ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΤΗ ΡΟΜΠΟΤΙΚΗ

ΘΕΜΑ

Σχεδιασμός διαδρομής αυτοκινούμενου ρομπότ με βάση εικόνα κάμερας, εφαρμογή της μεθόδου Πιθανοτικού Οδικού Χάρτη – PRM και καθοδήγηση μέσω WiFi

ΦΟΙΤΗΤΗΣ

ΜΑΤΡΑΛΗΣ ΓΡΗΓΟΡΙΟΣ

ΕΠΙΒΛΕΠΩΝ

ΣΠΥΡΙΔΩΝ ΚΑΖΑΡΛΗΣ

Περίληψη

Η παρούσα διπλωματική εργασία αναλύει την κατασκευή αυτόνομης ρομποτικής πλατφόρμα με την χρήση μικρο ελεγκτή Arduino Uno και τον έλεγχό της μέσω ηλεκτρονικού υπολογιστή και κάμερας εφαρμόζοντας την μέθοδο PRM σε καθορισμένο χώρο κίνησης. Η εργασία περιλαμβάνει την υλοποίηση δύο συστημάτων, ένα υψηλού επιπέδου που είναι υπεύθυνο για τον υπολογισμό της διαδρομής και ένα χαμηλού επιπέδου που αφορά την κατασκευή του ρομπότ.

Το σύστημα υψηλού επιπέδου υλοποιήθηκε στον ηλεκτρονικό υπολογιστή με την βοήθεια του προγράμματος Matlab. Η εργασία περιγράφει τα βήματα που ακολουθήθηκαν για την κατάλληλη επεξεργασία των εικόνων που λαμβάνει η κάμερα ώστε να γίνει ο προσδιορισμός των διαστάσεων του χώρου κίνησης καθώς και τον συντεταγμένων των εμποδίων αλλά και των σημείων αφετηρίας και τερματισμού. Στην συνέχεια περιγράφεται πως τα αποτελέσματα της μεθόδου μετατρέπονται κατάλληλα για να μπορεί να γίνει η επεξεργασία τους από τον μικρο ελεγκτή. Επίσης περιγράφεται ο τρόπος επικοινωνίας μεταξύ των δύο συστημάτων και πως αυτός υλοποιήθηκε.

Το σύστημα χαμηλού επιπέδου περιλαμβάνει την κατασκευή του ρομπότ και την ανάπτυξη αλγορίθμου για τον έλεγχό του. Στην εργασία περιγράφεται το hardware που χρησιμοποιήθηκε για την κατασκευή του ρομπότ και την συνδεσμολογία του. Επιπλέον περιγράφονται οι ρουτίνες ελέγχου που αναπτύχθηκαν για τον έλεγχο της κίνησης του ρομπότ.

Abstract

This paper describes the construction of an autonomous vechicle using the microcontorller Arduino Uno and how to control it through a camera by applying the PRM method with Matlab's help. Also, the steps for processing the pictures from camera in order to apply the PRM method, are explained, and also the way the data are sent to the robot. Also it describes the hardware that was used to construct the robot and the necessary routines that were developed in order to control it.

Κατάλογος περιεχομένων

Κατάλογος εικόνων

Ευρετήριο πινάκων

Κεφάλαιο 1: Αναλυτική διατύπωση του προβλήματος

Η μεταπτυχιακή εργασία, πραγματεύεται την υλοποίηση ενός αυτόνομου ρομποτικού οχήματος, που θα φέρει πλακέτα μικρο ελεγκτή, τύπου Arduino ή Raspberry, το οποίο θα αποτελέσει την πλατφόρμα για την υλοποίηση μεθόδων Σχεδιασμού Διαδρομής. Το ρομπότ θα κινείται σε αυστηρά καθορισμένο χώρο, όπου το πάτωμα του ελεύθερου χώρου θα είναι άσπρου χρώματος ενώ θα υπάρχουν και εμπόδια μαύρου χρώματος. Τον χώρο θα κατοπτεύει κάμερα που θα λαμβάνει ασπρόμαυρη εικόνα από τον χώρο κίνησης του ρομπότ. Η εικόνα θα φορτώνεται στο Matlab όπου και με την βοήθεια αλγορίθμου PRM θα σχεδιάζεται μία βέλτιστη διαδρομή κίνησης, από την αφετηρία στον τερματισμό, με αποφυγή των εμποδίων. Η σχεδιασμένη διαδρομή θα αποστέλλεται μέσω Bluetooth από το PC στο ρομπότ, το οποίο στη συνέχεια θα αναλαμβάνει να εκτελέσει την διαδρομή, χωρίς ανάδραση. Η πλατφόρμα του αυτόνομου ρομποτικού οχήματος θα ελέγχεται μέσω Arduino ή και Rasberry Pi όπου θα πρέπει να αναπτυχθούν οι βασικές ρουτίνες χαμηλού επιπέδου για κίνηση του ρομπότ, όπως και οι ρουτίνες επικοινωνίας μέσω του Bluetooth.

Κεφάλαιο 2: State of the art

Αυτό το κεφάλαιο περιγράφει τις μεθόδους σχεδίασης διαδρομών για αυτοκινούμενα ρομπότ. Οι μέθοδοι σχεδίασης διαδρομών χωρίζονται σε δύο κατηγορίες, οι ντετερμινιστικές μέθοδοι και οι στοχαστικές μέθοδοι.

Οι ντετερμινιστικές μέθοδοι, έχοντας τα ίδια δεδομένα ως είσοδο, θα καταλήξουν πάντα στο ίδιο αποτέλεσμα[1]. Αν το πρόβλημα είναι επιλύσιμο μία ντετερμινιστική μέθοδος θα βρίσκει πάντα την λύση. Ένα μειονέκτημα των ντετερμινιστικών μεθόδων είναι ότι απαιτούν μεγάλη επεξεργαστική ισχύ και για πολύπλοκα προβλήματα έχουν μεγάλο χρόνο εκτέλεσης.

Παραδείγματα ντετερμινιστικών μεθόδων είναι:

- Γραφήματα ορατότητας (Visibility Graphs)
- Διαγράμματα Voronoi(Voronoi Diagrams)

Οι στοχαστικές μέθοδοι, έχοντας τα ίδια δεδομένα ως είσοδο, δεν καταλήγουν στο ίδιο αποτέλεσμα. Οι στοχαστικές μέθοδοι προσπαθούν να επιλύσουν το πρόβλημα αρχικοποιόντας το με τυχαίες τιμές[2]. Ένα μειονέκτημα των στοχαστικών μεθόδων είναι ότι λόγω της τυχαίας αρχικοποίησης που εφαρμόζουν υπάρχει περίπτωση να μην βρεθεί λύση σε πρόβλημα που είναι επιλύσιμο. Αυτό το πρόβλημα μπορεί να ελαχιστοποιηθεί χρησιμοποιώντας κατάλληλες τεχνικές βελτιστοποίησης. Τα πλεονεκτήματα των μεθόδων αυτών είναι ότι απαιτούν μικρή επεξεργαστική ισχύ και δεν έχουν μεγάλο χρόνο εκτέλεσης.

Παραδείγματα στοχαστικών μεθόδων είναι:

- Πιθανοτικοί οδικοί χάρτες(Probabilistic Road Map - PRM)
- Ταχέων εξερευνητικών τυχαίων δένδρων(Rapidly Exploring Random Trees - RRT)

2.1 Πιθανοτικοί οδικοί χάρτες(Propabilistic RoadMaps)

Για την επίλυση του προβλήματος επιλέχθηκε η χρήση της μεθόδου των Πιθανοτικών οδικών χαρτών(στο εξής PRM).Η μέθοδος PRM δημιουργεί τυχαία σημεία εντός του

ελεύθερου χώρου και έπειτα ενώνει τα γειτονικά σημεία[3]. Στην συνέχεια δημιουργείται ο γράφος όπου θα πρέπει να αναζητήσει τη βέλτιστη διαδρομή. Κατά την διαδικασία της δημιουργίας των σημείων η μέθοδος τα παράγει επαναληπτικά ελέγχοντας εάν βρίσκονται εντός του ελεύθερου χώρου. Αν το σημείο που πρόκειται να παραχθεί βρίσκεται πάνω σε εμπόδιο τότε αυτό απορρίπτεται και η μέθοδος παράγει καινούργιο σημείο. Αφού έχει τελειώσει η παραγωγή των τυχαίων σημείων προστίθενται και τα σημεία αφετηρίας και τερματισμού. Τέλος κάθε σημείο ενώνεται με έναν αριθμό γειτονικών σημείων για να δημιουργηθεί ο γράφος.

Η δημιουργία των τυχαίων σημείων όπως και η ένωση των γειτονικών σημείων είναι παράμετροι λειτουργίας του αλγορίθμου και πρέπει να ρυθμιστούν για την βέλτιστη λειτουργία της μεθόδου.

Αν τα σημεία είναι πολύ λίγα τότε υπάρχει πιθανότητα να μην είναι εφικτή η διασύνδεση ανάμεσα στην αφετηρία και τον τερματισμό. Αν τα σημεία είναι πάρα πολλά ο παραπάνω κίνδυνος εξαλείφεται αλλά ο γράφος που δημιουργείται γίνεται σημαντικά μεγαλύτερος και πιο πολύπλοκος, με αποτέλεσμα η φάση της αναζήτησης της διαδρομής να είναι πιο δύσκολη. Η μέθοδος έχει την δυνατότητα να ξεκινήσει με μικρό αριθμό σημείων και σταδιακά να τα αυξάνει προσθέτοντας επιπλέον σημεία, εφόσον δεν έχει βρεθεί αποδεκτή λύση.

Για την σύνδεση των γειτονικών σημείων υπάρχουν διάφοροι μέθοδοι που μπορούν να χρησιμοποιηθούν, με τους πιο γνωστούς να είναι οι Euclidean, Scaled Euclidean, Minkowski, Modified Minkowski και Manhattan[4]. Παρακάτω αναλύονται οι δύο πρώτοι τρόποι, Euclidean και Scaled Euclidean. Ο πρώτος τρόπος είναι γνωστός και ως διασύνδεση των κ-κοντινότερων γειτόνων(k-nearest neighbors). Με τον τρόπο αυτό ο αλγόριθμος υπολογίζει, για κάθε σημείο του γράφου, την Ευκλείδεια απόσταση από τα υπόλοιπα σημεία του γράφου με τα οποία έχει "οπτική επαφή". Με τον όρο "οπτική επαφή" εννοείται ότι η ευθεία που ενώνει τα δύο σημεία βρίσκεται εξ' ολοκλήρου εντός του ελεύθερου χώρου και δεν τέμνει κανένα από τα εμπόδια[5][6]. Στην συνέχεια επιλέγονται τα k σημεία που έχουν την μικρότερη απόσταση από το εν λόγω σημείο και προς αυτά δημιουργούνται συνδέσεις-ακμές του γράφου. Η παράμετρος k είναι μία παράμετρος του αλγορίθμου. Αν το k είναι σχετικά μικρό τότε ο γράφος είναι αραιός(sparse). Αν το k είναι μεγάλο τότε ο γράφος είναι

πυκνός(dense)[5].

Ο δεύτερος τρόπος είναι η διασύνδεση των γειτονικών σημείων που βρίσκονται σε μικρότερη απόσταση από ένα όριο απόστασης d. Με τον τρόπο αυτό ο αλγόριθμος υπολογίζει, για κάθε σημείο του γράφου, την Ευκλείδεια απόσταση από τα υπόλοιπα σημεία του γράφου με τα οποία έχει "οπτική επαφή". Με τον όρο "οπτική επαφή" εννοείται ότι η ευθεία που ενώνει τα δύο σημεία βρίσκεται εξ' ολοκλήρου εντός του ελεύθερου χώρου και δεν τέμνει κανένα από τα εμπόδια[5][6]. Στην συνέχεια επιλέγονται τα σημεία που έχουν απόσταση από το εν λόγω σημείο μικρότερη από d: για συγκεκριμένο σημείο $i \in G$, για κάθε $j \in G$, $i \neq j$, επέλεξε j αν distance(i, j)<d όπου G ο γράφος των κόμβων. Τέλος, προς τα επιλεγμένα σημεία δημιουργούνται συνδέσεις-ακμές του γράφου από το εν λόγω σημείο. Η παράμετρος d είναι παράμετρος του αλγορίθμου. Αν το d είναι μικρό τότε ο γράφος είναι αραιός(sparse) και μπορεί να μην περιέχει ολοκληρωμένες διαδρομές από την αφετηρία προς τον τερματισμό. Αν το d είναι μεγάλο τότε ο γράφος είναι πυκνός(dense) οπότε είναι σίγουρη η ύπαρξη ολοκληρωμένων διαδρομών αλλά, γίνεται πιο δύσκολη η αναζήτησή τους από τον αλγόριθμο αναζήτησης διαδρομών στον γράφο. Μετά την σύνδεση των σημείων-κόμβων εισάγονται και τα σημεία αφετηρίας και τερματισμού και συνδέονται κι αυτά με τους υπόλοιπους κόμβους με βάση την ίδια μέθοδο[5].

Όπως κάθε μέθοδος, έτσι και η μέθοδος PRM έχει κάποια πλεονεκτήματα και κάποια μειονεκτήματα. Το κυριότερο πλεονέκτημα της μεθόδου είναι ότι είναι απλή στην υλοποίησή της. Επίσης, η μέθοδος PRM έχει αποδειχθεί θεωρητικά ότι είναι πιθανοτικά πλήρης, που σημαίνει ότι όταν ο αριθμός των τυχαίων σημείων τείνει στο άπειρο, η πιθανότητα μη εύρεσης μίας λύσης, εφόσον αυτή υπάρχει, τείνει οριακά στο μηδέν[5].

Ένα από τα μειονεκτήματα της μεθόδου είναι ότι η εκτέλεσή της υπόκειται σε παραμέτρους(αριθμός τυχαίων σημείων, k, d) που πρέπει να ρυθμιστούν για κάθε εφαρμογή. Επιπλέον, δεν υπάρχει τρόπος υπολογισμού του απαιτούμενου αριθμού κόμβων για κάθε χώρο διάταξης, και αυτός θα πρέπει να καθορίζεται πειραματικά με διαδικασίες δοκιμής και λάθους. Τέλος, σε πυκνούς χώρους εμποδίων η απόρριψη τυχαίων σημείων μπορεί να γίνεται σε υψηλό ποσοστό απαιτώντας υψηλό υπολογιστικό φόρτο[5].

Κεφάλαιο 3: Hardware

Για την υλοποίηση του ρομπότ χρησιμοποιήθηκαν:

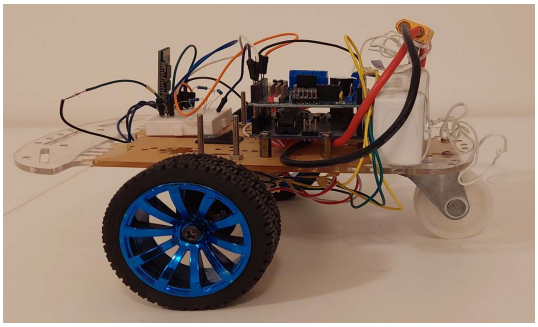
- 1x Μικροεπεξεργαστής Arduino Uno.
- 1x Πλακέτα οδήγησης μοτέρ L298P.
- 1x Πλακέτα Bluetooth HC-06.
- 2x Μοτέρ με 12V DC 130rpm με encoder.
- 2x Ρόδες με λάστιχο διαμέτρου 65mm.
- 1x Μπαταρία 11,1V LIPO 2200mAh
- 1x Πλατφόρμα δίτροχου ρομπότ.
- 1x USB camera.

