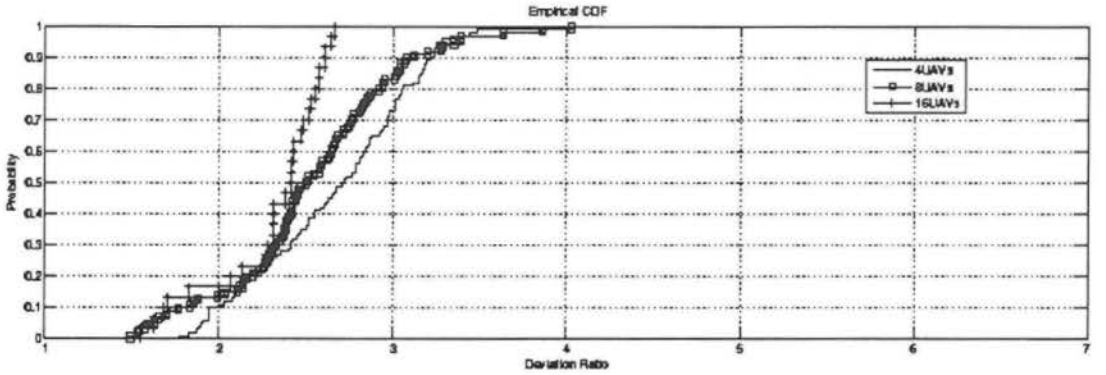


Εικόνα 4.4 (α)

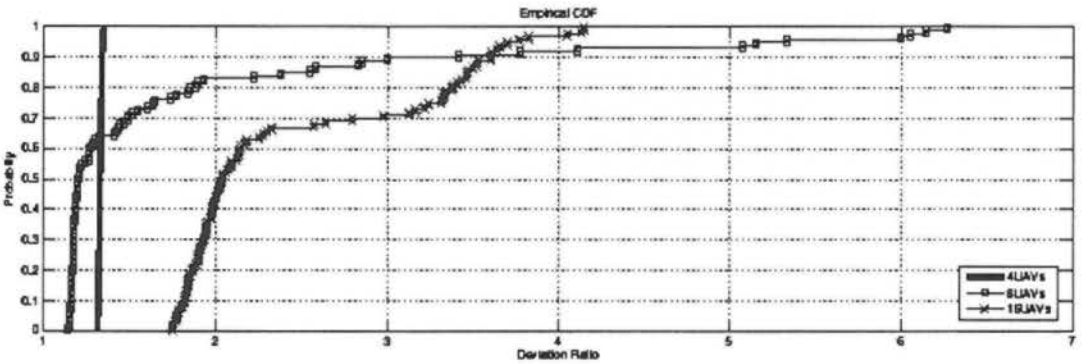


Εικόνα 4.4 (β)

Η παρουσίαση της αναλογίας απόκλισης είναι επίσης καλύτερη στην προσομοίωση σε ένα πέδιο 500*500 όπως φαίνεται στην εικόνα 4.5. Εκτός από την περίπτωση των 8 UAVs χρησιμοποιούν επιθετική ευρετική κατά την εκτίμησή μας, η επιθετική θα πρότεινε μια διαδρομή με μικρότερη αναλογία απόκλισης. Σημειωτέον ότι στην προσομοίωση του πεδίου 500*500 η αναλογία απόκλισης της επιθετικής έχει απροσδόκητη συμπεριφορά στην περίπτωση του ενός UAVs. Αυτό το φαινόμενο προκαλεί ένα ερώτημα: Γιατί αυτό συμβαίνει μόνο στην περίπτωση των 8 UAVs; Για να απαντήσουμε σε αυτό, πρέπει να επανεξετάσουμε την σχέση ανάμεσα στους αριθμούς των UAVs και στις μετρήσεις αξιολόγησης.

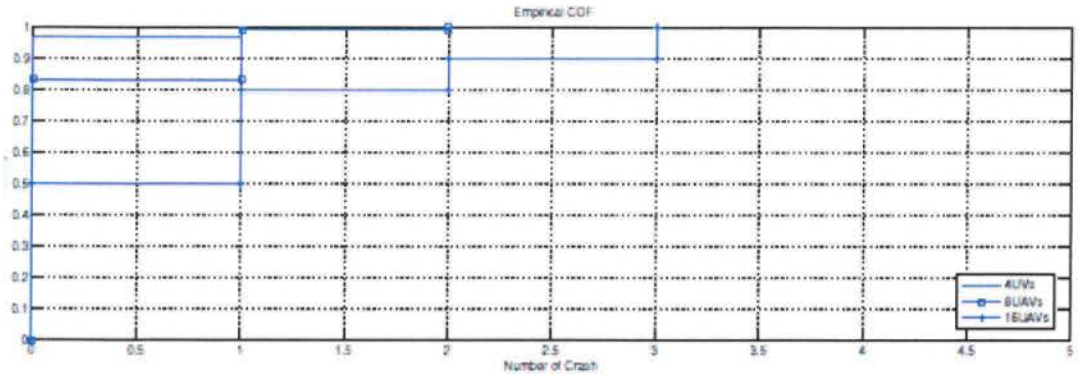


Εικόνα 4.5 (α)

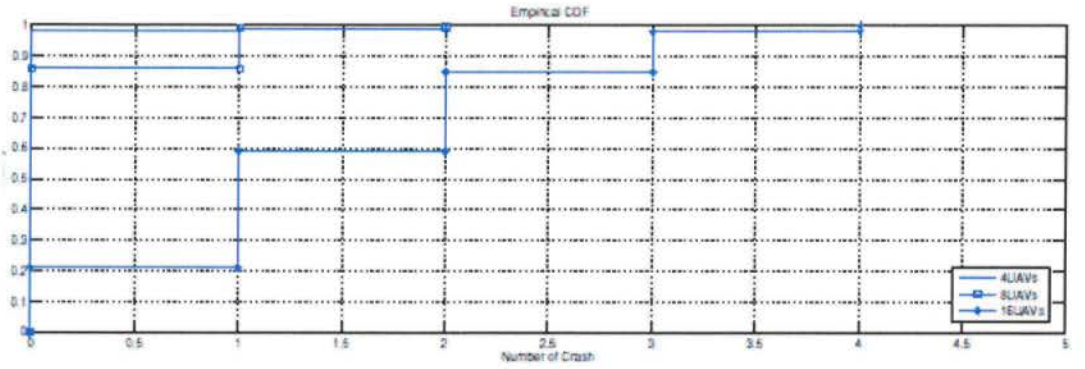


Εικόνα 4.5 (β)

Ο αριθμός των συγκρούσεων σε ένα πεδίο 1000*1000 δεν αλλάζει πολύ στην περίπτωση των 4 και 8 UAVs. Όμως, στην περίπτωση των 16 UAVs, και οι δύο ευρετικές επιτυγχάνουν έναν πολύ μικρότερο αριθμό συγκρούσεων όπως φαίνεται στην εικόνα 4.6 διότι τα UAVs μπορούν ν'αποφύγουν το ένα το άλλο πιο εύκολα σ'ένα μεγαλύτερο πεδίο. Επίσης, δεν είναι φανερό αλλά οι συγκρούσεις των επιθετικών ευρετικών είναι ελαφρώς περισσότερες απ'αυτές των συντηρητικών.