Ο μικροεπεξεργαστής(στο εξής Uno) είναι υπεύθυνος για δύο διαδικασίες. Η πρώτη είναι να διαβάσει τα δεδομένα που του στέλνει ο ηλεκτρονικός υπολογιστής(στο εξής Η/Υ). Η σύζευξη Η/Υ και Uno γίνεται μέσω του πρωτοκόλλου Bluetooth. Για να έχει την δυνατότητα να συνδεθεί ο Uno με το πρωτόκολλο αυτό χρησιμοποιήθηκε η πλακέτα Bluetooth HC-06(στο εξής HC-06). Το HC-06 επικοινωνεί με τον Uno με σειριακή επικοινωνία. Μόλις το HC-06 λάβει τα δεδομένα τα στέλνει στον Uno όπου είναι υπεύθυνος για την επεξεργασία τους.

Η δεύτερη διαδικασία που είναι υπεύθυνος ο Arduino είναι η οδήγηση και ο έλεγχος των μοτέρ. Επειδή οι έξοδοι του Arduino δεν έχουν αρκετή ισχύ για να οδηγήσουν απευθείας τα μοτέρ χρησιμοποιήθηκε η πλακέτα οδήγησης μοτέρ L298P (στο εξής L298P). Το L298P έχει την δυνατότητα να βγάλει στην έξοδο του την ισχύ που χρειάζεται για την οδήγηση των μοτέρ. Ο Arduino μπορεί να ελέγξει την ισχύ που θα παρέχει το L298P στα μοτέρ, και επομένως και την ταχύτητά τους, αξιοποιώντας την λειτουργία PWM(Pulse Width Modulation).

Για τον έλεγχο των μοτέρ ο Arduino χρησιμοποιεί τα δεδομένα από τους encoder των μοτέρ. Με αυτό τον τρόπο είναι δυνατόν να ελεγχθεί η ταχύτητα περιστροφής του κάθε μοτέρ και να μεταβληθεί σε επιθυμητές τιμές.

Όλα τα παραπάνω έχουν συνδεθεί πάνω στη πλατφόρμα δίτροχου ρομπότ (Εικόνα 1)όπου παρέχει και μία τρίτη ρόδα τύπου caster για την στήριξη του ρομπότ. Τέλος η κάμερα τύπου USB συνδέεται με τον Η/Υ για την λήψη φωτογραφιών του χώρου διάταξης.



Εικόνα 1: Ρομποτική Πλατφόρμα

3.1: Τεχνικά χαρακτηριστικά

Arduino Uno:

- Πλακέτα: Arduino UNO R3
- Μικρο ελεγκτής: ATmega328P
- Σύνδεση USB: USB-B
- Ψηφιακές θύρες I/O: 14
- Αναλογικές θύρες εισόδου: 6
- PWM pins: 6
- UART: Ναι
- I2C: Ναι
- SPI: Ναι
- Τάση θυρών I/O: 5V
- Τάση λειτουργίας: 7V-12V
- Ένταση ρεύματος DC ανά θύρα I/O: 20mA
- Χρονισμός: 16MHz
- Μνήμη: 2KB SRAM, 32KB FLASH, 1KB EEPROM

Πλακέτα οδήγησης μοτέρ L298P:

- Τάση ψηφιακής εισόδου: 5V
- Τάση λειτουργίας: VIN 6.5V - 12V, PWR IN 4.8V - 24V
- Ένταση ρεύματος ψηφιακών Iss: ≤ 36mA
- Ένταση ρεύματος ισχύος Io: ≤ 2A
- Μέγιστη κατανάλωση ισχύς: 25W (T= 75°C)
- Επίπεδο τάσης σήματος ελέγχου: High level: 2.3V≤ Vin ≤ 5V
  Low level: -0.3 V ≤ Vin ≤ 15V

Πλακέτα Bluetooth HC-06:

- Πρωτόκολλο Bluetooth: Bluetooth V2.0 protocol standard
- Επίπεδο ισχύς: Class2(+6dBm)
- Εύρος συχνοτήτων: 2.40GHz—2.48GHz, ISM Band
- Ευαισθησία δέκτη: -85dBm
- Πρωτόκολλο USB: USB v1.1/2.0
- Λειτουργία διαμόρφωσης: Gauss frequency Shift Keying
- Χαρακτηριστικά ασφαλείας: Authentication and encryption
- Τάση λειτουργίας: +3.3V - +6V
- Ένταση ρεύματος λειτουργίας: 40mA

Μοτέρ:

- Τάση λειτουργίας: DC 12V
- Ισχύς: 3W
- Αναλογία μειωτήρα: 1/50
- Έλεγχος ταχύτητας: Ναι
- Ταχύτητα: 130rpm
- Παλμοί ανά περιστροφή: 11

Κάμερα USB:

- Μοντέλο: DIGOO DG-PCS2
- Συνδεσιμότητα: USB 2.0
- Ανάλυση: 2.0 Mega Pixels, 1932 (H) x 1088 (V)
- Αισθητήρας εικόνας: 1/2.9
- Αναλογία σήματος προς θόρυβο: >50dB
- Ροή Video: H.264: 1920*1080/1920*720/640*360
  MIPEG: 1920*1080/1920*720/640*360
- Ψηφιακό κλείστρο: Αυτόματο

Πλατφόρμα δίτροχου ρομπότ

- Διαστάσεις (Μήκος x Πλάτος): 20cm x 14cm
- Διαστάσεις με ρόδες (Μήκος x Πλάτος): 20cm x 19cm

Κεφάλαιο 4: Software

Για την ανάπτυξη του κώδικα χρησιμοποιήθηκαν δύο προγράμματα, το Arduino IDE και το Matlab. Το Arduino IDE χρησιμοποιήθηκε για να αναπτυχθούν οι απαραίτητες ρουτίνες ελέγχου χαμηλού επιπέδου όπου τρέχουν στον Uno, όπως η οδήγηση των μοτέρ και η επικοινωνία μέσω Bluetooth με τον Η/Υ. Το Arduino IDE είναι ένα λογισμικό ανοιχτού κώδικα και διατίθεται δωρεάν. Η γλώσσες προγραμματισμού που υποστηρίζει είναι η C και η C++. Επιπλέον το Arduino IDE προσφέρει μια συλλογή από βιβλιοθήκες από συμβατά περιφερειακά που μπορούν να συνδεθούν με τον Uno, κάνοντας έτσι την εγκατάσταση και χρήση τους ευκολότερη. Μία ακόμη δυνατότητα που δίνει στους χρήστες το αναπτυξιακό αυτό είναι ο έλεγχος του κώδικα για τυχόν λογικά λάθη. Ο χρήστης μπορεί πολύ εύκολα, με το πάτημα ενός κουμπιού, να ελέγξει για τυχόν λάθη στον κώδικα. Ένα πλεονέκτημα του Arduino IDE είναι ότι μπορεί να προγραμματίσει τον Uno χωρίς την χρήση κάποιου εξειδικευμένου programmer. Το μόνο που χρειάζεται είναι ένα καλώδιο USB για την σύνδεση του Η/Υ με τον Uno κάνοντας όλη την διαδικασία πιο εύκολη και γρηγορότερη.

Δεύτερο αναπτυξιακό πρόγραμμα που χρησιμοποιήθηκε είναι το Matlab(Matrix Laboratory). Το Matlab δεν διατίθεται και απαιτείται η αγορά αδείας για την χρήση του. Υπάρχει και η δυνατότητα απόκτησης δοκιμαστικής έκδοσης που επιτρέπει την χρήση του προγράμματος για περιορισμένο χρονικό διάστημα. Το Matlab είναι ένα περιβάλλον αριθμητικής υπολογιστικής. Αποθηκεύει και κάνει πράξεις με βάση την άλγεβρα πινάκων. Χρησιμοποιείται για την επίλυση μαθηματικών προβλημάτων, ωστόσο είναι πολύ ισχυρό και μπορεί να να χρησιμοποιηθεί και για προγραμματισμό καθώς περιέχει πολλές εντολές από την C++. Περιέχει μεγάλο φάσμα βιβλιοθηκών για επίλυση προβλημάτων σε διάφορες τεχνολογίες όπως έλεγχος συστημάτων, επεξεργασία εικόνων, ρομποτική μηχανική μάθηση και άλλα.

Με την βοήθεια του Matlab έχουν υλοποιηθεί οι ρουτίνες υψηλού επιπέδου. Με τις ρουτίνες αυτές γίνεται η επεξεργασία των εικόνων ώστε να βρεθούν οι συντεταγμένες των εμποδίων, της αφετηρίας και του τερματισμού πάνω στο χώρο διάταξης. Στην συνέχεια με την χρήση της μεθόδου PRM υπολογίζεται η διαδρομή που πρέπει να ακολουθήσει το ρομποτ και στέλνεται με την χρήση του πρωτοκόλου Bluetooth. Τέλος κατά την διάρκεια της εκτέλεσης της διαδρομής που ακολουθεί το ρομπότ μπορεί να υπολογιστεί αν η διαδρομή που ακολουθείται είναι σωστή. Αν η διαδρομή δεν είναι σωστή τότε μέσω του Matlab μπορούν να υπολογιστουν διορθωτικές κινήσεις για το ρομπότ ώστε να αποφευχθεί η σύγκρουσή του με κάποιο εμπόδιο.

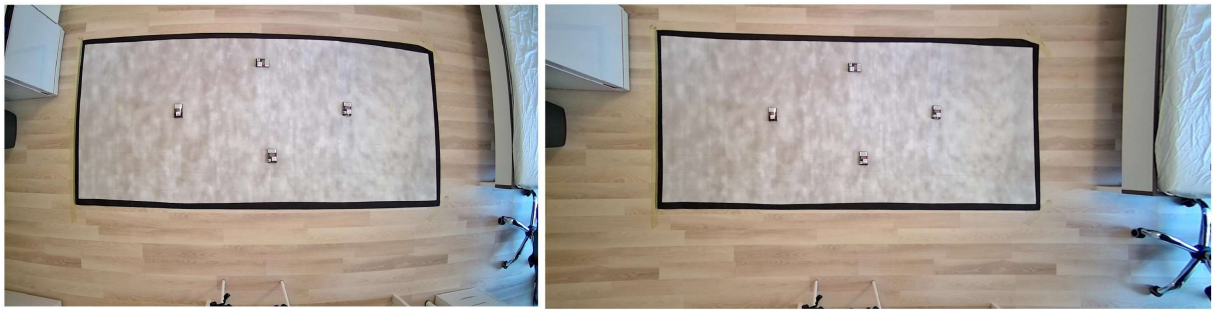
Κεφάλαιο 5: Μεθοδολογία Δημιουργίας PRM

Εικόνα 2: Χώρος κίνησης με εμπόδια

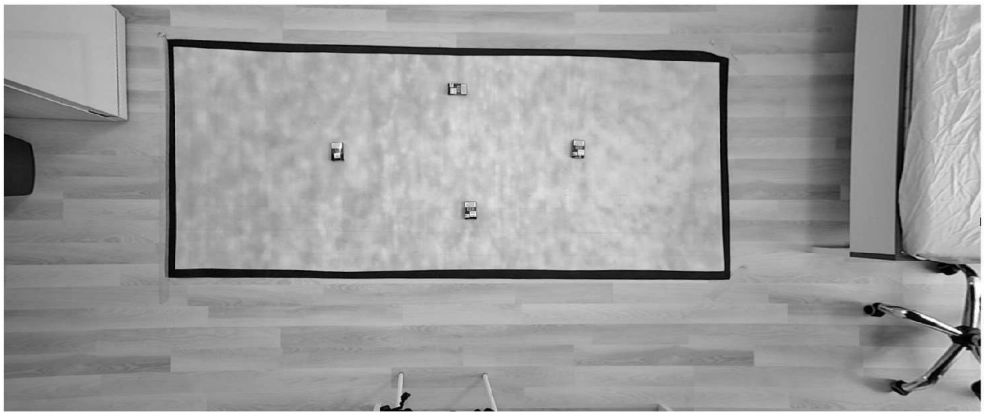
Για την υλοποίηση του συστήματος πρέπει να δημιουργηθεί ο κατάλληλος χώρος κίνησης. Ως χώρος κίνησης επιλέχθηκε χαρτόνι άσπρου χρώματος σε σχήμα παραλληλόγραμμου με διαστάσεις 2,34m μήκος και 1,05m πλάτος. Έπειτα οι άκρες του χαρτονιού καλύφθηκαν με μαύρη κολλητική ταινία που σηματοδοτεί το τέλος του ελεύθερου χώρου(Εικόνα 2). Ως εμπόδια χρησιμοποιήθηκαν μικρά παραλληλόγραμμα κουτιά από χαρτόνι με διαστάσεις 5cm πλάτος και 8cm μήκος(Εικόνα 2).

Για την λήψη της φωτογραφίας η κάμερα πρέπει να στερεωθεί σε κατάλληλο ύψος ώστε να μπορεί να βλέπει ολόκληρο το χώρο κίνησης. Για αυτό τον λόγο η κάμερα στερεώθηκε σε ύψος 2,5m από το χαρτόνι με τον φακό της κάμερας να βρίσκεται κάθετα με το κέντρο του χαρτονιού.

Η λήψη της φωτογραφίας γίνεται αυτόματα μέσω του προγράμματος Matlab. Μέσω του Matlab υπάρχει η δυνατότητα να γίνει λήψη φωτογραφίας από κάμερα που είναι συνδεδεμένη με τον Η/Υ με USB και να αποθηκευτεί για περαιτέρω επεξεργασία. Ένα από τα προβλήματα που παρουσιάστηκαν είναι ότι κατά την διαδικασία της λήψης της φωτογραφίας η κάμερα παραμορφώνει την εικόνα. Όπως φαίνεται και στην εικόνα 3 αριστερά στην αρχική εικόνα ο χώρος κίνησης δείχνει να έχει καμπυλότητα. Για την επίλυση αυτού του προβλήματος έγινε βαθμονόμιση της κάμερας. Όπως φαίνεται και στην εικόνα 3 δεξιά η καμπυλότητα δεν υπάρχει μετά την βαθμονόμιση της κάμερας.

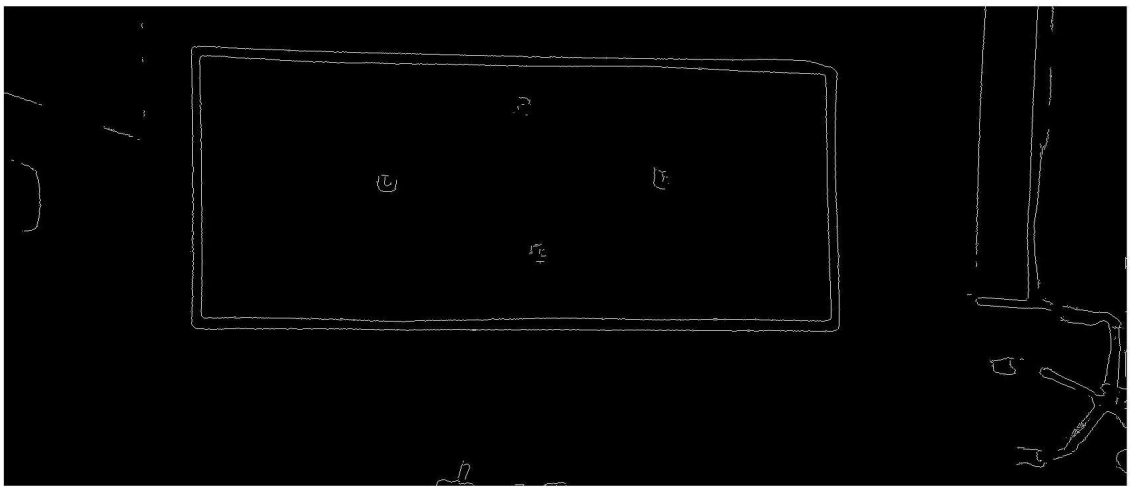


Εικόνα 3: Αριστερά: Αρχική εικόνα, Δεξιά: Εικόνα μετά από βαθμονόμιση κάμερας

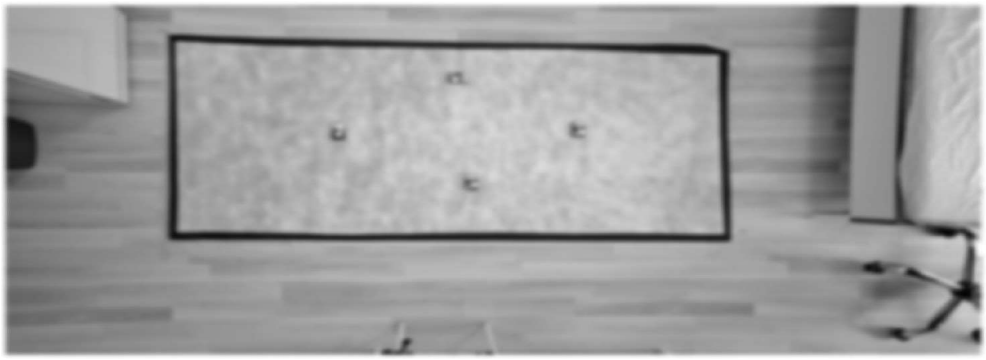


Εικόνα 4: Ασπρόμαυρη εικόνα

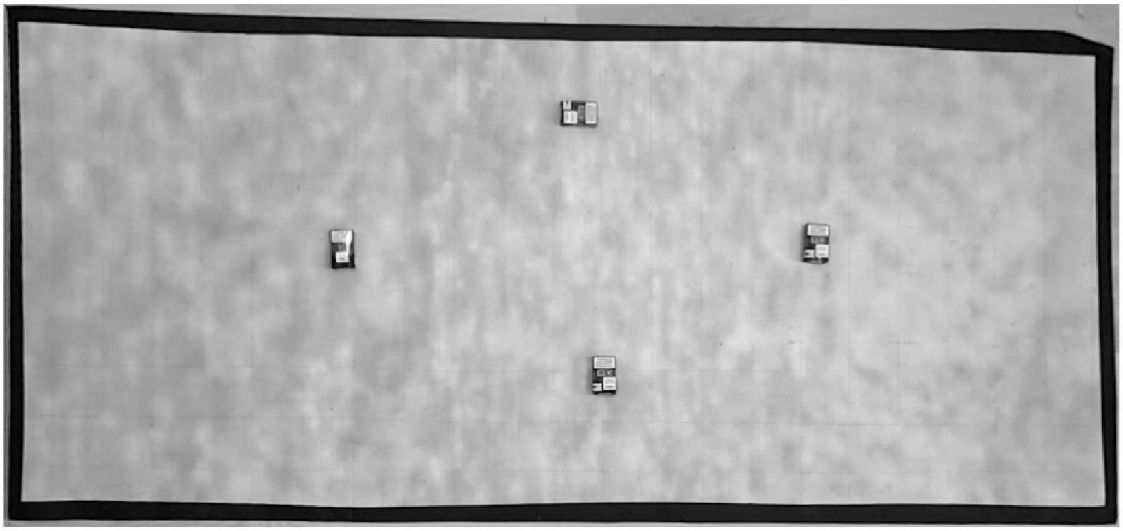
Για την σωστή μέτρηση των αποστάσεων από την μέθοδο PRM θα πρέπει να αφαιρεθούν από την εικόνα οι περιττές πληροφορίες που περιέχει. Αυτό επιτυγχάνεται μετατρέποντας την εικόνα αρχικά από έγχρωμη σε εικόνα βαθμίδων του γκρι(Εικόνα 4). Στην συνέχεια εξάγονται οι ακμές της εικόνες(Εικόνα 5) με την χρήση του αλγορίθμου sobel, αφού πρώτα εφαρμοστεί στην εικόνα φίλτρο Gauss με τυπική απόκλιση 4,6.



Εικόνα 5: Εξαγωγή ακμών εικόνας

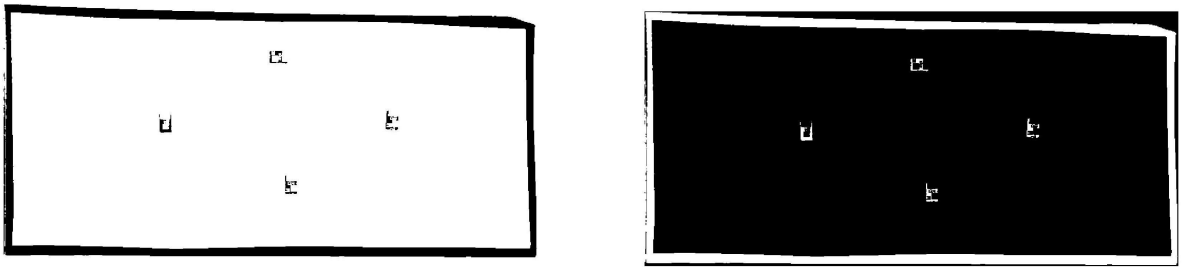


Εικόνα 6: Εφαρμογή φίλτρου Gauss

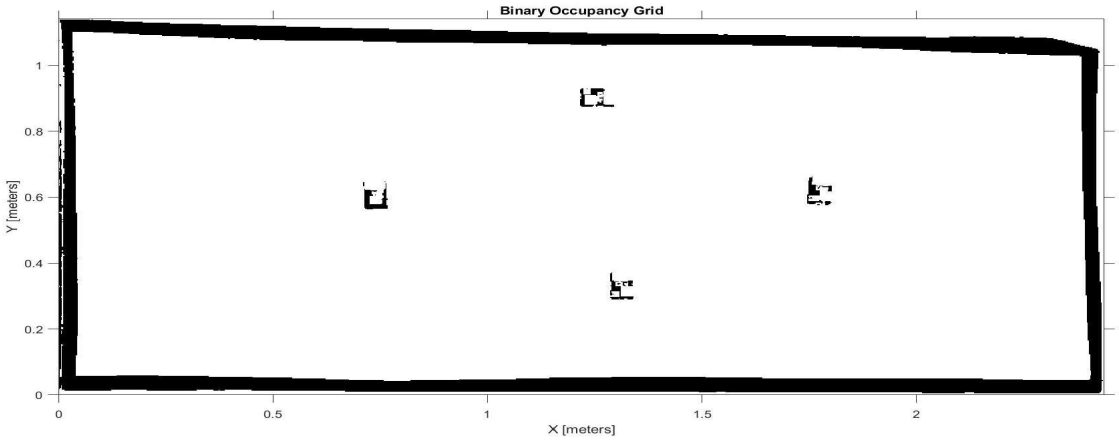


Εικόνα 7: Αποκοπή εικόνας

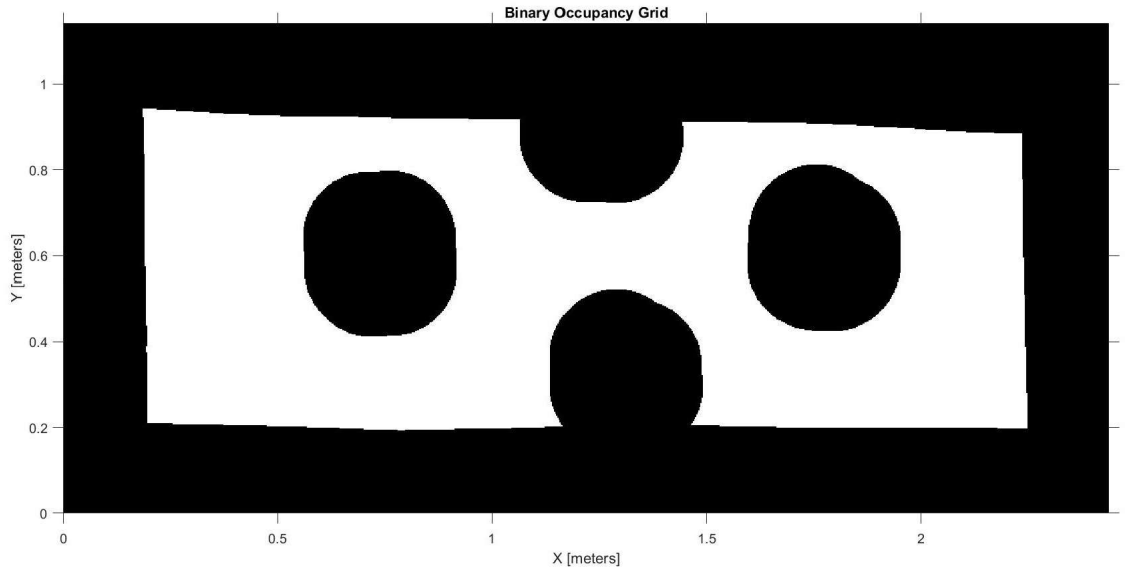
Η εξαγωγή των ακμών είναι απαραίτητη ώστε να γίνει αποκοπή της εικόνας έτσι ώστε να φαίνεται μόνο ο χώρος κίνησης του ρομπότ(Εικόνα 7). Στην συνέχεια η εικόνα μετατρέπεται σε ασπρόμαυρη(Εικόνα 8 αριστερά) και έπειτα γίνεται αντιστροφή των χρωμάτων(Εικόνα 8 δεξιά). Η τελευταία εικόνα χρησιμοποιείται για να υπολογιστεί το binary occupancy grid(Εικόνα 9). Το binary occupancy grid είναι μια ασπρόμαυρη εικόνα που περιέχει τις διαστάσεις του χώρου κίνηση καθώς και τις συντεταγμένες των εμποδίων.



Εικόνα 8: Αριστερά: Ασπρόμαυρη εικόνα, Δεξιά: Αντιστροφή χρωμάτων



Εικόνα 9: Binary Occupancy Grid

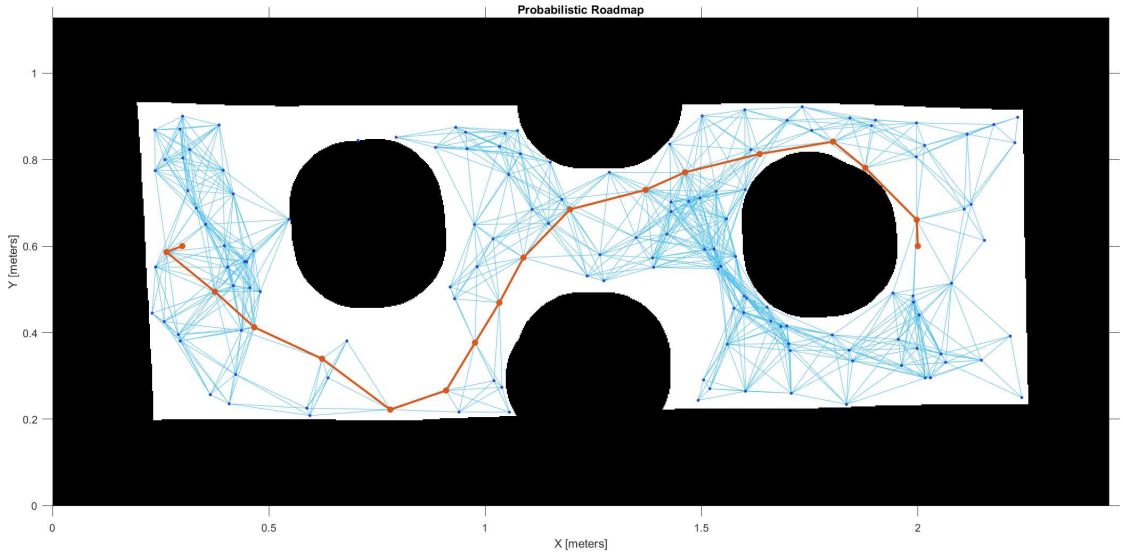


Εικόνα 10: Διόγκωση Occupancy Grid

Έχοντας δημιουργήσει το binary occupancy grip μπορεί να εφαρμοστεί η μέθοδος PRM. Για να δημιουργηθεί όμως διαδρομή όπου το ρομπότ δεν θα συγκρουστεί με κάποιο εμπόδιο χρειάζεται να προστεθούν και οι διαστάσεις του ρομπότ. Αυτό επιτυγχάνεται κάνοντας διόγκωση του binary occupancy grip(Εικόνα 10). Η διαδικασία της διόγκωσης προσθέτει στις διαστάσεις των εμποδίων αυτές του ρομπότ. Τέλος προστίθενται τα σημεία αφετηρίας και τερματισμού και εφαρμόζεται η μέθοδος PRM όπου υπολογίζει την διαδρομή που θα ακολουθήσει το ρομπότ(Εικόνα 11). Στον Πίνακα 1 φαίνεται ένα παράδειγμα της εξόδου της μεθόδου. Η μέθοδος υπολογίζει τις συντεταγμένες των σημείων, πάνω στον χώρο κίνησης, όπου πρέπει να περάσει το ρομπότ. Το πρώτο σημείο είναι η αφετηρία και το τελευταίο ο τερματισμός. Τα ενδιάμεσα σημεία είναι κόμβοι όπου το ρομπότ θα πρέπει να αλλάξει την κατεύθυνσή του. Η διαδρομή του ρομπότ θεωρείται επιτυχημένη αν καταφέρει να πλησιάσει το σημείο του τερματισμού εντός περιμέτρου 10cm.

Πίνακας 1: Έξοδος PRM

X	Ψ
0.3	0.6
0.329419	0.686525
0.667085	0.841252
1.303202	0.699780
1.835447	0.835815
2.027467	0.634011
2	0.6



Εικόνα 11: Εφαρμογή μεθόδου PRM

Για να αξιοποιηθούν οι συντεταγμένες των σημείων από τον Uno πρέπει να μετατραπούν σε μορφή που μπορούν να επεξεργαστούν. Στην κυκλωματική διάταξη του Uno δεν συμπεριλαμβάνεται κάποιος αισθητήρας, πέραν από τον encoder που έχουν τα μοτέρ. Αυτό σημαίνει ότι το μόνο που μπορεί να ελέγξει ο Uno είναι ο αριθμός περιστροφών της ρόδας του κάθε μοτέρ, οπότε και τα δεδομένα του Πίνακα 1 πρέπει να μετασχηματιστούν από συντεταγμένες σημείων σε αριθμό περιστροφών.

Για να επιτευχθεί αυτό πρέπει πρώτα να υπολογιστεί το μήκος και η κλίση των ευθύγραμμων τμημάτων που ενώνουν τα διαδοχικά σημεία. Το μήκος μπορεί να υπολογιστεί από τον παρακάτω τύπο:

$$distance = sqrt[(x2 - x1)^2 + (y2 - y1)^2] \quad (1).$$

Οι μονάδες του αποτελέσματος της απόστασης είναι σε μέτρα. Η κλίση δίνεται από τον τύπο:

$$angle = arctan[(y2 - y1) / (x2 - x1)] \quad (2).$$

Το Matlab παρέχει την συνάρτηση *atan2d* που επιστρέφει το αποτέλεσμα απευθείας σε μοίρες με εύρος τιμών [-180, 180]. Για να γίνει μετατροπή του εύρος τιμών από [-180, 180] σε [0, 360] προστίθεται η τιμή 360 μόνο αν το αποτέλεσμα που επιστρέψει η *atan2d* είναι αρνητικό. Κλίση 0 μοιρών σηματοδοτεί ευθύγραμμο τμήμα παράλληλο με τον άξονα X του χώρου κίνησης και οι μοίρες αυξάνονται αριστερόστροφα.