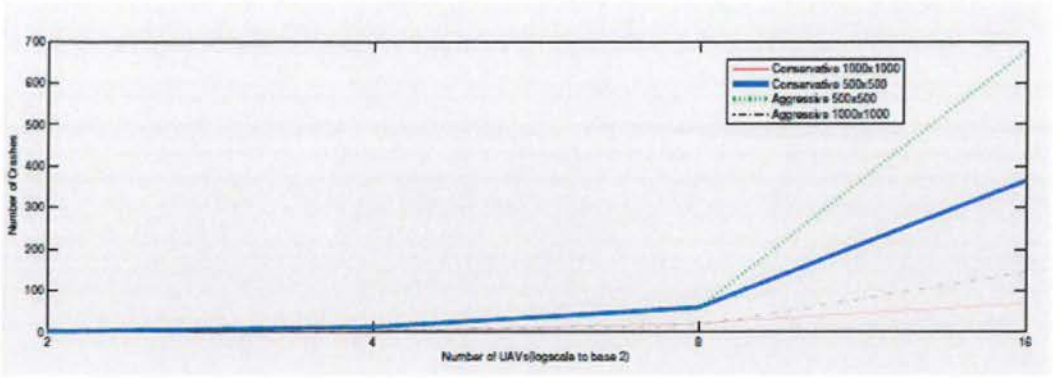
Εικόνα 4.6 (α)



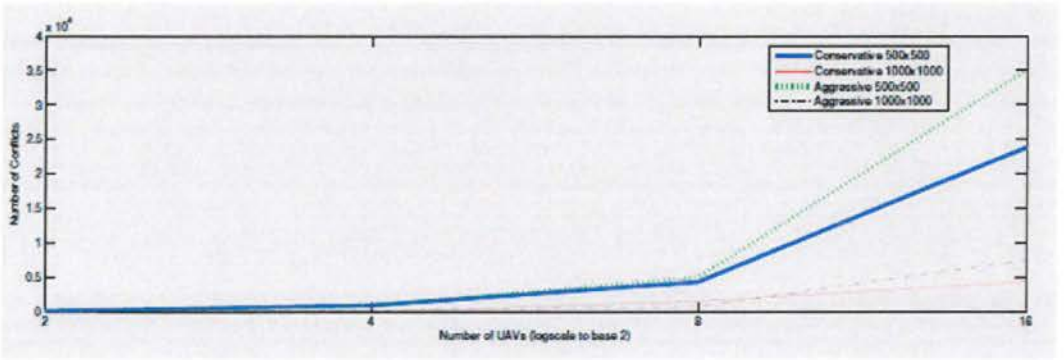
Εικόνα 4.6 (β)

4.4 Σχέση μεταξύ του αριθμού των UAVs και των μετρήσεων

Στην περίπτωση των 8 UAVs, είδαμε ότι οι επιθετικές ευρετικές μπορούν να αποτύχουν σε μερικές ακραίες περιπτώσεις. Για να επιβεβαιώσουμε την σκέψη μας, επανεξετάζουμε την σχέση μεταξύ του αριθμού των UAVs και των μετρήσεων για να συγκρίνουμε αυτές τις δύο ευρετικές όπως φαίνεται στις εικόνες 4.7 και 4.8.



Εικόνα 4.7

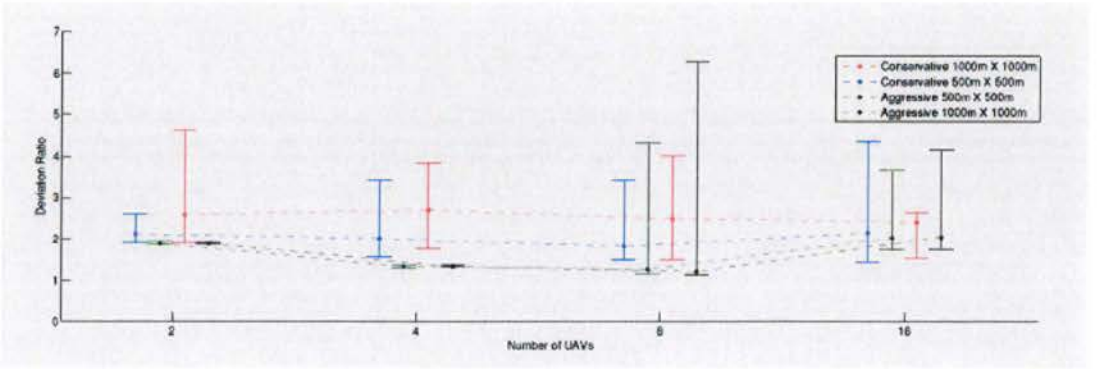


Εικόνα 4.8

Όταν ο αριθμός των UAVs είναι μεγαλύτερος από 8, ο αριθμός των συγκρούσεων και των ατυχημάτων που χρησιμοποιούν την επιθετική ευρετική αυξάνεται αρκετά. Χωρίς να έχει σημασία ποιο είναι το μέγεθος του πεδίου, παρατηρούμε οτι υπάρχουν περισσότερα ατυχήματα και συγκρούσεις χρησιμοποιώντας την επιθετική ευρετική. Ωστόσο, αυτό δεν απαντά ακόμα στο γιατί οι ακραίες περιπτώσεις συμβαίνουν μόνο στην ακραία περίπτωση των 8 UAVs. Πρέπει να επανεξετάσουμε τη σχέση μεταξύ του αριθμού των UAVs και της αναλογίας απόκλισης. Η αναλογία απόκλισης αλλάζει όταν ο αριθμός των UAVs αυξάνεται όπως φαίνεται στην εικόνα 4.9. Οι αναλογίες των συντηρητικών δεν διαφέρουν σημαντικά και οι αναλογίες των επιθετικών έχουν την τάση να μειώσουν τα UAVs από 2 σε 8. Ο λόγος μπορεί να είναι ότι ο A* υπεραντιδρά με τον εισβολέα διότι η επιθετική θέτει υψηλότερο κόστος στη διαδρομή που πλησιάζει τους εισβολείς.

Όταν υπάρχει μόνο ένας εισβολέας, μόνο η κατεύθυνση προς τον ειβολέα έχει υψηλότερο κόστος έτσι ώστε το UAV να μπορεί να αποκλίνει τόσο πολύ. Όταν υπάρχουν περισσότεροι εισβολείς από κατευθύνσεις, η αναλογία απόκλισης μικραίνει επειδή ο A* δεν υπεραντιδρά σε μια συγκεκριμένη κατεύθυνση.

Η εικόνα 4.9 επίσης αναπαριστά τη μέγιστη και την ελάχιστη αναλογία απόκλισης σε 100 προσομοιώσεις. Βλέπουμε ότι η αναλογία απόκλισης της επιθετικής ευρετικής δεν αλλάζει πολύ για 2 ή 4 UAVs. Περιπτώσεις υψηλότερης αναλογίας απόκλισης έχουμε σε ακραίες περιπτώσεις. Γι'αυτό, οι ακραίες περιπτώσεις με 2 ή 4 UAVs μπορεί να μην έχουν σημαντικές επιρροές στην επιθετική ευρετική γιατί μπορούν καλύτερα να βρουν διαδρομή όταν το πεδίο είναι αραίο.