Στην συνέχεια γίνεται η μετατροπή της απόστασης και της γωνίας σε παλμούς. Για να γίνει η μετατροπή σε παλμούς πρέπει να είναι γνωστά η περίμετρος της ρόδας και ο αριθμός των παλμών μιας πλήρους περιστροφής. Από τα τεχνικά χαρακτηριστικά του μοτέρ είναι γνωστό πως μία περιστροφή του μοτέρ είναι 11 παλμοί και ο μειωτήρας έχει σχέση 1:50. Αυτό σημαίνει πως μια πλήρη περιστροφή στη έξοδο του μειωτήρα είναι πενήντα περιστροφές του μοτέρ, άρα 11 * 50 = 550 παλμοί για μία πλήρη περιστροφή της ρόδας. Στην συνέχεια αποδείχθηκε πως το αποτέλεσμα των 550 παλμών είναι εσφαλμένο και βρέθηκε πειραματικά πως ο αριθμός των παλμών για μία πλήρη περιστροφή είναι 600. Η πειραματική

διαδικασία εύρεσης των παλμών περιγράφεται στο κεφάλαιο 7. Από τα τεχνικά χαρακτηριστικά της ρόδας είναι γνωστό πως η διάμετρός της είναι 65mm. Στη συνέχεια μπορεί να υπολογιστεί η περίμετρος της ρόδας, σε μέτρα, από τον τύπο:

$$περίμετρος = π * διάμετρος \quad (3).$$

Γνωρίζοντας την περίμετρο και τους παλμούς μιας πλήρους περιστροφής η μετατροπή της απόστασης σε παλμούς γίνεται από τον τύπο:

$$παλμοί = (απόσταση/περίμετρος) * παλμοί μίας περιστροφής \quad (4).$$

Για τον υπολογισμό των παλμών της γωνίας πρώτα πρέπει να βρεθεί το μήκος του τόξου, σε μέτρα, που θα διανύσει το ρομπότ. Αυτό δίνεται από τον τύπο:

$$τόξο = (π*πλάτος) * (γωνία/360) \quad (5).$$

Στην συνέχεια το μήκος του τόξου μετατρέπεται σε παλμούς με την χρήση του τύπου (4).

Για να μειωθεί ο κίνδυνος σύγκρουσης του ρομπότ με κάποιο εμπόδιο αναπτύχθηκε κώδικας διόρθωσης της γωνίας του ρομπότ. Το πρώτο βήμα είναι ο αλγόριθμος να εντοπίσει την θέση του ρομπότ εντός του χώρου κίνησης, να μετρήσει την γωνία που έχει το ρομπότ κι αν το σφάλμα είναι μεγαλύτερο από κάποιο όριο τότε το ρομπότ πρέπει να διορθώσει την γωνία του. Ο εντοπισμός του ρομπότ γίνεται με την βοήθεια δύο κύκλων ένας ανοιχτού χρώματος και ένας σκούρου χρώματος(Εικόνα 12). Η χρήση κύκλων διαφορετικών χρωμάτων είναι απαραίτητη ώστε ο αλγόριθμος να καταλάβει την φορά του ρομπότ. Ο αλγόριθμος ψάχνει πρώτα για τον ανοιχτόχρωμο κύκλο και έπειτα για τον σκουρόχρωμο. Αφού έχουν βρεθεί οι δύο κύκλοι υπολογίζεται το ευθύγραμμο τμήμα που ενώνει τα κέντρα τους. Στην συνέχεια ο αλγόριθμος υπολογίζει την κλίση του ευθύγραμμου τμήματος και το συγκρίνει με την κλίση που θα πρέπει να έχει το ρομπότ με βάση την έξοδο της μεθόδου PRM για το συγκεκριμένο σημείο. Αν βρεθεί απόκλιση μεγαλύτερη ή ίση των 3,5 μοιρών τότε το ρομπότ πρέπει να διορθώσει την γωνία του, οπότε υπολογίζεται με βάση την καινούργια θέση του ρομπότ η φορά περιστροφής και η γωνία που χρειάζεται το ρομπότ ώστε να μειώσει το σφάλμα και στέλνονται μέσω Bluetooth. Το όριο των 3,5 μοιρών βρέθηκε πειραματικά. Ο αριθμός των 3,5 μοιρών υπολογίστηκε από δύο παράγοντες όπου αθροιστικά έδωσαν το αποτέλεσμα αυτό. Ο πρώτος παράγοντας είναι κατά των υπολογισμό της κλίσης του ευθύγραμμου τμήματος που ενώνει τους δύο κύκλους, η μέγιστη απόκλιση που παρατηρήθηκε ήταν 0,5 μοίρες. Ο δεύτερος παράγοντας είναι η διόρθωση της γωνίας όταν αυτή χρειάζεται και ταυτόχρονα η αποφυγή του αλγορίθμου να προσπαθεί να διορθώνει συνέχεια την θέση του ρομπότ με αποτέλεσμα η ολοκλήρωση της διαδρομής να χρειάζεται πολύ ώρα ή στην χειρότερη περίπτωση ο αλγόριθμος να μπει σε ατέρμονα βρόχο με αποτέλεσμα η διαδρομή να μην ολοκληρωθεί πότε. Παρατηρήθηκε ότι ορίζοντας τον δεύτερο παράγοντα στο όριο των 3 μοιρών ικανοποιούνται οι προηγούμενες προϋποθέσεις.

Εικόνα 12: Μετατροπή ρομπότ για εντοπισμό θέσης

Κεφάλαιο 6: Τεχνική ανάλυση κώδικα MATLAB

Στο κεφάλαιο αυτό γίνεται επεξήγηση του κώδικα που αναπτύχθηκε στο MATLAB. Για την ανάπτυξη του κώδικα χρησιμοποιήθηκαν οι βιβλιοθήκες που παρέχονται με μια τυπική εγκατάσταση του αναπτυξιακού. Για την υλοποίηση του αλγορίθμου PRM χρησιμοποιήθηκαν ρουτίνες από την βιβλιοθήκη robotics που προσφέρει το MATLAB. Οι ρουτίνες αυτές έχουν το πρόθεμα robotics. στο όνομά τους. Ο αλγόριθμος που αναπτύχθηκε είναι διαδικαστικός. Για την υλοποίηση του κώδικα, εκτός από τις ρουτίνες που προσφέρει το MATLAB, αναπτύχθηκαν ρουτίνες οι οποίες βοηθούν στην λύση προβλημάτων σωστής καθοδήγησης του ρομπότ. Παρακάτω γίνεται επεξήγηση των ρουτινών.

6.1 Ρουτίνες MATLAB:

6.1.1 Y = snapshot(X)

Η ρουτίνα snapshot χρησιμοποιείται για την λήψη φωτογραφίας.

Y: Επιστρεφόμενη τιμή της ρουτίνας - η εικόνα που λήφθηκε.

X: Είσοδος στην ρουτίνα - η κάμερα που θα χρησιμοποιηθεί για την λήψη της φωτογραφίας.

6.1.2 UndImage = undistortImage(Image, cameraParams, 'OutputView', 'Valid)

Η ρουτίνα undistortImage χρησιμοποιείται για την αφαίρεση της παραμόρφωσης της εικόνας που εισάγει η κάμερα.

UndImage: Επιστρεφόμενη τιμή της ρουτίνας - η εικόνα χωρίς παραμόρφωση.

Image: Είσοδος στην ρουτίνα - η αρχική εικόνα με παραμόρφωση.

cameraParams: Είσοδος στην ρουτίνα - πίνακας μεταβλητών με τα χαρακτηριστικά της κάμερας .

'OutputView', 'Valid': Είσοδος στην ρουτίνα - ρυθμίσεις για την απεικόνιση της επιστρεφόμενης τιμής.

6.1.3 GreyImage = rgb2gray(Image)

Η ρουτίνα rgb2gray χρησιμοποιείται για την μετατροπή έγχρωμης εικόνας του χρωματικού χώρου rgb(red, green, blue) σε εικόνα χρωματικού χώρου διαβαθμίσεων του γκρι.

GreyImage: Επιστρεφόμενη τιμή της ρουτίνας - εικόνα σε διαβαθμίσεις του γκρι.

Image: Είσοδος στην ρουτίνα - εικόνα σε απόχρωση rgb.

6.1.4 imageBlur = imgaussfilt(image, X)

Η ρουτίνα imgaussfilt χρησιμοποιείται για την εφαρμογή φίλτρου Gauss σε εικόνα.

imageBlur: Επιστρεφόμενη τιμή ρουτίνας - εικόνα που έχει εφαρμοστεί φίλτρο Gauss.

image: Είσοδος στην ρουτίνα - εικόνα που θα εφαρμοστεί το φίλτρο Gauss.

X: Είσοδος στην εικόνα - η τυπική απόκλιση για την εφαρμογή του φίλτρου.

6.1.5 imageEdges = edge(image, 'sobel')

Η ρουτίνα edge χρησιμοποιείται για την εύρεση ακμών εικόνας.

imageEdges: Επιστρεφόμενη τιμή ρουτίνας - οι ακμές που εντοπίστηκαν.

image: Είσοδος στην ρουτίνα - εικόνα για εντοπισμό ακμών.

'sobel': Είσοδος στην ρουτίνα - τρόπος εντοπισμού ακμών που θα χρησιμοποιήσει το MATLAB .

6.1.6 cropedImage = imcrop(image, X, Y, WIDTH, HEIGHT)

Η ρουτίνα imcrop χρησιμοποιείται για την περικοπή της εικόνας σε επιθυμητό μέγεθος.

cropedImage: Επιστρεφόμενη τιμή ρουτίνας, κομμένη εικόνα.

image: Αρχική εικόνα.

X: Είσοδος στην ρουτίνα - το αρχικό σημείο X του τετραγώνου περικοπής.

Y: Είσοδος στην ρουτίνα - το αρχικό σημείο Y του τετραγώνου περικοπής.

WIDTH: Είσοδος στην ρουτίνα - το πλάτος του τετραγώνου περικοπής.

HEIGHT: Είσοδος στην ρουτίνα - το ύψος του τετραγώνου περικοπής.

6.1.7 imageBin = imbinarize(image)

Η ρουτίνα imbinarize χρησιμοποιείται για την μετατροπή εικόνας διαβαθμίσεων του γκρι σε ασπρόμαυρη.

imageBin: Επιστρεφόμενη τιμή ρουτίνας - ασπρόμαυρη εικόνα.

image: Είσοδος στην ρουτίνα - εικόνα διαβαθμίσεων του γκρι.

6.1.8 imageComp = imcomplement(image)

Η ρουτίνα imcomplement χρησιμοποιείται για την αντιστροφή χρωμάτων ασπρόμαυρης εικόνας.

imageComp: Επιστρεφόμενη τιμή ρουτίνας - ασπρόμαυρη εικόνα με αντεστραμμένα χρώματα.

image: Είσοδος στην ρουτίνα - ασπρόμαυρη εικόνα.

6.1.9 grid = robotics.BinaryOccupancyGrid(map, RES)

Η ρουτίνα robotics.BinaryOccupancyGrid χρησιμοποιείται για την μετατροπή ασπρόμαυρης εικόνας σε δυαδικό πλέγμα πληρότητας(Binary Occupancy Grid). Στο πλέγμα αυτό το κάθε κελί έχει την τιμή 1 εάν είναι κατειλημμένο και την τιμή 0 εάν είναι ελεύθερο.

grid: Επιστρεφόμενη τιμή ρουτίνας - πλέγμα με τα ελεύθερα και κατειλημμένα κελιά.

map: Είσοδος στην ρουτίνα - ασπρόμαυρη εικόνα για την δημιουργία του πλέγματος.

RES: Είσοδος στην ρουτίνα - πόσα κελιά ανά μέτρο να έχει το πλέγμα.

6.1.10 inflate(gridInflated, radius)

Η ρουτίνα inflate χρησιμοποιείται για να γίνει διόγκωση των κατειλημμένων κελιών του πλέγματος πληρότητας.

Η ρουτίνα δεν έχει επιστρεφόμενη τιμή.

gridInflated: Είσοδος στην ρουτίνα - πλέγμα πληρότητας όπου θα γίνει η διόγκωση των κατειλημμένων κελιών.

radius: Είσοδος στην ρουτίνα - τιμή μεγέθους διόγκωσης κατειλημμένων κελιών.

6.1.11 path = findpath(prm, startLocation, endLocation)

Η ρουτίνα findpath χρησιμοποιείται για την εύρεση της διαδρομής που θα ακολουθήσει το ρομπότ για να φτάσει από το σημείο της αφετηρίας στο σημείο του τερματισμού.

path: Επιστρεφόμενη τιμή ρουτίνα - πίνακας συντεταγμένων Χ,Ψ με τα σημεία όπου πρέπει να περάσει το ρομπότ.

prm: Είσοδος στην ρουτίνα - πίνακας μεταβλητών με δεδομένα για την υλοποίηση του αλγορίθμου prm. Ο πίνακας prm δημιουργείται ως εξής:

prm = robotics.PRM - δημιουργία αντικειμένου.

prm.Map = gridInflated - καθορισμός του occupancy grid που θα χρησιμοποιηθεί.

prm.NumNodes = 150 - καθορισμός του αριθμού των τυχαίων κόμβων.

prm.ConnectionDistance = 0.3 - καθορισμός της μέγιστης απόστασης για διασύνδεση κόμβων.

startLocation: Είσοδος στην ρουτίνα - συντεταγμένες Χ,Ψ του σημείου της αφετηρίας.

endLocation: Είσοδος στην ρουτίνα - συντεταγμένες Χ,Ψ του σημείου τερματισμού.

6.1.12 fprintf(dev, format, data)

Η τυπική χρήση της ρουτίνας fprintf είναι για εγγραφή δεδομένων σε αρχείο. Το MATLAB παρέχει την δυνατότητα να χρησιμοποιηθεί η ίδια ρουτίνα για αποστολή δεδομένων μέσω bluetooth.

Η ρουτίνα δεν έχει επιστρεφόμενη τιμή.

dev: Είσοδος στην ρουτίνα - αντικείμενο που χαρακτηρίζει την συσκευή bluetooth για την επικοινωνία. Το αντικείμενο dev αποτελείται από:

dev = hc.connect - σύνδεση με την συσκευή hc.

dev.BytesAvailableFcnMode = 'terminator' - καθορισμός σηματοδότησης τέλους μετάδοσης - χαρακτήρας.

dev.Terminator = 'LF' - ορισμός χαρακτήρα που σηματοδοτεί το τέλος μετάδοσης -

χαρακτήρας νέας γραμμής.

Το αντικείμενο hc αποτελείται από:

hc = Mybt - δημιουργία αντικειμένου bluetooth.

hc.deviceName = 'HC-06' - όνομα συσκευής bluetooth.

format: Είσοδος στην ρουτίνα - σηματοδοτεί τον τύπο των δεδομένων που θα σταλούν, '%c' σηματοδοτεί χαρακτήρα, '%i' σηματοδοτεί ακέραιο αριθμό.

data: Είσοδος στην ρουτίνα - τα δεδομένα που θα σταλούν.

6.1.13 data = fscanf(dev, format)

Όπως και με την ρουτίνα fprintf έτσι και η ρουτίνα fscanf η τυπική της χρήση είναι για ανάγνωση δεδομένων από αρχείο. Το MATLAB παρέχει την δυνατότητα να χρησιμοποιηθεί η ρουτίνα fscanf για την ανάγνωση δεδομένων που λήφθηκαν μέσω bluetooth

data: Επιστρεφόμενη τιμή ρουτίνας - τα δεδομένα που λήφθηκαν μέσω bluetooth.

dev: Είσοδος στην ρουτίνα - αντικείμενο που χαρακτηρίζει την συσκευή bluetooth για την επικοινωνία. Το αντικείμενο dev αποτελείται από τα ίδια στοιχεία όπως περιγράφεται στο 6.1.12.

format: Είσοδος στην ρουτίνα - σηματοδοτεί τον τύπο των δεδομένων που θα ληφθούν, '%c' σηματοδοτεί χαρακτήρα, '%i' σηματοδοτεί ακέραιο αριθμό.