Εικόνα 4.9

Όταν ο αριθμός των UAVs είναι μικρότερος από 8, η επιθετική ευρετική πάντα έχει μικρότερη αναλογία απόκλισης. Ωστόσο, στην περίπτωση των 16 UAVs, η αναλογία της επιθετικής ευρετικής πλησιάζει την αναλογία της συντηρητικής σε ένα 500*500 ή σε ένα 1000*1000 πεδίο. Συνεπώς, η επιθετική ευρετική χάνει την ανταγωνιστική της στην περίπτωση των 16 UAVs. Το ερώτημα είναι γιατί συμβαίνει αυτό;

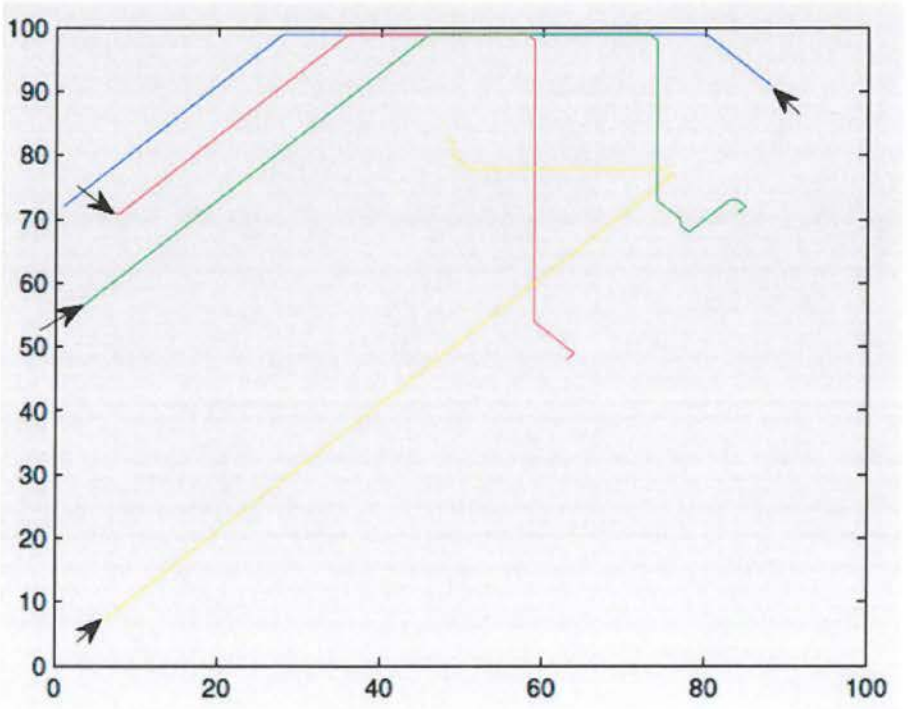
Πριν απαντήσουμε σε αυτό το ερώτημα, πρέπει να συζητήσουμε το προηγούμενο θέμα για τις ακραίες περιπτώσεις των 8 UAVs. Τα 8 UAVs αποτελούν οριακό σημείο (εικόνα 4.9) ενώ η αναλογία απόκλισης της επιθετικής ευρετικής αυξάνεται. Έτσι, οι ακραίες περιπτώσεις των 8 UAVs μπορούν να αποτελέσουν κακό οιωνό της επιθετικής ευρετικής στις περιπτώσεις των 16 UAVs. Ο λόγος εμφάνισης της υψηλής αναλογίας απόκλισης στις περιπτώσεις των 8 UAVs συνδέεται με τον αριθμό των συγκρούσεων. Όπως φαίνεται στην εικόνα 4.8, οι συγκρούσεις είναι πολύ περισσότερες στις περιπτώσεις των 16 UAVs έτσι ώστε σ'αυτές τις περιπτώσεις τα UAVs δεν μπορούν να αποφύγουν την σύγκρουση αλλά το ατύχημα. Στην περίπτωση των 8 UAVs, τα UAVs έχουν αρκετό χώρο για να αποφεύγουν το ένα το άλλο ακόμη και με μεγαλύτερη απόκλιση σε κάποιες περιπτώσεις.

Η επιθετική ευρετική επιτυγχάνεται στις περιπτώσεις των 16 UAVs επειδή δεν υπάρχει αρκετός χώρος για την αποφυγή της σύγκρουσης. Να σημειωθεί ότι η επιθετική αφορά μόνο τις θέσεις των εισβολέων, δεν έχει τη στρατηγική όπως κανόνες στον αέρα για να τους αποφύγει. Γι'αυτό, όταν υπάρχουν πολλά UAVs στον αέρα, η επιθετική ευρετική χάνει την ικανότητά της να σχεδιάζει καλύτερες διαδρομές είτε με αποφυγή των συγκρούσεων, είτε με κατεύθυνση προς τον προορισμό. Όχι μόνο καταλήγει σε περισσότερες συγκρούσεις και ατυχήματα. αλλά επίσης αυξάνει την αναλογία απόκλισης.

4.5 Ακραίες περιπτώσεις για την αποφυγή σύγκρουσης

Κατόπιν μελέτης πολλών προσομοιώσεων των UAVs, βρήκαμε μερικές ακραίες περιπτώσεις κατά τις οποίες τα UAVs μπορεί να δυσκολευτούν να βρουν μια βατή διαδρομή. Ερευνώντας αυτές τις περιπτώσεις, μπορούμε να βρούμε μερικά όρια και περιορισμούς για τις μετρήσεις αξιολόγησης των UAVs.

Μια απ'τις δύσκολες καταστάσεις για την αποφυγή της σύγκρουσης είναι όταν τα UAVs πετούν σε κατάλληλες γραμμές. Σ'αυτήν την περίπτωση, η διαδρομή του UAV στη μέση θα μπλοκαριστεί απ'τα άλλα δυο UAVs απ'τις δυο πλευρές. Γενικά, ο αλγόριθμος A* μπορεί να χειριστεί αυτήν την κατάσταση θέτοντας διαφορετικά κόστη στην δεξιά και την αριστερή στροφή. Ωστόσο, στο παράδειγμα της εικόνας 4.10, υπάρχουν 3 UAVs παράλληλα το ένα με το άλλο στην αρχή. Επίσης, η αρχική θέση των UAVs στην κορυφή είναι κοντά στο όριο του πεδίου, έτσι ώστε να υπάρχει αρκετός χώρος για να αποφύγει το ένα το άλλο. Επιπλέον, το UAV απ'τα δεξιά κινείται επίσης στην ίδια γραμμή ώστε να αποφύγει το UAV στο κάτω μέρος. Παρόλο που τα UAVs εισέρχονται στην ίδια γραμμή στην κορυφή σε διαφορετικές χρονικές στιγμές, μερικά απ'αυτά επιτυγχάνουν να βγουν απ'τη γραμμή εγκαίρως και τελικά συμβαίνει η σύγκρουση.

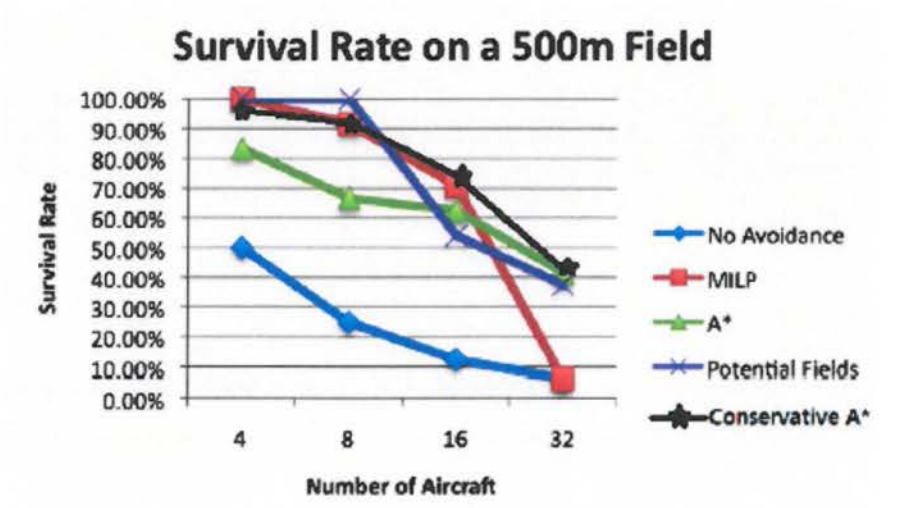


Εικόνα 4.10

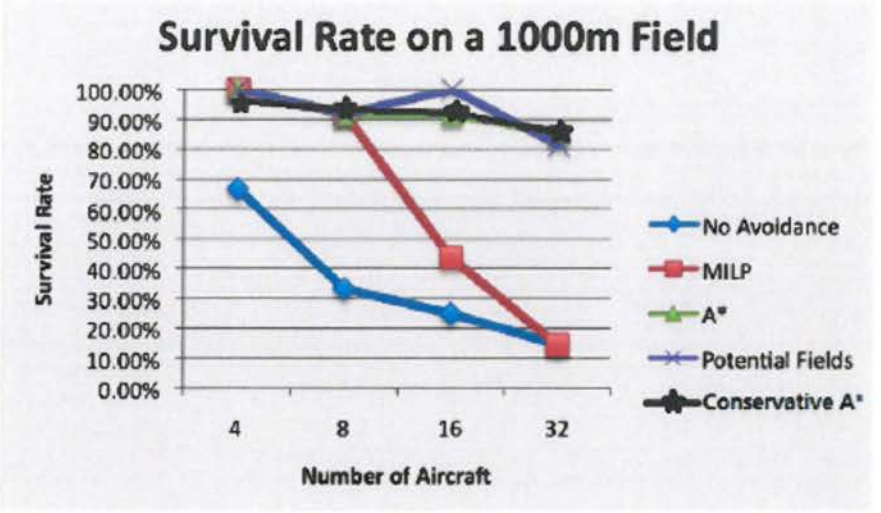
4.6 Σύγκριση με άλλους αλγορίθμους

Σ'αυτό το κεφάλαιο, παρουσιάσαμε προσομοιώσεις με διαφορετικούς αριθμούς UAVs και μεγέθη πεδίων.

Στις εικόνες 4.11 και 4.12, ο συντηριτικός A* είναι ο αλγόριθμος που χρησιμοποιήσαμε σ'αυτή την μελέτη. Η αναλογία επιβίωσης της προσέγγισής μας μπορεί να υπολογιστεί απ'την εικόνα 4.8 όπως φαίνεται στην εικόνα 4.13. Λαμβάνοντας υπόψη το ρυθμό επιβίωσης, η συντηριτική ευρετική έχει καλύτερο αποτέλεσμα στις περισσότερες περιπτώσεις. Συγκριτικά, για τις περιπτώσεις των 16 UAVs στο πεδίο 500m*500m, ο ρυθμός επιβίωσης στην συντηριτική είναι ελαφρώς υψηλότερος απ' το MILP (Mixed Integer Linear Programming) που είναι το καλύτερο αποτέλεσμα (εικόνα 4.11). Για τις περιπτώσεις στο πεδίο 1000m*1000m, οι ρυθμοί επιβίωσης και των δυο ευρετικών είναι μεγαλύτεροι του 90% που είναι όμοιοι με την παρουσίαση του A* (εικόνα 4.12) .

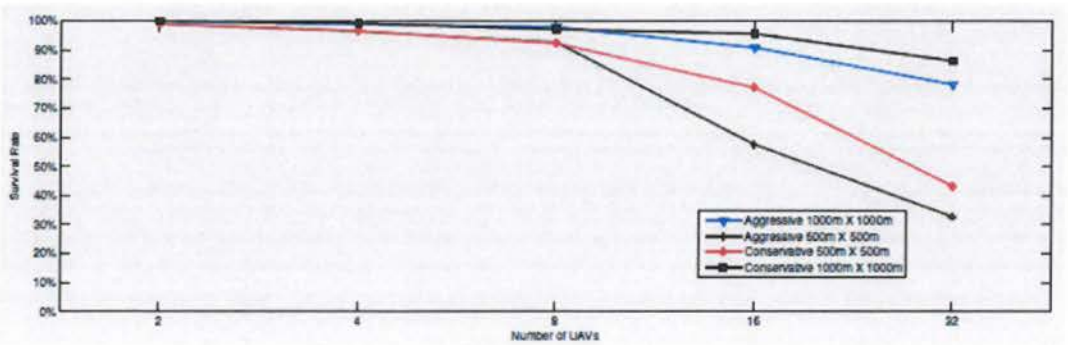


Εικόνα 4.11



Εικόνα 4.12

Επίσης και το MILP και το APF (Artificial Potential Field) έχουν φανερά πλεονεκτήματα και μειονεκτήματα. Για παράδειγμα, το APF έχει τον υψηλότερο βαθμό επιβίωσης στις περιπτώσεις πεδίου με χαμηλότερη πυκνότητα. Όμως, ο ρυθμός επιβίωσης πέφτει σημαντικά στην περίπτωση ενός 500m*500m πεδίου όταν ο αριθμός των UAVs είναι μεγαλύτερος των 16. Γι'αυτό, ακόμη και αν η προσέγγισή μας δεν έχει 100% ρυθμό επιβίωσης στις περισσότερες περιπτώσεις, είναι ακόμη ανταγωνιστική όταν ο αριθμός των UAVs είναι μικρότερος των 16 γιατί παρέχει σταθερό καλό ρυθμό επιβίωσης υπό πολλές διαμορφώσεις.