6.1.14 [centersCircles, radii] = imfindcircles(image, [10, 14], 'ObjectPolarity', 'dark', 'Sensitivity', 0.94]

Η συνάρτηση imfindcircles χρησιμοποιείται για να εντοπιστούν οι κύκλοι που έχουν τοποθετηθεί πάνω στο ρομπότ ώστε να γίνει ο εντοπισμός του, στον χώρο κίνησης.

centersCircles: Επιστρεφόμενη τιμή ρουτίνας - συντεταγμένες X,Y των pixels του κέντρου του κύκλου που εντοπίστηκε.

radii: Επιστρεφόμενη τιμή ρουτίνας - η ακτίνα του κύκλου που εντοπίστηκε σε pixels.

image: Είσοδος στην ρουτίνα - η εικόνα όπου θα γίνει η εύρεση των κύκλων.

[10, 14]: Είσοδος στην ρουτίνα - εύρος τιμών, σε pixels, επιθυμητών ακτίνων κύκλου

'ObjectPolarity': Είσοδος στην ρουτίνα - παράμετρος ρύθμισης της ρουτίνας, υπάρχει η δυνατότητα η ρουτίνα να εντοπίσει μόνο σκουρόχρωμους κύκλους ορίζοντας την τιμή 'dark' ή μόνο ανοιχτόχρωμους κύκλους ορίζοντας την τιμή 'bright'.

'dark': Είσοδος στην ρουτίνα - τιμή ρύθμισης για το χρώμα κύκλου που θα εντοπίσει η ρουτίνα, πρέπει να είναι μετά το 'ObjectPolarity', πιθανές τιμές: 'dark', 'bright'.

'Sensitivity': Είσοδος στην ρουτίνα - παράμετρος ρύθμισης για την ευαισθησία εντοπισμού κύκλου της ρουτίνας.

0.94: Είσοδος στην ρουτίνα - τιμή ρύθμισης της ευαισθησίας εντοπισμού της ρουτίνας, πρέπει να είναι μετά το 'Sensitivity', πιθανές τιμές: [0, 1.00].

6.1.15 roundValue = round(value)

Η ρουτίνα round χρησιμοποιείται για να γίνει στρογγυλοποίηση των δεκαδικών ψηφίων, εφόσον υπάρχουν, προς τα πάνω στον πλησιέστερο ακέραιο αριθμό.

roundValue: Επιστρεφόμενη τιμή ρουτίνας - ο στρογγυλοποιημένος αριθμός.

value: Είσοδος στην ρουτίνα - αριθμός προς στρογγυλοποίηση.

6.1.16 imageCopy = copy(image)

Η ρουτίνα copy χρησιμοποιείται για να δημιουργηθεί ένα αντίγραφο του occupancy grid όπου θα γίνει η διόγκωση των εμποδίων.

imageCopy: Επιστρεφόμενη τιμή ρουτίνας - το αντίγραφο της εικόνας.

image: Είσοδος στην ρουτίνα - η εικόνα προς αντιγραφή.

6.1.17 Value = length(array)

Η ρουτίνα length χρησιμοποιείται για να βρεθεί το πλήθος των στοιχείων που έχει ένας μονοδιάστατος πίνακας.

Value: Επιστρεφόμενη τιμή ρουτίνας - το πλήθος των στοιχείων του πίνακα

array: Είσοδος στην ρουτίνα - ο μονοδιάστατος πίνακας.

6.1.18 pause(time)

Η ρουτίνα pause χρησιμοποιείται για να γίνει "διακοπή" του προγράμματος ώστε να υπάρχει αρκετός χρόνος για την λήψη και φόρτωση των φωτογραφιών.

Η ρουτίνα δεν έχει επιστρεφόμενη τιμή.

time: Είσοδος στην ρουτίνα - χρόνος "διακοπής" του προγράμματος.

6.2 Ρουτίνες που αναπτύχθηκαν

6.2.1 distance = calcDistance(path)

Η ρουτίνα calcDistance χρησιμοποιείται για να υπολογιστεί η απόσταση που έχουν μεταξύ τους δύο διαδοχικοί κόμβοι της διαδρομής.

distance: Επιστρεφόμενη τιμή ρουτίνας - πίνακας με τις αποστάσεις των κόμβων

path: Είσοδος στην ρουτίνα - πίνακας συντεταγμένων Χ,Ψ με τα σημεία όπου πρέπει να περάσει το ρομπότ.

6.2.2 degrees = calcDegrees(path)

Η ρουτίνα calcDegrees χρησιμοποιείται για να υπολογιστεί η γωνία που έχουν μεταξύ τους δύο διαδοχικοί κόμβοι της διαδρομής.

degrees: Επιστρεφόμενη τιμή ρουτίνας - πίνακας με τις γωνίες των κόμβων.

path: Είσοδος στην ρουτίνα - πίνακας συντεταγμένων Χ,Ψ με τα σημεία όπου πρέπει να περάσει το ρομπότ.

6.2.3 [estimateCenter, estimateDeg] = EstimateRobotPosition(image, cameraParams, whiteRow, whiteCol)

Η ρουτίνα EstimateRobotPosition χρησιμοποιείται για τον εντοπισμό της θέσης του ρομπότ στον χώρο κίνησης.

estimateCenter: Επιστρεφόμενη τιμή ρουτίνας - συντεταγμένες Χ,Ψ του ρομπότ στον χώρο κίνησης.

estimateDeg: Επιστρεφόμενη τιμή ρουτίνας - κλίση του ρομπότ στον χώρο κίνησης.

image: Είσοδος στην ρουτίνα - φωτογραφία του χώρου κίνησης με την νέα θέση του ρομπότ.

cameraParams: Είσοδος στην ρουτίνα - πίνακας μεταβλητών με τα χαρακτηριστικά της κάμερας .

whiteRow: Είσοδος στην ρουτίνα - πίνακας τιμών όπου έχουν αποθηκευτεί οι τιμές Χ,Ψ των pixels των οριζόντιων ακμών που εντοπίστηκαν με την ρουτίνα 6.1.5 - χρησιμοποιείται για την περικοπή εικόνας.

whiteCol: Είσοδος στην ρουτίνα - πίνακας τιμών όπου έχουν αποθηκευτεί οι τιμές Χ,Ψ των pixels των κατακόρυφων ακμών που εντοπίστηκαν με την ρουτίνα 6.1.5 - χρησιμοποιείται για την περικοπή εικόνας.

6.2.4 [direction, turn] = calcTurn(degrees(i), robotPose)

Η ρουτίνα calcTurn χρησιμοποιείται για να υπολογιστούν η φορά και οι μοίρες που χρειάζεται να στρίψει το ρομπότ, εφόσον χρειάζεται, για να φτάσει στον επόμενο κόμβο.

direction: Επιστρεφόμενη τιμή ρουτίνας - κατεύθυνση στροφής.

turn: Επιστρεφόμενη τιμή ρουτίνας - μοίρες στροφής.

degrees(i): Είσοδος στην ρουτίνα - η κλίση της ευθείας που ενώνει δύο διαδοχικούς κόμβους.

robotPose: Είσοδος στην ρουτίνα - η κλίση του ρομπότ στον χώρο κίνησης.

6.2.5 arc = degreesToArc(turn, robotWidth)

Η ρουτίνα degreesToArc χρησιμοποιείται για να υπολογιστεί το μήκος του τόξου της στροφής που χρειάζεται να κάνει το ρομπότ για να μετατραπεί στην συνέχεια σε παλμούς.

arc: Επιστρεφόμενη τιμή ρουτίνας - το μήκος τόξου της στροφής.

turn: Είσοδος στην ρουτίνα - οι μοίρες που χρειάζεται να στρίψει το ρομπότ.

robotWidth: Είσοδος στην ρουτίνα - το μήκος της απόστασης που έχουν μεταξύ τους οι ρόδες του ρομπότ.

6.2.6 pulses = distanceToPulses(distance, pulsesPerRotation, wheelCircumference)

Η ρουτίνα distanceToPulses χρησιμοποιείται για να υπολογιστεί ο αριθμός των παλμών που χρειάζονται ώστε το ρομπότ να εκτελέσει την συγκεκριμένη κίνηση.

pulses: Επιστρεφόμενη τιμή ρουτίνας - ο αριθμός των παλμών που χρειάζονται για την κίνηση.

distance: Είσοδος στην ρουτίνα - η ζητούμενη απόσταση που χρειάζεται να διανύσει το ρομπότ.

pulsesPerRotation: Είσοδος στην ρουτίνα - ο αριθμός των παλμών που χρειάζονται ώστε η ρόδα του ρομπότ να κάνει μία πλήρη περιστροφή.

wheelCircumference: Είσοδος στην ρουτίνα - η περίμετρος της ρόδας του ρομπότ.

6.2.7 correctPose(robotPose, degrees(i), robotWidth, pulsesPerRotation, wheelCircumference, dev)

Η ρουτίνα correctPose χρησιμοποιείται για να μειωθεί το σφάλμα της στροφής του ρομπότ.

Η ρουτίνα δεν έχει επιστρεφόμενη τιμή.

robotPose: Είσοδος στην ρουτίνα - η γωνία του ρομπότ στον χώρο κίνησης.

degrees(i): Είσοδος στην ρουτίνα - η γωνία μεταξύ δύο διαδοχικών κόμβων.

robotWidth: Είσοδος στην ρουτίνα - το μήκος της απόστασης που έχουν μεταξύ τους οι ρόδες του ρομπότ.

pulsesPerRotation: Είσοδος στην ρουτίνα - ο αριθμός των παλμών που χρειάζονται ώστε η ρόδα του ρομπότ να κάνει μία πλήρη περιστροφή.

wheelCircumference: Είσοδος στην ρουτίνα - η περίμετρος της ρόδας του ρομπότ.

dev: Είσοδος στην ρουτίνα - αντικείμενο που χαρακτηρίζει την συσκευή bluetooth για την επικοινωνία. Το αντικείμενο dev αποτελείται από τα ίδια στοιχεία όπως περιγράφεται στο 6.1.12.

6.3 Περιγραφή global μεταβλητών

Κατά την ανάπτυξη του κώδικα δημιουργήθηκαν global μεταβλητές για την ευκολότερη ανάπτυξή του. Παρακάτω γίνετε η περιγραφή αυτών των μεταβλητών.

6.3.1 Μεταβλητές μηχανικών χαρακτηριστικών ρομπότ

Υπάρχουν τέσσερις μεταβλητές που ορίζουν τα μηχανικά χαρακτηριστικά του ρομπότ. Οι μεταβλητές αυτές είναι:

- *wheelCircumference:* περίμετρος ρόδας ρομπότ.
- *robotWidth:* μήκος απόστασης που έχουν οι ρόδες του ρομπότ μεταξύ τους.
- *pulsesPerRotation:* αριθμός παλμών μιας πλήρους περιστροφής της ρόδας.

- *robotRadius:* ακτίνα του ρομπότ.

Οι τρεις πρώτες μεταβλητές, wheelCircumference, robotWidth, και pulsesPerRotation, χρησιμοποιούνται για να γίνει η μετατροπή της απόστασης σε παλμούς. Η τέταρτη μεταβλητή, robotRadius, χρησιμοποιείται για την διόγκωση του χάρτη κατά την δημιουργία του PRM. Με τον όρο ακτίνα του ρομπότ εννοείτε η ευθεία που ενώνει κατά μήκος το κέντρο του σασί του ρομπότ μέχρι την άκρη διότι το μήκος του ρομπότ είναι μεγαλύτερο από το πλάτος.

6.3.2 Μεταβλητές σύνδεσης εξωτερικών περιφερειακών

Για την λειτουργία του συστήματος το πρόγραμμα χρειάζεται να έχει πρόσβαση σε εξωτερικά περιφερειακά όπως η κάμερα, για την λήψη φωτογραφιών, και η διεπαφή bluetooth για την επικοινωνία με τον arduino. Για την σύνδεση με την κάμερα δημιουργήθηκε η μεταβλητή *cam* η οποία αρχικοποιείται ως εξής: *cam = webcam.* Στην συνέχεια ρυθμίζεται η επιθυμητή ανάλυση της κάμερας με την μεταβλητή *cam.resolution = '1280x720'.* Οι έγκυρες τιμές για την ανάλυση της κάμερας είναι: 1280x720, 640x480 και 640x360. Επιλέχθηκε η τιμή 1280x720 διότι παρέχει την μεγαλύτερη ανάλυση.

Για να γίνει η σύνδεση με την διεπαφή bluetooth χρειάζονται δύο μεταβλητές. Αρχικά γίνεται η αρχικοποίηση του αντικειμένου με την μεταβλητή *hc=MyBt.* Στην συνέχεια γίνεται ο ορισμός του ονόματος της συσκευής με την οποία θα γίνει η σύνδεση με την μεταβλητή *hc.deviceName = "HC-06".* Έπειτα χρειάζεται να γίνει η σύζευξη της διεπαφής bluettoth του Η/Υ με την διεπαφή bluettoth του arduino. Αυτό επιτυγχάνεται χρησιμοποιώντας την μεταβλητή *dev = hc.connect* όπου αποθηκεύονται οι πληροφορίες της σύνδεσης. Με τις μεταβλητές *dev.BytesAvailableFcnMode = 'terminator'* και *dev.Terminator ='LF'* ορίζεται ο τρόπος σηματοδότησης της λήξης μετάδοσης δεδομένων. Με την *dev.BytesAvailableFcnMode = 'terminator'* ορίζεται ότι ο τρόπος λήξης θα είναι χαρακτήρας και με την *dev.Terminator ='LF'* ορίζεται ότι ο χαρακτήρας θα είναι ο χαρακτήρας νέας γραμμής.

6.3.3 Μεταβλητές επεξεργασίας εικόνας

Στο πρόγραμμα χρησιμοποιούνται μεταβλητές για την επεξεργασία της φωτογραφίας που θα χρησιμοποιηθεί για την δημιουργία του PRM. Η κύρια μεταβλητή είναι η μεταβλητή *map* όπου αποθηκεύεται η φωτογραφία που θα τραβήξει η κάμερα και στην συνέχεια όποια επεξεργασία γίνεται στην εικόνα αποθηκεύεται ξανά στην μεταβλητή *map.* Στο πρόγραμμα υπάρχουν και κάποιες βοηθητικές μεταβλητές για την περικοπή της εικόνας. Αυτές είναι οι *mapgrey, mapblur, edgemap* και *cropedMap.* Η μεταβλητή *mapgrey* χρησιμοποιείται για να αποθηκευτεί η μετατροπή της εικόνας από τον χρωματικό χώρο rgb στον χρωματικό χώρο διαβαθμίσεων του γκρι. Στο επόμενο βήμα εφαρμόζεται φίλτρο Gauss και η νέα εικόνα αποθηκεύεται στην μεταβλητή *mapblur.* Η εφαρμογή του φίλτρου Gauss γίνεται για την καλύτερη απόδοση της ρουτίνας εντοπισμού ακμών όπου τα αποτελέσματα αποθηκεύονται στην μεταβλητή *edgemap.* Στην μεταβλητή cropedMap αποθηκεύεται η κομμένη εικόνα όπου στην συνέχεια η μεταβλητή *cropedMap* αποθηκεύεται ξανά πίσω στην μεταβλητή *map.* Σε αυτό το σημείο η μεταβλητή *map* περιέχει το κομμάτι της αρχικής φωτογραφίας που απεικονίζει τον χώρο κίνησης. Έπειτα γίνεται η μετατροπή της εικόνας στον χρωματικό χώρο διαβαθμίσεων του γκρι, στην συνέχεια η εικόνα μετατρέπεται σε ασπρόμαυρη και τέλος

γίνεται η αντιστροφή των χρωμάτων της ασπρόμαυρης εικόνας. Σε κάθε στάδιο η κάθε μετατροπή αποθηκεύεται πίσω στην μεταβλητή *map.* Στην συνέχεια γίνετε η δημιουργία του δυαδικού πλέγματος πληρότητας με την βοήθεια της συνάρτησης *robotics.BinaryOccupancyGrid* και αποθηκεύεται πίσω στην μεταβλητή *map*. Έπειτα δημιουργείται ένα αντίγραφο της μεταβλητής *map* με το όνομα *mapInflated.* Τέλος στην μεταβλητή *mapInflated* αποθηκεύεται το διογκωμένο πλέγμα πληρότητας που δημιουργήθηκε με την βοήθεια της συνάρτησης *inflate.* Η μεταβλητή *mapInflated* είναι αυτή που θα χρησιμοποιηθεί για την δημιουργία του αλγορίθμου PRM. Υπάρχουν δύο επιπλέον μεταβλητές που χρησιμοποιούνται για την περικοπή της εικόνας. Αυτές είναι οι *whiteCol και whiteRow.* Στην μεταβλητή *whiteCol* αποθηκεύονται οι συντεταγμένες των pixels των στηλών της εικόνας όπου έχουν εντοπιστεί ακμές. Στην μεταβλητή *whiteRow* αποθηκεύονται οι συντεταγμένες των pixels των γραμμών της εικόνας όπου έχουν εντοπιστεί ακμές. Ο τρόπος που οι μεταβλητές αυτές παίρνουν τις τιμές τους εξηγείται στο κεφάλαιο **6.4.2 Επεξήγηση κώδικα περικοπής εικόνας**.