Εικόνα 4.13

ΚΕΦΑΛΑΙΟ 5

ΣΥΜΠΕΡΑΣΜΑ

Για να μπορέσουν τα UAVs να πετάνε αυτόνομα, είναι απαραίτητο ένα αξιόπιστο σύστημα αποφυγής της σύγκρουσης. Στο κεφάλαιο 2 είδαμε πολλές αντιπροσωπευτικές προσεγγίσεις για να επιλύσουμε το πρόβλημα αποφυγής της σύγκρουσης. Μερικές από αυτές έχουν αδυναμίες όπως υπολογιστικά ακριβό αλγόριθμο, ακριβή αισθητήρα, πρόσθετο εξοπλισμό. Τότε συμπεράναμε ότι ο A* είναι ευέλικτος επειδή επιτρέπει σε διάφορες ευρετικές συναρτήσεις να χειρίζονται διάφορες καταστάσεις έτσι ώστε να μπορούμε να τροποποιήσουμε τον αρχικό A* ώστε να αντιμετωπίσει δυναμικά εμπόδια. Γι'αυτό, ο A* είναι ένας καλός υποψήφιος για την παρουσίαση του πραγματικού χρόνου σχεδιασμού διαδρομής.

Ευρετικές Λειτουργίες

Η κυρίως στρατηγική μας των συντηρητικών ευρετικών είναι: «μην περνάς μπροστά από έναν εισβολέα». Αυτή η στρατηγική μπορεί τουλάχιστον να αποφύγει την επικείμενη προσέγγιση άλλων εισβολέων. Εάν λάβουμε υπόψη άλλες καταστάσεις συγκρούσεων, οι ευρετικές μπορούν πιθανώς να βελτιωθούν. Για παράδειγμα, μπορούμε να λάβουμε υπόψη την περίπτωση της παράλληλης πτήσης. Αντί να λύσουμε αυτή την περίπτωση, μπορούμε να προσπαθήσουμε να θέσουμε επιπλέον κόστη για να εμποδίσουμε την παράλληλη πτήση. Όμως, κάθε φορά που προσθέτουμε μια καινούρια ευρετική, πρέπει να προσαρμόσουμε την αναλογία κόστους ανάμεσα στο μήκος της διαδρομής (μέθοδος Manhattan) και την ασφάλεια (συντηρητική και επιθετική ευρετική). Για να διασφαλίσουμε την ασφάλεια και το μήκος της διαδρομής σε ένα αποδεκτό εύρος, επιλέγουμε κατάλληλη αναλογία η οποία είναι περίπου 5:1 για την συντηρητική ευρετική και 1:1 για την επιθετική.

Μελλοντικά Έργα

Σύμφωνα με το αποτέλεσμα αυτής της μελέτης, πιστεύουμε ότι χρησιμοποιώντας δυο εναλλακτικές ευρετικές συναρτήσεις στον αλγόριθμο A*, θα μπορούσαμε να επιτύχουμε την καλύτερη παρουσίαση. Εφόσον η επιθετική ευρετική δουλεύει καλά, όταν ο αιρθμός των UAVs είναι μικρότερος των 8, θα μπορούσαμε να την χρησιμοποιήσουμε για περιπτώσεις χαμηλότερης πυκνότητας. Όταν ο αριθμός των UAVs είναι μεγαλύτερος των 8, μπορούμε να την αλλάξουμε σε συντηρητική ευρετική για να εξασφαλίσουμε την ασφάλεια.

Οφέλη στη ρομποτική

Το αποτέλεσμα της προσομοίωσης δείχνει ότι η αποφυγή της σύγκρουσης χρησιμοποιώντας τον A* είναι αξιόπιστη όταν η πυκνότητα (αριθμός των UAVs πάνω απ'την περιοχή του πεδίου) είναι μικρότερη από 8 UAVs/1000000m^2. Η σχέση μεταξύ της πυκνότητας και της παρουσίασης του A*, μπορεί να είναι γραμμική διότι η πτήση περισσότερων UAVs την ίδια χρονική στιγμή απαιτεί πιο σύνθετους υπολογισμούς για τις διαδρομές τους. Αυτή η μελέτη ερεύνησε την εφαρμογή του αλγορίθμου A* για τον σχεδιασμό διαδρομής σε ένα δυναμικό περιβάλλον, κι αυτό είναι συνηθισμένο θέμα για τα κινητά ρομπότ. Γι'αυτό, η μελέτη του αλγορίθμου A* μπορεί να υιοθετηθεί και για άλλα κινητά ρομπότ.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Τεχνολογίες Υλοποίησης Αλγορίθμων https://www.ceid.upatras.gr/webpages/faculty/zaro/teaching/alg-eng/
2. Διαδικτυακός Τόπος της ομάδας Arduino http://arduino.cc/en/
3. Σύγκριση των αλγορίθμων IDA* και A* στο 15-puzzle http://www.icte.uowm.gr/uploads/thesis/manoliadhs_am55_diplomatikh.pdf
4. Artifical Intelligence Center PREVIEW http://www.ai.sri.com/shakey/
5. Αλγόριθμοι και Πολυπλοκότητα http://www.softlab.ntua.gr/~fotakis/algorithms.html
6. A* Pathfinding για Αρχάριους http://www.policyalmanac.org/games/aStarTutorial_greek.htm
7. Αλγόριθμοι Ευρετικής Αναζήτησης http://aibook.csd.auth.gr/include/slides/Chap04.pdf
8. http://en.wikipedia.org/wiki/A*_search_algorithm
9. B. Goertzel, Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms (Frontiers in Artificial Intelligence and Applications)
10. http://intelligence.worldofcomputing.net/ai-search/iterative-deepening-a-star.html#.UtW3LScyM24
11. Βλαχάβας, Κεφαλάς, Βασιλειάδης, Κόκκορας, Σακελλαρίου, Τεχνητή Νοημοσύνη - Γ' Έκδοση, Εκδόσεις Γκιούρδας
12. http://www.informit.com/articles/article.aspx?p=1881386&seqNum=2
13. http://www.apl.jhu.edu/~hall/AI-Programming/IDA-Star.html
14. Βλαχάβας, Κεφαλάς, Βασιλειάδης, Κόκκορας, Σακελλαρίου, Τεχνητή Νοημοσύνη - Γ' Έκδοση, Εκδόσεις Γκιούρδας
15. http://www.daniweb.com/software-development/cpp/threads/351851/thread-safe-timer-for-c-callback

16. Stuart Russel και Peter Norvig, Τεχνητή Νοημοσύνη, μια σύγχρονη προσέγγιση, εκδόσεις Κλειδάριθμος
17. http://el.wikipedia.org/wiki/%CE%A5%CF%80%CE%BF%CE%BB%CE%BF%CE%B3%CE%B9%CF%83%CF%84%CE%B9%CE%BA%CE%AE_%CE%BD%CE%BF%CE%B7%CE%BC%CE%BF%CF%83%CF%8D%CE%BD%CE%B7
18. http://www.faa.gov/documentLibrary/media/Advisory_Circular/TCAS%20II%20V7.1%20Intro%20booklet.pdf
19. Dalong Wang, Dikai Liu, and Gamini Dissanayake. A variable speed force eld method for multi-robot collaboration. In IROS, pages 2697{2702, 2006.