6.3.4 Μεταβλητές δημιουργίας PRM

Για την υλοποίηση του PRM αρχικά δημιουργείται η μεταβλητή *prm* και αρχικοποιείται ως αντικείμενο της κλάσης robotics.PRM(*prm = robotics.PRM).* Στην μεταβλητή *prm.Map* αποθηκεύεται το πλέγμα πληρότητας που θα χρησιμοποιηθεί. Με την μεταβλητή *prm.NumNodes* ορίζεται ο μέγιστος αριθμός των τυχαίων κόμβων που θα δημιουργήσει ο αλγόριθμος. Η μεταβλητή *prm.ConnectionDistance* ορίζει την μέγιστη απόσταση που μπορεί να έχουν δύο κόμβοι ώστε ο αλγόριθμος να τους ενώσει μεταξύ τους. Για την εύρεση διαδρομής ο αλγόριθμος χρειάζεται να γνωρίζει τα σημεία αφετηρίας και τερματισμού. Αυτά ορίζονται με τις μεταβλητές *startLocation* και *endLocation* αντίστοιχα. Τέλος με την συνάρτηση *findpath* γίνεται η εφαρμογή του αλγορίθμου PRM όπου επιστρέφει τις συντεταγμένες των κόμβων από όπου το ρομπότ θα πρέπει να περάσει συμπεριλαμβανομένων και των σημείων αφετηρίας και τερματισμού. Οι πληροφορίες αυτές αποθηκεύονται στην μεταβλητή *path.*

6.4 Περιγραφή κώδικα

Στο κεφάλαιο αυτό γίνεται η περιγραφή του κύριου μέρους του κώδικα που αναπτύχθηκε αλλά και ανάλυση ορισμένων δυσνόητων σημείων.

6.4.1 Περιγραφή κύριου μέρους κώδικα

Στο κεφάλαιο αυτό γίνεται η περιγραφή του κυρίως μέρους του κώδικα τμηματικά. Με τις παρακάτω εντολές:

distance = calcDistance(path)

degrees = calcDegrees(path)
γίνεται ο υπολογισμός της απόστασης δύο διαδοχικών κόμβων, με την βοήθεια της ρουτίνας *calcDistance,* καθώς και η κλίση των ευθειών που ενώνουν δύο κόμβους, με την βοήθεια της ρουτίνας *calcDegrees*. Στη μεταβλητή *distance* αποθηκεύονται οι πληροφορίες σχετικά με την απόσταση και στην μεταβλητή *degrees* αποθηκεύονται οι πληροφορίες σχετικά με την κλίση των ευθειών. Στη συνέχεια χρησιμοποιείται η εντολή *input*('press Enter to continue').

Σε αυτό το σημείο ο κώδικας περιμένει από τον χρήστη να πιέσει το πλήκτρο enter για να συνεχιστεί η εκτέλεσή του. Η εντολή αυτή χρησιμοποιήθηκε για να ορίζεται από τον χρήστη η έναρξη κίνησης του ρομπότ. Η επόμενη εντολή είναι η αρχικοποίηση της μεταβλητής *count* με την τιμή μηδέν (*count = 0*). Η μεταβλητή *count* χρησιμοποιείται ως παράμετρος εξόδου από τον βρόχο επανάληψης που ευθύνεται για τον εντοπισμό του ρομπότ στον χώρο κίνησης. Στο επόμενο βήμα ξεκινάει ο βρόχος επανάληψης όπου μέσα σε αυτόν γίνονται ο εντοπισμός του ρομπότ στον χώρο κίνησης, ο υπολογισμός των παλμών που πρέπει να εκτελέσει το ρομπότ και η ανταλλαγή πληροφοριών μέσω της διεπαφής bluetooth. Ο βρόχος ξεκινάει με την εντολή: *for i=1 : length(distance)*. Με την εντολή αυτή ο αριθμός των επαναλήψεων που θα εκτελέσει ο βρόχος θα είναι ίσος με τον αριθμό των στοιχείων που έχει ο πίνακας *distance* όπου είναι ίσο με τον αριθμό των κόμβων που χρειάζεται να περάσει το ρομπότ. Εντός του βρόχου επανάληψης υπάρχει η λογική της καθοδήγησης του ρομπότ. Περιληπτικά αυτή μπορεί να περιγραφεί ως:

1. Εντοπισμός του ρομπότ στον χώρο κίνησης και υπολογισμός της κλίσης του,
2. Υπολογισμός της στροφής και της κατεύθυνσης που πρέπει να κάνει το ρομπότ ώστε η κλίση του να είναι ίδια με την κλίση της ευθείας που ενώνει τον επόμενο κόμβο,
3. Μετατροπή των δεδομένων στροφής και απόστασης σε παλμούς,
4. Αποστολή πληροφοριών στροφής μέσω της διεπαφής bluetooth,
5. Έλεγχος αν η κλίση του ρομπότ είναι σωστή και διόρθωση αυτής σε περίπτωση που υπάρχει απόκλιση μεγαλύτερη του επιθυμητού,
6. Αποστολή δεδομένων για ευθεία κίνηση του ρομπότ με σκοπό να φτάσει τον επόμενο κόμβο.

Τα έξι αυτά βήματα επαναλαμβάνονται μέχρι το ρομπότ να φτάσει στο σημείο του τερματισμού. Αναλύοντας παραπάνω, για τον εντοπισμό της θέσης του ρομπότ στον χώρο κίνησης χρησιμοποιήθηκε βρόχος επανάληψης με χρήση της εντολής *while count < 10*. Η μεταβλητή *count* είναι η μεταβλητή που αναφέρεται παραπάνω και λειτουργεί ως συνθήκη εξόδου από τον βρόχο επανάληψης σε περίπτωση που δεν είναι δυνατός ο εντοπισμός του ρομπότ. Στην συνέχεια ο κώδικας τραβάει μια καινούργια φωτογραφία με την τρέχουσα θέση του ρομπότ στον χώρο κίνησης, με την εντολή *estimate = snapshot(cam)*. Στην μεταβλητή *estimate* αποθηκεύεται η καινούργια εικόνα. Στην συνέχεια καλείται η ρουτίνα που αναπτύχθηκε για τον εντοπισμό του ρομπότ με την μορφή: *[estimateCenter, estimateDeg] = EstimateRobotPosition(estimate, cameraParams, whiteRow, whiteCol)*. Εάν η ρουτίνα μπόρεσε επιτυχώς να εντοπίσει το ρομπότ τότε οι μεταβλητές *estimateCenter,* θέση του ρομπότ, και *estimateDeg*, κλίση του ρομπότ, έχουν τις ανανεωμένες τιμές της τρέχουσας θέσης του ρομπότ. Σε περίπτωση που δεν ήταν εφικτός ο εντοπισμός του ρομπότ η μεταβλητή *estimateCenter* έχει την τιμή -1 και η μεταβλητή *estimateDeg* έχει την τιμή 'e'. Ο λόγος που επιλέχθηκε ο χαρακτήρας 'e' ως επιστρεφόμενη τιμή λάθους είναι διότι αρνητικές τιμές κλίσης είναι αποδεκτές τιμές. Στην συνέχεια αυξάνεται η μεταβλητή *count* κατά ένα(*count = count + 1)* και γίνεται έλεγχος για το αν εντοπίστηκε το ρομπότ χρησιμοποιώντας την εντολή *if estimateCenter > 0*. Στην περίπτωση που ο εντοπισμός ήταν επιτυχής τότε η μεταβλητή *robotPose* ανανεώνεται με την τρέχουσα κλίση του ρομπότ (*robotPose = estimateDeg)* και η μεταβλητή *robotCenter* με την τρέχουσα θέση του ρομπότ(*robotCenter = estimateCenter)*. Να σημειωθεί πως η μεταβλητή *robotCenter* δημιουργήθηκε προσπαθώντας

να πραγματοποιηθεί έλεγχος και διόρθωση θέσης του ρομπότ, καθώς όμως αυτό δεν ήταν εφικτό, η μεταβλητή αυτή δεν χρησιμοποιείται. Στην συνέχεια δίνεται στην μεταβλητή *count* η τιμή μηδέν (*count = 0*), για την επόμενη εκτέλεση του κυρίως βρόχου και ο βρόχος εντοπισμού του ρομπότ διακόπτεται με την εντολή *break.* Στην περίπτωση που δεν ήταν δυνατόν να εντοπιστεί το ρομπότ τότε ελέγχεται το πόσες προσπάθειες έγιναν με την εντολή *if(count >= 9)* και εάν ο αριθμός είναι μεγαλύτερος ή ίσος του εννιά τότε η μεταβλητή *robotPose* ανανεώνεται με την τιμή της κλίσης που έχει η τρέχουσα ευθεία που ενώνει τον τρέχοντα κόμβο με τον επόμενο (*robotPose = degrees(i)*), δίνεται η τιμή μηδέν στην μεταβλητή *count* και ο βρόχος επανάληψης εντοπισμού του ρομπότ σταματάει. Στην περίπτωση που οι αποτυχημένες προσπάθειες είναι λιγότερες από εννιά τότε επαναλαμβάνεται η παραπάνω διαδικασία.

Αφού έχει ολοκληρωθεί η διαδικασία του εντοπισμού της τρέχουσας θέσης και κλίσης του ρομπότ, σειρά έχει ο υπολογισμός της στροφής και της κατεύθυνσης που πρέπει να εκτελέσει το ρομπότ. Αυτό γίνεται με την εντολή: *[direction, turn] = calcTurn(degrees(i), robotPose).* Έπειτα η μεταβλητή *direction* έχει την κατεύθυνση που πρέπει να στρίψει το ρομπότ, δεξιά ή αριστερά, και η μεταβλητή *turn* περιέχει τις μοίρες της στροφής. Έπειτα γίνεται έλεγχος και οι τιμές αναπροσαρμόζονται ώστε το ρομπότ να διανύσει την μικρότερη γωνία όπως φαίνεται στο παρακάτω κομμάτι κώδικα:

```
if abs(turn) > 180
   turn = abs(360 - abs(turn));
   if direction == 'l'
      direction = 'r';
   elseif direction == 'r'
      direction = 'l';
   end
end
```

Στο επόμενο βήμα γίνεται η μετατροπή των δεδομένων στροφής και απόστασης σε παλμούς. Για να γίνει η μετατροπή της γωνίας που πρέπει να στρίψει το ρομπότ σε παλμούς αρχικά πρέπει να υπολογιστεί το τόξο που αντιστοιχεί σε αυτή τη γωνία. Αυτό επιτυγχάνεται με την χρήση της εντολής *arcFromDegrees = degreesToArc(turn, robotWidth)* όπου στην μεταβλητή *arcFromDegrees* αποθηκεύεται το μήκος τόξου της γωνίας. Στην συνέχεια το μήκος τόξου μετατρέπεται σε παλμούς με την εντολή *arcFromDegrees = distanceToPulses(arcFromDegrees, pulsesPerRotation, wheelCircumference)* όπου το αποτέλεσμα αποθηκεύεται πίσω στην μεταβλητή *arcFromDegrees*. Επειδή είναι πιθανό το αποτέλεσμα της μετατροπής να είναι δεκαδικός αριθμός γίνεται στρογγυλοποίηση της τιμής στον πλησιέστερο ακέραιο με την εντολή *arcFromDegrees = round(arcFromDegrees).* Η στρογγυλοποίηση γίνεται διότι δεν είναι δυνατό να μετρηθούν υποδιαιρέσεις περιστροφής του τροχού από τον Arduino με ικανοποιητική ακρίβεια. Στην συνέχεια ακολουθείται παρόμοια διαδικασία για την μετατροπή της απόστασης σε παλμούς όπως φαίνεται στις παρακάτω εντολές:

```
dist = distanceToPulses(distance(i), pulsesPerRotation, wheelCircumference)
dist = round(dist)
```

Έτσι στο τέλος της διαδικασίας αυτής στη μεταβλητή *arcFromDegrees* έχει αποθηκευτεί ο αριθμός των παλμών που σχετίζονται με την στροφή και στην μεταβλητή *dist* ο αριθμός των παλμών που σχετίζονται με την απόσταση.

Αφού έχει γίνει ο υπολογισμός των παλμών για την κάθε κίνηση που πρέπει να εκτελέσει το ρομπότ με σκοπό να φτάσει τον επόμενο κόμβο οι πληροφορίες αυτές πρέπει να αποσταλούν στο ρομπότ μέσω της διεπαφής bluetooth. Η πρώτη κίνηση που θα εκτελέσει το ρομπότ είναι η στροφή ώστε να ευθυγραμμιστεί με τον επόμενο κόμβο. Αρχικά στέλνεται η κατεύθυνση που πρέπει να στρίψει το ρομπότ με την εντολή *fprintf(dev, '%c', direction)* και στην συνέχεια ο κώδικας διαβάζει την απάντηση που στέλνει ο Arduino με την εντολή *data = fscanf(dev, '%i')*. Για να συνεχιστεί η εκτέλεση του κώδικά πρέπει η απάντηση που θα στείλει ο Arduino, όπου αποθηκεύεται στην μεταβλητή *data*, να έχει την τιμή 2, έτσι μέχρι να ληφθεί η τιμή αυτή δεν αποστέλνονται καινούργια δεδομένα όπως φαίνεται στις παρακάτω εντολές:

```
while(data~=2)
  data = fscanf(dev, '%i');
end
```

Εφόσον ληφθεί η τιμή 2 στην συνέχεια στέλνεται η πληροφορία των παλμών της στροφής, ακολουθώντας την ίδια λογική με την μόνη διαφορά ότι σαν απάντηση αναμένεται η τιμή 1 όπως φαίνεται στις παρακάτω εντολές:

```
fprintf(dev, '%i', abs(arcFromDegrees));
data = fscanf(dev, '%i');
while(data~=1)
   data = fscanf(dev, '%i');
end
```

Η λήψη της τιμής 1 σηματοδοτεί ότι το ρομπότ εκτέλεσε επιτυχώς μία κίνηση και στην συγκεκριμένη περίπτωση η κίνηση αυτή ήταν η εκτέλεση στροφής. Στο επόμενο βήμα γίνεται ο έλεγχος για το εάν η κλίση του ρομπότ δεν έχει μεγάλη απόκλιση από την επιθυμητή κλίση καθώς η διαδικασία της στρογγυλοποίησης των παλμών που έγινε προηγουμένως σε συνδυασμό με την ορμή που αποκτάει το ρομπότ κατά την κίνησή του δημιουργεί αποκλίσεις από την επιθυμητή τιμή. Αρχικά το πρόγραμμα περιμένει για μισό δευτερόλεπτο με την χρήση της εντολής *pause(0.5).* Η εντολή αυτή χρησιμοποιήθηκε για να βεβαιωθεί πως κατά την λήψη της φωτογραφίας για την εκτίμηση της κλίση του ρομπότ, το ρομπότ έχει ακινητοποιηθεί πλήρως και δεν εκτελεί ακόμα κάποια κίνηση. Για τον έλεγχο της κλίσης του ρομπότ εφαρμόστηκε παρόμοια λογική με αυτήν που αναλύεται παραπάνω σχετικά με τον εντοπισμό της θέσης του ρομπότ στον χώρο κίνησης. Η διαφορά είναι πως αφού εντοπιστεί η θέση του ρομπότ στον χώρο κίνησης, ελέγχεται εάν η απόκλιση της κλίσης του ρομπότ από την κλίση της ευθείας που ενώνει τον τρέχοντα βρόχο με τον επόμενο είναι μεγαλύτερη ή ίση από 3.5 μοίρες, χρησιμοποιώντας την εντολή *while (abs(robotPose - degrees(i)) >= 3.5).* Στην συνέχεια εκτελείται η εντολή *correctPose(robotPose, degrees(i), robotWidth, pulsesPerRotation, wheelCircumference, dev)* όπου αποστέλνονται στο ρομπότ οι διορθωτικές κινήσεις που πρέπει να κάνει ώστε να μειωθεί το σφάλμα. Έπειτα γίνεται λήψη νέας φωτογραφίας (*estimate = snapshot(cam)*) καλείται ξανά η ρουτίνα για τον εντοπισμό του ρομπότ στο χώρο κίνησης (*[estimateCenter, estimateDeg] = EstimateRobotPosition(estimate, cameraParams, whiteRow, whiteCol)*) και γίνεται ανανέωση της τιμής της κλίσης του ρομπότ με την νέα τιμή (*robotPose = estimateDeg*). Εάν το σφάλμα είναι μικρότερο από 3.5 μοίρες τότε το πρόγραμμα συνεχίζει την εκτέλεσή τού, διαφορετικά επαναλαμβάνεται η παραπάνω διαδικασία μέχρι το σφάλμα να είναι εντός αποδεκτών ορίων.