ΠΑΡΑΡΤΗΜΑ – ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΟΥ A*

- Main

```
package astar22;
import java.lang.*;
/*
 * To change this template, choose Tools | Templates and open the template in
 * the editor.
 */
/**
 *
 * @author giannis
 */
public class Main {
  public static void main(String args[]){
      Startup s=new Startup();
      s.Initialise();
    }
  }
```

- Map

```
package astar22;
import java.awt.*;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.Serializable;
public class Map extends java.awt.Panel implements Serializable
{
  int w;
  int h;
  transient Image buffer;
  GridCell gridCell[][];
        public Map(int dim, int dim1)
        {
                super();
         w=dim;
         h=dim1;
         gridCell=new GridCell[w][h];
                setLayout(new GridLayout(w,h));
                setSize(insets().left + insets().right + 430,insets().top + insets().bottom
+ 270);
                //}}
    for(int i=0;i<w;i++){
       for(int j=0;j<h;j++){
         gridCell[i][j] = new GridCell();
         gridCell[i][j].setPosition(new Point(i,j));
         add(gridCell[i][j]);
       }
```

```
    }
  }
  @Override
  public void paint(Graphics g){
    if(buffer == null){buffer = createImage(getBounds().width,getBounds().height);}
    Graphics bg = buffer.getGraphics();
    super.paint(bg);
    bg.setColor(Color.black);
    g.drawImage(buffer,0,0,null);
  }
  @Override
  public void update(Graphics g){
    paint(g);
  }
  public Point getStartPosition(){
    GridCell start = GridCell.getStartCell();
    return start.getPosition();
  }
  public GridCell[] getAdjacent(GridCell g){
    GridCell next[] = new GridCell[4];
    Point p = g.getPosition();
    if(p.y!=0){next[0]=gridCell[p.x][p.y-1];}
    if(p.x!=w-1){next[1]=gridCell[p.x+1][p.y];}
    if(p.y!=h-1){next[2]=gridCell[p.x][p.y+1];}
    if(p.x!=0){next[3]=gridCell[p.x-1][p.y];}
    return next;
  }
  public GridCell getLowestAdjacent(GridCell g){
```

```
        GridCell next[] = getAdjacent(g);
        GridCell small = next[0];
        double dist = Double.MAX_VALUE;
        for(int i=0;i<4;i++){
          if(next[i]!=null){
            double nextDist = next[i].getDistFromStart();
            if(nextDist<dist && nextDist>=0){
              small = next[i];
                dist = next[i].getDistFromStart();
            }
          }
        }
        return small;
    }
    private void readObject(ObjectInputStream ois) throws IOException,
ClassNotFoundException{
        GridCell.tidy=false;
        ois.defaultReadObject();
        GridCell.setShowPath(false);
    } }
```

- GridCell

```
    package astar22;
    import java.awt.*;
    import java.awt.event.MouseEvent;
    import java.io.IOException;
    import java.io.ObjectInputStream;
    import java.io.Serializable;
    import java.util.Vector;
```

```
public class GridCell extends java.awt.Component implements Serializable
{
    public static final int SET_BLOCKS=0,SET_START=1,SET_FINISH=2;
    public static final double NORMAL = 1,BLOCK=Double.MAX_VALUE;
    private static double newBlockStrength = BLOCK;
    private static int editMode = SET_BLOCKS;
    private static GridCell startCell;
    private static GridCell finishCell;
    private boolean isStart = false;
    private boolean isFinish = false;
  private static Vector cells = new Vector();
  public static boolean tidy = false;
    private static boolean showPath = true;
    private static boolean showDist = true;
    private double cost = 1.0;
    private transient boolean used = false;
    private transient double distFromStart = -1;
    private transient double distFromFinish = -1;
  private boolean partOfPath = false;
    private Point position;
    public GridCell(){
      cells.addElement(this);
      tidy=true;
      enableEvents(AWTEvent.MOUSE_EVENT_MASK);
    }
  public GridCell(boolean block){
      this();
      setTotalBlock(block);
```

```
    }
  public void setPosition(Point p){
      position = p;
    }
  public Point getPosition(){
      return position;
    }
  public static void setEditMode(int mode){
      editMode = mode;
      System.out.println("mode set");
    }
 @Override
   public void processMouseEvent(MouseEvent e){
     super.processMouseEvent(e);
     if(e.getID()==e.MOUSE_CLICKED){
       setShowPath(false);
       switch(editMode){
         case(SET_BLOCKS):
           if(cost!=newBlockStrength){cost=newBlockStrength;}
           else{cost=NORMAL;}
           repaint();
           break;
         case(SET_START):
           setStart(true);
           break;
         case(SET_FINISH):
           setFinish(true);
           break;
```

```
                }
            }
        }
    @Override
        public Dimension getPreferredSize(){
            return new Dimension(10,10);
        }
    public void addToPathFromStart(double distSoFar){
            used = true;

            if(distFromStart == -1){
                distFromStart = distSoFar+cost;
                return;
            }
            if(distSoFar+cost<distFromStart){
                distFromStart = distSoFar+cost;
            }
        }
    public void addToPathFromFinish(double distSoFar){
            used = true;
            if(distFromFinish == -1){
                distFromFinish = distSoFar+cost;
                return;
            }
            if(distSoFar+cost<distFromFinish){
                distFromFinish = distSoFar+cost;
            }
        }
```

```
public double getCost(){
    return cost;
  }
public void setCost(double c){
    cost=c;
  }

  public static GridCell getStartCell(){
    return startCell;
  }
public boolean isStart(){
    return startCell == this;
  }
public void setStart(boolean flag){
      if(flag){
        GridCell temp = this;
        if(startCell !=null){temp = startCell;temp.setStart(false);}
        startCell=this;
        isStart=true;
        repaint();
        temp.repaint();
      }
      else{
        isStart=false;
      }
  }
public static GridCell getFinishCell(){
    return finishCell;
```

```
    }
  public boolean isFinish(){
      return finishCell == this;
    }

    public void setFinish(boolean flag){
      if(flag){
        GridCell temp = this;
        if(finishCell!=null){temp=finishCell;temp.setFinish(false);}
        finishCell=this;
        isFinish=true;
        repaint();
        temp.repaint();
      }
      else{
          isFinish=false;
      }
    }
  public boolean isTotalBlock(){
      return cost==BLOCK;
    }
  public void setTotalBlock(boolean flag){
      if(flag){cost = BLOCK;}
      else{cost = NORMAL;}
    }
  public boolean isUsed(){
      return used;
    }
```

```
    private void resetCell(){
        used = false;
        setPartOfPath(false);
        distFromStart = distFromFinish = -1;
    }
  public static void reset(){
        for(int i=0;i<cells.size();i++){
            ((GridCell)cells.elementAt(i)).resetCell();
        }
    }
  private void clearCell(){
        setCost(NORMAL);
    }
  public static void clearAll(){
        for(int i=0;i<cells.size();i++){
            ((GridCell)cells.elementAt(i)).clearCell();
        }
    }
  public static void setNewBlockStrength(double s){
        if(s<0){newBlockStrength = BLOCK;}
        else{newBlockStrength = s;}
    }
  public static void setShowPath(boolean flag){
            showPath = flag;
    }

    public static boolean isShowPath(){
```

```
        return showPath;
    }
  public boolean isPartOfPath(){
        return partOfPath;
    }
  public void setPartOfPath(boolean flag){
        partOfPath = flag;
    }
  public double getDistFromStart(){
        if(GridCell.startCell == this){return 0;}
        if(isTotalBlock()){return -1;}
        return distFromStart;
    }
  @Override
    public void paint(Graphics g){
        Dimension size = getSize();
        g.setColor(Color.white);
        if(cost!=NORMAL){
                if(cost==BLOCK){g.setColor(Color.black);}
        }
        if(showPath && partOfPath){
          g.setColor(Color.yellow);
        }
        if(startCell == this){
          g.setColor(Color.green);
        }
        if(finishCell == this){
          g.setColor(Color.red);
```

```
        }
        g.fillRect(0,0,size.width,size.height);
    g.setColor(Color.black);
        if(showDist &&
    distFromStart>0){g.drawString(""+distFromStart,1,(int)(size.height*0.75));}
        g.drawRect(0,0,size.width-1,size.height-1);
      }
    private void readObject(ObjectInputStream ois) throws IOException,
    ClassNotFoundException{
        if(!tidy){
                cells = new Vector();
          tidy=true;
        }
    ois.defaultReadObject();
        cells.addElement(this);
        if(isStart){setStart(true);}
        if(isFinish){setFinish(true);}
      }
    }
```

- HeuristicStar

```
package astar22;
/*
 * To change this template, choose Tools | Templates and open the template in
 * the editor.
 */
/**
 *
 * @author giannis
 */
import java.awt.Point;
import java.util.Vector;
public class HuristicAStar extends OneTailAStar implements PathFinder
{
   double minCost;
   public HuristicAStar(){
      super();
      stepSpeed=20;
   }
   /**
    * calculates the waighted manhattan distance from a to b
    */
   public double cbDist(Point a,Point b,double low){
      return low * (Math.abs(a.x-b.x)+Math.abs(a.y-b.y)-1);
   }
   public GridCell[] findPath(Map map)
   {
      minCost = Double.MAX_VALUE;
```

```
        for(int i=0;i<map.w;i++){
          for(int j=0;j<map.h;j++){
            minCost = Math.min(map.gridCell[i][j].getCost(),minCost);
          }
        }
        //minCost=0.9;
        System.out.println("Cheepest Tile = "+minCost);
        return super.findPath(map);
      }
    public int step(){
        int tests = 0;
        boolean found = false;
        boolean growth = true;
        GridCell finish = GridCell.getFinishCell();
        Point end = finish.getPosition();
        Vector temp = (Vector) edge.clone();
        //find the most promesing edge cell
        double min = Double.MAX_VALUE;
        double score;
        //int best = -1;
        GridCell best = (GridCell)temp.elementAt(temp.size()-1); ;
        GridCell now;
        for(int i=0;i<temp.size();i++){
          now = (GridCell)temp.elementAt(i);
          if(!done.contains(now)){
            //score =now.getDistFromStart();
            score =now.getDistFromStart();
            score += cbDist(now.getPosition(),end,minCost);
```

```
            if(score<min){
              min = score;
                best = now;
            }
          }
        }
        now = best;
        edge.removeElement(now);
        done.addElement(now);
        GridCell next[] = map.getAdjacent(now);
        for(int i=0;i<4;i++){
          if(next[i]!=null){
            if(next[i]==finish){found=true;}
            if(!next[i].isTotalBlock()){
              next[i].addToPathFromStart(now.getDistFromStart());
              tests++;
              if(!edge.contains(next[i]) &&
!done.contains(next[i])){edge.addElement(next[i]);growth=true;}
            }
          }
          if(found){return FOUND;}
        }
        map.repaint();
        if(edge.size()==0){return NO_PATH;}
       //now process best.
        return NOT_FOUND;
    }
}
```

- PathFinder

```
package astar22;
 public interface PathFinder
{
public GridCell[] findPath(Map map);
}
```

- StartUp

```
package astar22;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
/*
 * To change this template, choose Tools | Templates and open the template in
 * the editor.
 */
/**
 *
```

```
 * @author giannis
 */
public class Startup implements ActionListener
{
int dim, dim1;
private JTextField tex=new JTextField(3);
   private JTextField tex1=new JTextField(3);
  void Initialise()
   {
      System.out.println("ΑΡΧΙΚΟΠΟΙΗΣΗ");

      JFrame basic=new JFrame("ΕΡΓΑΣΙΑ A*");
      JPanel p1=new JPanel();
     p1.setLayout(new GridLayout(3,2));
      basic.setContentPane(p1);
     JLabel lab=new JLabel("ΕΙΣΑΓΕΤΑΙ ΑΡΙΘΜΟ ΓΡΑΜΜΩΝ");
      JLabel lab1=new JLabel("ΕΙΣΑΓΕΤΑΙ ΑΡΙΘΜΟ ΣΤΗΛΩΝ");
      JButton but1=new JButton("OK");
     but1.addActionListener(this);
      but1.setActionCommand("OK");
     p1.add(lab);
      p1.add(tex);
      p1.add(lab1);
      p1.add(tex1);
      p1.add(but1);
      p1.setBorder(BorderFactory.createEmptyBorder(20, 40, 20, 40));
     basic.pack();
      basic.setMinimumSize(new Dimension(200, 20));
```

```
        basic.setLocationRelativeTo(null);
        basic.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        basic.setVisible(true);
    }
     void spawnWindow(int dim,int dim1)
    {
        System.out.println("ΔΗΜΙΟΥΡΓΙΑ ΔΙΑΣΤΑΣΕΩΝ");

        AStar as=new AStar(dim,dim1);
      as.pack();
       as.setLocationRelativeTo(null);
       as.setVisible(true);
    }
     @Override
    public void actionPerformed(ActionEvent e)
    {
     if("OK".equals(e.getActionCommand()))
      {
     if(tex.getText().isEmpty())
            JOptionPane.showMessageDialog(null,"ΠΑΡΑΚΑΛΩ ΕΙΣΑΓΕΤΑΙ ΑΡΙΘΜΟ
ΓΡΑΜΜΩΝ",
                    "Error",JOptionPane.ERROR_MESSAGE);
        else
        {
           try
           {
     int dim2=Integer.parseInt(tex.getText());
     if(dim2<2||dim2>10)
              {
```

```
                JOptionPane.showMessageDialog(null,"ΟΙ ΓΡΑΜΜΕΣ ΠΡΕΠΕΙ ΝΑ
 ΕΙΝΑΙ " +
 "ΜΕΤΑΞΥ 2 ΚΑΙ 10","Error",JOptionPane.ERROR_MESSAGE);
                  return;
                }
                dim=dim2;
              }
              catch(NumberFormatException nfe)
              {
                JOptionPane.showMessageDialog(null,"ΜΟΝΟ ΑΚΕΡΑΙΕΣ ΤΙΜΕΣ
 ΕΠΙΤΡΕΠΟΝΤΑΙ",
                    "Error",JOptionPane.ERROR_MESSAGE);
                return;
              }
            }
        if(tex1.getText().isEmpty())
              JOptionPane.showMessageDialog(null,"ΠΑΡΑΚΑΛΩ ΕΙΣΑΓΕΤΑΙ ΑΡΙΘΜΟ
 ΣΤΗΛΩΝ",
                    "Error",JOptionPane.ERROR_MESSAGE);
            else
            {
              try
              {
 int dim3=Integer.parseInt(tex1.getText());
 if(dim3<2||dim3>10)
                {
                  JOptionPane.showMessageDialog(null,"ΟΙ ΣΤΗΛΕΣ ΠΡΕΠΕΙ ΝΑ
 ΕΙΝΑΙ" +
                      "ΜΕΤΑΞΥ 2 ΚΑΙ 10","Error",JOptionPane.ERROR_MESSAGE);
                  return;
```

```
                }


                dim1=dim3;
            }
            catch(NumberFormatException nfe)
            {
                JOptionPane.showMessageDialog(null,"MONO AKEPAIEΣ TIMEΣ
ΕΠΙΤΡΕΠΟΝΤΑΙ",
                    "Error",JOptionPane.ERROR_MESSAGE);
                return;
            }
        }
    }
        spawnWindow(dim,dim1);
}
}
```