Με την ολοκλήρωση του ελέγχου της κλίσης του ρομπότ, ότι βρίσκεται εντός επιθυμητών ορίων, στέλνονται μέσω της διεπαφής bluetooth οι πληροφορίες για την ευθύγραμμη κίνηση του, χρησιμοποιώντας τις ίδιες εντολές με αυτές για την αποστολή των

πληροφοριών της στροφής, όπως φαίνεται παρακάτω:

```
fprintf(dev, '%c', 'f');
data = fscanf(dev, '%i');
while(data~=2)
   data = fscanf(dev, '%i');
end
fprintf(dev, '%i', dist);
data = fscanf(dev, '%i');
while(data~=1)
   data = fscanf(dev, '%i');
end
```

Η λήψη της τιμής 1 αυτή την φορά σηματοδοτεί πως το ρομπότ εκτέλεσε επιτυχώς την ευθύγραμμη κίνηση και πως βρίσκεται στον επόμενο κόμβο. Τέλος χρησιμοποιείται ξανά η εντολή *pause(0.5)* και η διαδικασία επαναλαμβάνεται ξανά από την αρχή μέχρι το ρομπότ να φτάσει στο σημείο του τερματισμού.

6.4.2 Επεξήγηση κώδικα περικοπής εικόνας

Για να γίνει η περικοπή της εικόνας χρησιμοποιήθηκε η ρουτίνα που περιγράφεται στο κεφάλαιο **6.1.6**. Για να μπορέσει η ρουτίνα να κάνει την περικοπή χρειάζεται να δοθεί το τετράγωνο όπου χαρακτηρίζει το επιθυμητό μέρος της εικόνας. Παρακάτω φαίνεται το κομμάτι του κώδικα που αναπτύχθηκε με σκοπό να βρεθεί το επιθυμητό τετράγωνο της εικόνας δηλαδή, ο χώρος κίνησης του ρομπότ:

```
whiteCol ={};
whiteRow={};
colPrev = -1;
rowPrev = -1;
for col = 1:600
   for row = 250:1400
      if(edgemap(col,row) > 0)
         if (col~=colPrev)
            whiteCol=[whiteCol, col];
            colPrev = col;
         end
         if (row~=rowPrev)
            whiteRow=[whiteRow, row];
            rowPrev = row;
         end
      end
   end
end
```

Αρχικά αρχικοποιούνται οι μεταβλητές *whiteCol* και *whiteRow* ως κενοί πίνακες. Στις μεταβλητές αυτές θα αποθηκευτούν οι συντεταγμένες των σημείων της εικόνας όπου υπάρχουν ακμές. Στην μεταβλητή *whiteCol* αποθηκεύεται η συντεταγμένη X και στην μεταβλητή *whiteRow* αποθηκεύεται η συντεταγμένη Y. Οι δύο επόμενες μεταβλητές, *colPrev* και *rowPrev* είναι βοηθητικές μεταβλητές που βοηθούν στην ανάλυση της εικόνας

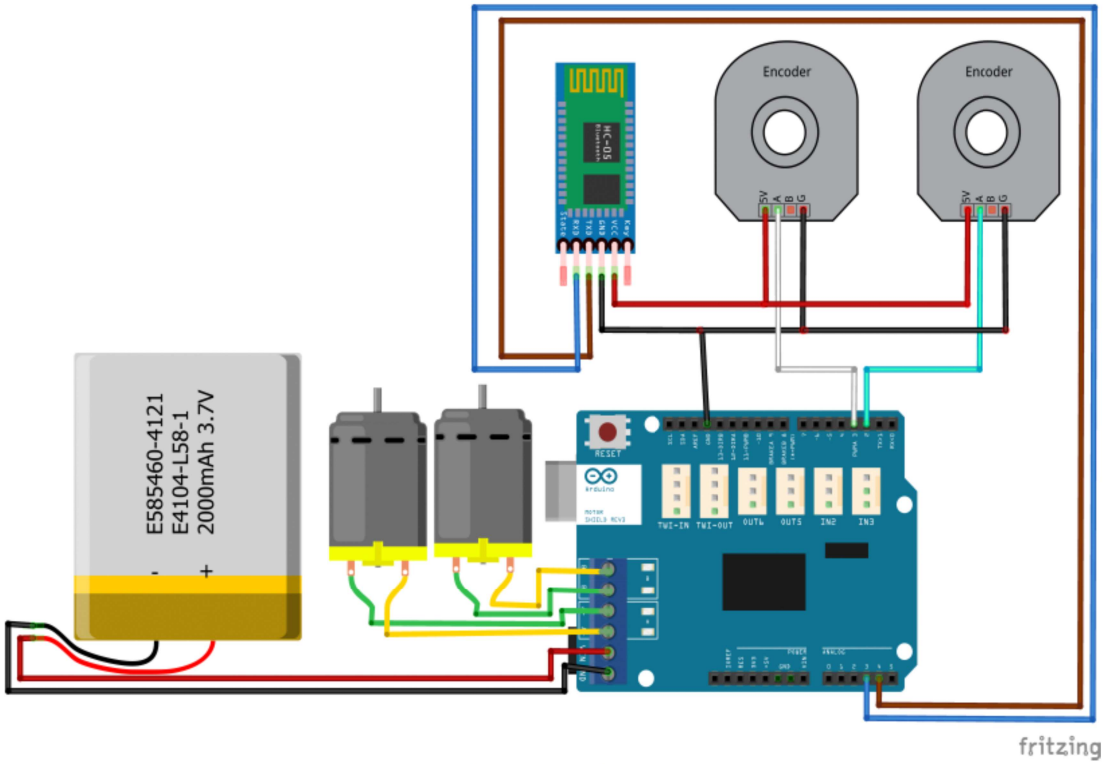
και αρχικοποιούνται με την τιμή -1. Για να γίνει η ανάλυση της εικόνας χρησιμοποιήθηκαν δύο βρόχοι επανάληψης, ο πρώτος *for col = 1:600,* διαβάζει τις στήλες της εικόνας και ο δεύτερος, *for row = 250:1400*, διαβάζει τις γραμμές της εικόνας. Όπως φαίνεται και στη εικόνα 5 εντοπίζονται κι άλλες ακμές πέραν των ακμών που χαρακτηρίζουν το περίγραμμα του χώρου κίνησης. Για να γίνει πιο εύκολη η επεξεργασία των δεδομένων δεν γίνεται ανάλυση ολόκληρης της εικόνας αλλά επιλέγεται ένα εύρος όπου υπάρχει ο χώρος κίνησης και οι υπόλοιπες ακμές μένουν εκτός της ανάλυσης. Το εύρος καθορίζεται από τις τιμές 1:600 που ορίζουν το ύψος και 250:1400 που ορίζουν το πλάτος όπου θα γίνει η ανάλυση. Οι τιμές αυτές βρέθηκαν πειραματικά. Η διαδικασία αυτή χρησιμοποιήθηκε διότι το περιβάλλον διεξαγωγής των πειραμάτων ήταν σταθερό, η σχετική θέση κάμερας - χώρου κίνησης δεν άλλαζε, κι έτσι ήταν γνωστό πως ο χώρος κίνησης θα υπήρχε πάντα μέσα σε αυτό το εύρος. Στην συνέχεια με την εντολή *if(edgemap(col,row) > 0)* ελέγχεται αν στο συγκεκριμένο σημείο της εικόνας το χρώμα είναι μαύρο (0) ή όχι (>0). Αν το σημείο είναι μαύρο αυτό σημαίνει πως δεν υπάρχει ακμή και ο κώδικας συνεχίζει στο επόμενο σημείο. Αν δεν είναι μαύρο τότε αποθηκεύονται οι συντεταγμένες στους αντίστοιχους πίνακες. Με τις εντολές *if (col~=colPrev), if (row~=rowPrev)* ελέγχεται ότι η συντεταγμένη αυτή δεν είναι η ίδια με την τελευταία που έχει προστεθεί στο πίνακα. Ο έλεγχος αυτός γίνεται λόγω του διπλού βρόχου επανάληψης όπου εξαιτίας του υπάρχει περίπτωση το ίδιο σημείο να αποθηκευτεί παραπάνω από μία φορά. Στην συνέχεια στο τέλος του κάθε πίνακα προσθέτονται τα καινούργια δεδομένα με τις εντολές: *whiteCol=[whiteCol, col], whiteRow=[whiteRow, row]* και οι εντολές *colPrev = col, rowPrev = row* ανανεώνουν τις πληροφορίες σχετικά με το τελευταίο σημείο που προστέθηκε σε κάθε πίνακα. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να γίνει η ανάλυση όλου του εύρους.

Όταν τελειώσει η ανάλυση η μεταβλητή *whiteCol* περιέχει τη συντεταγμένη Χ των σημείων της εικόνας όπου εντοπίστηκαν οι ακμές του χώρου κίνησης και η μεταβλητή *whiteRow* τη συντεταγμένη Υ των σημείων. Στην συνέχεια η μεταβλητές αυτές χρησιμοποιούνται για να γίνει η περικοπή της εικόνας όπως φαίνεται παρακάτω:

cropedMap = imcrop(map, [min(cell2mat(whiteRow)) min(cell2mat(whiteCol)) (max(cell2mat(whiteRow)) - min(cell2mat(whiteRow))) (max(cell2mat(whiteCol)) - min(cell2mat(whiteCol)))])

Η εντολή *min(cell2mat(whiteRow))* επιστρέφει το αρχικό σημείο Χ του τετραγώνου περικοπής, η εντολή *min(cell2mat(whiteCol))* επιστρέφει το αρχικό σημείο Υ, το πλάτος του τετραγώνου δίνεται από την εντολή *(max(cell2mat(whiteRow)) - min(cell2mat(whiteRow)))* και το ύψος του δίνεται από την εντολή *(max(cell2mat(whiteCol)) - min(cell2mat(whiteCol))).*

Κεφάλαιο 7: Συνδεσμολογία Hardware



Εικόνα 13: Σχεδιάγραμμα συνδεσμολογίας Arduino

Η Εικόνα 13 δείχνει την συνδεσμολογία που πραγματοποιήθηκε για την κατασκευή του ρομπότ. Να σημειωθεί πως η εικόνα δεν απεικονίζει τα πραγματικά υλικά που χρησιμοποιήθηκαν και τα χαρακτηριστικά που αναγράφονται πάνω σε αυτά(πχ. μπαταρία 3.7V) δεν αντιστοιχούν στα πραγματικά χαρακτηριστικά. Τα υλικά της εικόνας έχουν αντιστοιχία με τα πραγματικά μόνο ως προς την ονομασία των pins και πρέπει να γίνεται αναφορά σε αυτήν μόνο για την συνδεσμολογία.

Τα υλικά που χρησιμοποιήθηκαν για την δημιουργία του ρομπότ είναι:

- Μικροεπεξεργαστής: Arduino Uno.
- Οδηγός μοτέρ: Πλακέτα οδήγησης μοτέρ L298P.
- Πλακέτα Bluetooth HC-06.
- Μπαταρία LiPo 12V 2200mAh.
- 2x 12V DC μοτέρ με encoder.

Ο οδηγός μοτέρ L298P έχει την μορφή shield, δηλαδή είναι μία πλακέτα που έχει τα ίδια pins με τον Uno και μπαίνει πάνω από αυτόν, δίνοντας πρόσβαση σε όλα τα pins του Uno. Το L298P τροφοδοτείται από την μπαταρία και έχει την δυνατότητα να τροφοδοτήσει και τον

Uno. Παρέχει δύο εξόδους για μοτέρ τις MotorA και MotorB. Τα μοτέρ τροφοδοτούνται από τις εξόδους αυτές. Στον πίνακα 2 αναφέρονται τα pins του Uno που ελέγχουν την λειτουργία του L298P. Η ταχύτητα μπορεί να ελεγχθεί από το ψηφιακό pin 10 του μικροεπεξεργαστή για την έξοδο MotorA και το pin 11 για την έξοδο MotorB. Τα pins αυτά έχουν την δυνατότητα παραγωγής παλμού PWM με τον οποίο ρυθμίζεται η τάση τροφοδοσίας των μοτέρ. Η φορά περιστροφής των μοτέρ ρυθμίζεται από το pin 12 για την έξοδο MotorA και το pin 13 για την έξοδο MotorB.

Πίνακας 2: Pins ελέγχου L298P

	Pin ταχύτητας	Pin κατεύθυνσης	Κατεύθυνση
Motor A	D10	D12	1 = Δεξιόστροφα, 0 = Αριστερόστροφα
Motor B	D11	D13	1 = Δεξιόστροφα, 0 = Αριστερόστροφα

Το κάθε μοτέρ έχει από έναν encoder, encoderA για το MotorA και encoderB για το MotorB. Ο κάθε encoder έχει δύο pins εισόδου και δύο pins εξόδου. Είσοδος είναι η τροφοδοσία του encoder και τα pins συνδέονται με τα pins 5V και GND του Uno. Έξοδος του encoder είναι το σήμα των παλμών που αντιστοιχούν στην περιστροφή του μοτέρ. Ο κάθε encoder έχει δύο εξόδους όπου οι παλμοί μεταξύ των δύο έχουν διαφορά φάσης 90 μοίρες. Η διαφορά φάσης στις εξόδους του encoder βοηθάει στην ανίχνευση φοράς περιστροφής του μοτέρ. Στην κατασκευή του ρομπότ χρησιμοποιήθηκε μόνο η μία έξοδος του κάθε encoder καθώς η φορά περιστροφής είναι γνωστή από τα pins κατεύθυνσης του L298P. Το σήμα εξόδου του encoderA συνδέθηκε στο ψηφιακό pin 2 του Uno και του encoderB στο ψηφιακό pin 3. Τα δύο αυτά pins έχουν την δυνατότητα external interupt. Με την χρήση των interupts δεν χρειάζεται να γίνεται έλεγχος μέσα στον κώδικα για το αν έχει αλλάξει κατάσταση η έξοδος του encoder. Το hardware μπορεί να εντοπίσει την αλλαγή στην έξοδο του encoder και μόλις αυτή γίνει διακόπτεται η κανονική ροή του προγράμματος και εκτελείται συγκεκριμένη ρουτίνα για κάθε ένα από τα pins του Uno. Στην συνέχεια το πρόγραμμα επιστρέφει στο σημείο που διακόπηκε και συνεχίζεται η κανονική του ροή.