- OneTailA*

```
package astar22;

import java.util.Vector;

public class OneTailAStar extends java.lang.Object implements
PathFinder,Runnable

{

public final int NO_PATH=-1,NOT_FOUND=0,FOUND=1;

  protected Vector edge;

  protected Vector done;

  protected Map map;

  int stepSpeed = 100;

  private int maxSteps = 1000;

  Thread loop;

  double distFromStart = 0;

  private boolean findFirst = false;

  @Override

  public GridCell[] findPath(Map map)

  {

    this.map = map;

    GridCell.reset();

    edge = new Vector();

    done = new Vector();

    System.out.println("calculating route");

    if(GridCell.getStartCell() == null){

      System.out.println("No start point set");

      return null;

    }

    if(GridCell.getFinishCell() == null){
```

```
            System.out.println("No finish point set");
            return null;
        }
        System.out.println("Starting from "+map.getStartPosition());
        loop = new Thread(this);
        loop.start();
        return null;
    }
     @Override
    public void run(){
        edge.addElement(GridCell.getStartCell());
        int pass =0;
        boolean found = false;
        double start,diff;
        int state=NOT_FOUND;
        while(state==NOT_FOUND && pass<maxSteps){
            pass++;
            start = System.currentTimeMillis();
            state = step();
            diff = System.currentTimeMillis()-start;
            try{
                loop.sleep(Math.max((long)(stepSpeed-diff),0));
            }
            catch(InterruptedException e){}
           // System.out.println(diff);
        }
        if(state==FOUND){
            setPath(map);
```

```
        }
        else{System.out.println("No Path Found");}
    }
  public int step(){
        int tests = 0;
        boolean found=false;
        boolean growth=false;
        GridCell finish = GridCell.getFinishCell();
        Vector temp = (Vector) edge.clone();
        for(int i=0;i<temp.size();i++){
          GridCell now = (GridCell)temp.elementAt(i);
          GridCell next[] = map.getAdjacent(now);
          for(int j=0;j<4;j++){
            if(next[j]!=null ){
              if(next[j]==finish){found=true;}
              next[j].addToPathFromStart(now.getDistFromStart());
              tests++;
              if(!next[j].isTotalBlock() &&
!edge.contains(next[j])){edge.addElement(next[j]);growth=true;}
            }
          }
          if(found){return FOUND;}
          done.addElement(now);
    }
     map.repaint();
     if(!growth){return NO_PATH;}
    return NOT_FOUND;
  }
  public void setPath(Map map){
```

```
        System.out.println("Path Found");
        GridCell.setShowPath(true);
        boolean finished = false;
        GridCell next;
        GridCell now = GridCell.getFinishCell();
        GridCell stop = GridCell.getStartCell();
        while(!finished){
          next=map.getLowestAdjacent(now);
          now=next;
          now.setPartOfPath(true);
          now.repaint();
          if(now == stop){finished = true;}
          try{
            loop.sleep(stepSpeed);
          }
          catch(InterruptedException e){}
      }
      System.out.println("Done");
}
}
```

- A*

```
package astar22;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
```

```
import javax.swing.JFrame;
public class AStar extends JFrame implements ItemListener,ActionListener
{
   boolean isDesign = false;
   CheckboxGroup group;
   Checkbox blocks,start,finish;
   Choice level = new Choice();
Map map;
Choice preset = new Choice();
   Choice user = new Choice();
   Button go,clear;


  PathFinder finder = new HuristicAStar();
    public AStar (int dim,int dim1)
    {
     map = new Map(dim,dim1);
    Container cp=getContentPane();
    setLayout(new BorderLayout());
    setSize(612,482);
     cp.add(map,"Center");
     cp.setBackground(Color.BLACK);
     Panel m = new Panel();
     m.setLayout(new GridLayout(1,0));
     m.add(new Label(" Classic A* "));
    m.setBackground(Color.LIGHT_GRAY);
    cp.add(m,"North");
     Panel p = new Panel();
```

```
p.setLayout(new GridLayout(4,1));
p.setBackground(Color.LIGHT_GRAY);
Panel b = new Panel();
b.setLayout(new GridLayout(2,1));
b.setBackground(Color.LIGHT_GRAY);
level.add("ΕΜΠΟΔΙΟ");
level.addItemListener(this);
preset.addItemListener(this);


group = new CheckboxGroup();
blocks = new Checkbox("ΕΜΠΟΔΙΟ",group,true);
start = new Checkbox("ΑΡΧΗ",group,false);
finish = new Checkbox("ΤΕΛΟΣ",group,false);
blocks.addItemListener(this);
start.addItemListener(this);
finish.addItemListener(this);
b.add(blocks);
b.add(level);
p.add(b);
p.add(start);
p.add(finish);
Panel g = new Panel();
g.setBackground(Color.LIGHT_GRAY);
if(!isDesign){
  g.setLayout(new GridLayout(3,1));
}
else{
```

```
            g.setLayout(new GridLayout(2,2));
        }
        go = new Button("ΕΝΑΡΞΗ");
        clear = new Button("ΚΑΘΑΡΙΣΜΟΣ");
        g.add(go);
        g.add(clear);



        p.add(g);
        go.addActionListener(this);
        clear.addActionListener(this);
       cp.add(p,"East");
       }
       @Override
              public void itemStateChanged(ItemEvent e){
                 if(e.getSource()==level){
                    blocks.setState(true);
                    GridCell.setEditMode(GridCell.SET_BLOCKS);
                    switch(level.getSelectedIndex()){
                       case 0:
                          GridCell.setNewBlockStrength(GridCell.BLOCK);
                          return;
default:
                          GridCell.setNewBlockStrength(GridCell.NORMAL);
                          return;
                    }
                 }
```

```
                Checkbox box = group.getSelectedCheckbox();
                if(box ==
blocks){GridCell.setEditMode(GridCell.SET_BLOCKS);return;}
                if(box == start){GridCell.setEditMode(GridCell.SET_START);return;}
                if(box == finish){GridCell.setEditMode(GridCell.SET_FINISH);return;}
}
@Override
            public void actionPerformed(ActionEvent e){
              if(e.getSource() == go){
                finder.findPath(map);
              }
              if(e.getSource() == clear){
                GridCell.clearAll();
                map.repaint();
              }
            }

@Override
            public Dimension getPreferredSize(){
              return new Dimension(520,420);
            }
}
```