Τέλος το HC-06 τροφοδοτείται από τα pins 5V και GND του Uno. Επικοινωνεί με τον Uno με σειριακό πρωτόκολλο. Για την σύνδεση χρησιμοποιήθηκε η βιβλιοθήκη *SoftwareSerial* που μπορεί να προσομοιώσει pins σειριακής επικοινωνίας σε οποιαδήποτε pins του Uno. Το pin A4 του Uno προσομοιώνει το σήμα Rx και συνδέθηκε με το pin Tx του HC-06 και το pin A5 του Uno προσομοιώνει το σήμα Tx και συνδέθηκε με το pin Rx του HC-06.

Κεφάλαιο 8: Μεθοδολογία Κώδικα Arduino

Για την υλοποίηση του ρομπότ αρχικά αναπτύχθηκε η λογική της επικοινωνίας Bluetooth. Για να έχει την δυνατότητα ο Uno να επικοινωνεί με Bluetooth χρησιμοποιήθηκε το module HC-06. Τα προβλήματα που έπρεπε να λυθούν ήταν:

1. Διαχωρισμός εντολών κατεύθυνσης
2. Διαχωρισμός εντολών κατεύθυνσης και αριθμού παλμών
3. Συγχρονισμός κώδικα Matlab και κώδικα μικροεπεξεργαστή.

Το πρόβλημα του συγχρονισμού λύθηκε χρησιμοποιώντας συγκεκριμένες τιμές που σηματοδοτούν την κατάσταση του ρομπότ. Μόλις το ρομπότ είναι έτοιμο να πραγματοποιήσει μία κίνηση στέλνει την τιμή *'1'* και ως απάντηση περιμένει την κατεύθυνση που πρέπει να ακολουθήσει. Το Matlab περιμένει να διαβάσει την τιμή *'1'* και στέλνει την εντολή σχετικά με την κατεύθυνση και περιμένει επιβεβαίωση από το ρομπότ για να στείλει τον αριθμό των παλμών. Μόλις το ρομπότ λάβει την πληροφορία σχετικά με την κατεύθυνση στέλνει την τιμή *'2'* που σηματοδοτεί ότι το ρομπότ περιμένει των αριθμό των παλμών. Το Matlab διαβάζει την τιμή *'2'* και στέλνει των αριθμό των παλμών. Στην συνέχεια το ρομπότ εκτελεί την κίνηση. Μόλις αυτή ολοκληρωθεί στέλνει ξανά την τιμή *'1'* για να ενημερώσει ότι είναι έτοιμο να εκτελέσει καινούργια κίνηση και ο κύκλος επαναλαμβάνεται μέχρι το ρομπότ, να φτάσει στον τερματισμό. Οι κινήσεις εκτελούνται πάντα με την ίδια σειρά. Το ρομπότ πρώτα θα στρίψει για να διορθώσει την γωνία του και έπειτα θα κινηθεί ευθεία για να φτάσει στον επόμενο κόμβο.

Η διαδικασία που αναλύθηκε παραπάνω λύνει τα προβλήματα του συγχρονισμού και της διαφοροποίησης εντολών κατεύθυνσης και αριθμού παλμών. Το επόμενο βήμα είναι να γίνει ο διαχωρισμός των εντολών κατεύθυνσης, δηλαδή να "γνωρίζει" το ρομπότ αν πρέπει να κινηθεί ευθεία, να στρίψει αριστερά ή να στρίψει δεξιά. Το πρόβλημα αυτό λύθηκε εφαρμόζοντας την ίδια προσέγγιση με το πρόβλημα του συγχρονισμού. Χρησιμοποιήθηκαν συγκεκριμένες τιμές για τον ορισμό της κάθε κίνησης που μπορεί να κάνει το ρομπότ. Η τιμή *'f'* ορίζει ευθεία κίνηση μπροστά, η τιμή *'r'* ορίζει στροφή δεξιά, η τιμή *'l'* ορίζει στροφή αριστερά και η τιμή *'b'* ορίζει ευθεία κίνηση προς τα πίσω. Όταν το ρομπότ στείλει την τιμή *'1'* περιμένει ως απάντηση από το Matlab μία από τις τιμές *'f', 'r'* ή, *'l'*. Η τιμή *'b'* δεν χρησιμοποιείται από τον αλγόριθμο του Matlab. Η ευθεία κίνηση προς τα πίσω εκτελείται ως στροφή 180 μοιρών και κίνηση ευθεία μπροστά.

Το επόμενο βήμα ήταν να καθοριστεί ο αριθμός μιας πλήρους περιστροφής του μοτέρ. Τα τεχνικά χαρακτηριστικά αναφέρουν πως μια πλήρης περιστροφή αντιστοιχεί σε 550 παλμούς. Το νούμερο αυτό, όπως αναφέρεται και στο κεφάλαιο 5, αποδείχθηκε πως δεν είναι σωστό.

Εικόνα 14: Σύνδεση DC motor με Stepper motor

Η εύρεση του ακριβή αριθμού παλμών μιας πλήρους περιστροφής έγινε με την βοήθεια ενός μοτέρ τύπου stepper. Επιλέχθηκε μοτέρ τύπου stepper διότι χρησιμοποιούνται για εφαρμογές που χρειάζονται μεγάλη ακρίβεια και είναι εύκολος ο υπολογισμός των παλμών που χρειάζονται για να κάνουν μία περιστροφή. Το stepper που χρησιμοποιήθηκε έχει βήμα 1,8 μοίρες ανά παλμό άρα, χρειάζονται 200 παλμοί για μία πλήρη περιστροφή. Τα δύο μοτέρ συνδέθηκαν όπως φαίνεται στην Εικόνα 14. Για την οδήγηση του stepper χρησιμοποιήθηκε το L298P. Στην συνέχεια συνδέθηκε ο encoder του μοτέρ του ρομπότ στον Uno για να γίνει η μέτρηση των παλμών. Για την μέτρηση των παλμών το stepper κάνει μια περιστροφή δεξιόστροφα και έπειτα μια περιστροφή αριστερόστροφα. Η διαδικασία αυτή επαναλαμβάνεται δέκα φορές για κάθε μοτέρ.

Πίνακας 3: Αποτελέσματα δεξιόστροφης περιστροφής

MotorA Δεξιόστροφα		MotorB Δεξιόστροφα	
Pulses(Stepper)	Steps(Encoder)	Pulses(Stepper)	Steps(Encoder)
199	600	205	602
196	600	196	600
196	600	196	601
197	602	196	601
196	601	200	600
196	601	200	600
196	601	200	600
196	601	200	601
196	602	200	601
169	601	200	600
M.O.		M.O	
196.4	600,6	199.3	600.6

Πίνακας 4: Αποτελέσματα αριστερόστροφης περιστροφής

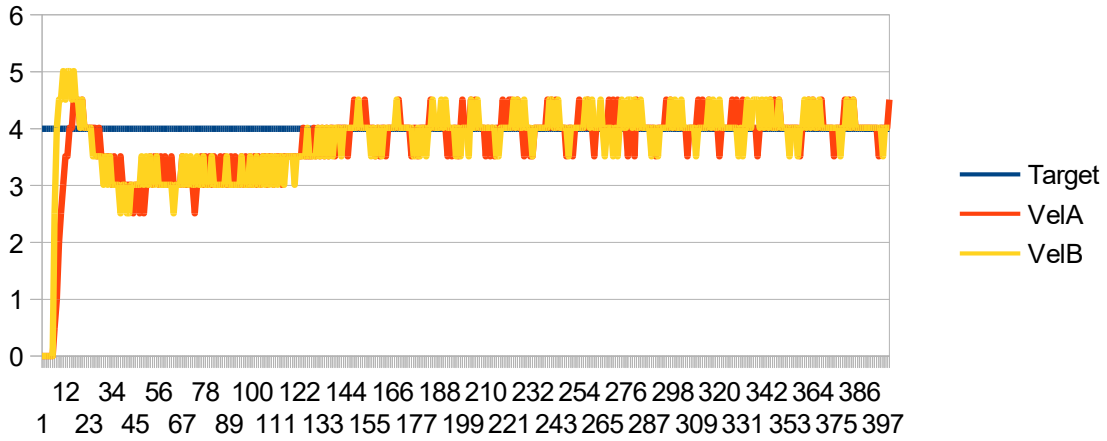
MotorA Αριστερόστροφα		MotorB Αριστερόστροφα	
Pulses(Stepper)	Steps(Encoder)	Pulses(Stepper)	Steps(Encoder)
197	603	199	600
196	600	196	600
197	603	196	600

196	600	197	602
197	603	196	601
196	600	196	601
196	600	196	601
196	600	196	601
196	601	196	602
200	600	196	601
M.O.		M.O	
196,7	601	196,4	600,9

Ο πίνακας 3 δείχνει τα αποτελέσματα της δεξιόστροφης περιστροφής και ο πίνακας 4 τα αποτελέσματα της αριστερόστροφης περιστροφής. Όπως φαίνεται από τους δύο αυτούς πίνακες ο αριθμός παλμών μιας πλήρους περιστροφής δεν είναι 550 όπως αναφέρουν τα τεχνικά χαρακτηριστικά αλλά, πλησιάζουν την τιμή 600. Να σημειωθεί πως δεν ήταν δυνατή η μέτρηση των παλμών με 100% ακρίβεια λόγο περιορισμών του hardware.

Το επόμενο βήμα ήταν η μέτρηση της ταχύτητας των μοτέρ και ο έλεγχος αυτής. Όπως αναφέρεται στο κεφάλαιο 6 οι έξοδοι των encoders συνδέονται σε pins του Uno που έχουν την δυνατότητα external interupts. Μόλις η έξοδος κάποιου encoder μεταβεί από κατάσταση λογικού μηδέν σε λογικό ένα εκτελείται η αντίστοιχη ρουτίνα. Μέσα στην ρουτίνα υπάρχει μία μεταβλητή η οποία αυξάνεται κατά ένα σε κάθε εκτέλεση. Στην συνέχεια, για να γίνει ο υπολογισμός της ταχύτητας, η κύρια ρουτίνα επανάληψης του προγράμματος υπολογίζει την διαφορά των τιμών μεταξύ δύο διαδοχικών επαναλήψεων και το αποτέλεσμα της διαφοράς διαιρείται με τον χρόνο που χρειάστηκε για να εκτελεστούν οι δύο επαναλήψεις. Παρακάτω δίνεται ο τύπος υπολογισμού:

$$u = (pulses - pulsesPrev) / dt \qquad (6)$$

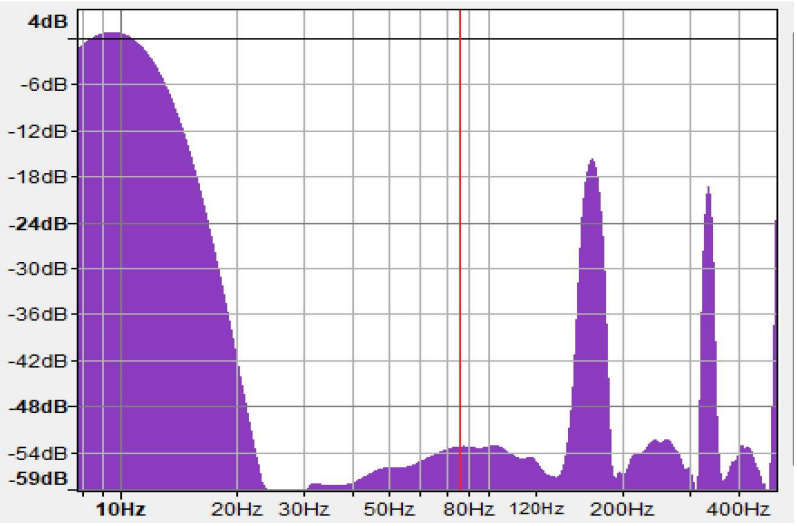


Εικόνα 15: Διάγραμμα ταχύτητας με θόρυβο

Οι μονάδες του αποτελέσματος του τύπου (6) είναι σε pulses/s. Για να γίνει έλεγχος της ταχύτητας πρέπει να γίνει μετατροπή σε RPM. Η μετατροπή γίνεται με τον τύπο:

$$RPM = pulsesOneRevolution / (u * dt) \qquad (7)$$

όπου pulsesOneRevolution είναι ο αριθμός των παλμών μιας πλήρους περιστροφής. Για τον έλεγχο της ταχύτητας χρησιμοποιήθηκες ο αλγόριθμος PID. Όπως φαίνεται και στην Εικόνα 15 ο αλγόριθμος δυσκολεύεται να σταθεροποιηθεί στον στόχο των 4RPM διότι η μέτρηση της ταχύτητας περιέχει θόρυβο που "μπερδεύει" τον αλγόριθμο ελέγχου. Για την βελτίωση του ελέγχου πρέπει να εφαρμοστεί φίλτρο το οποίο θα μειώνει την επίδραση του θορύβου κατά τους υπολογισμούς του αλγορίθμου PID. Για να βρεθεί ο τύπος του φίλτρου που θα χρησιμοποιηθεί έγινε ανάλυση του διαγράμματος της ταχύτητας στο πεδίο της συχνότητας.



Εικόνα 16: Ανάλυση ταχύτητας στο πεδίο της συχνότητας

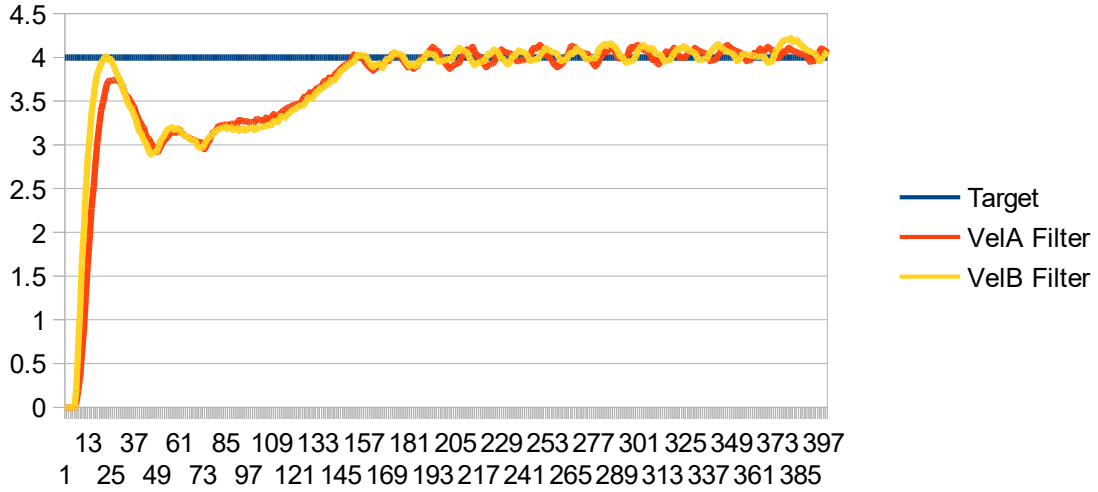
Όπως φαίνεται και από την Εικόνα 16 το σήμα επικεντρώνεται μεταξύ 10Hz και 20Hz και εμφανίζονται αρμονικές κοντά στα 200Hz και 400Hz. Μετά την ανάλυση της ταχύτητας στο πεδίο της συχνότητας εφαρμόστηκε χαμηλοπερατό φίλτρο με συχνότητα αποκοπής τα 25Hz. Στην Εικόνα 17 φαίνεται πως μετά την εφαρμογή του χαμηλοπερατού φίλτρου ο αλγόριθμος PID μπορεί να ελέγξει με μεγαλύτερη ακρίβεια την ταχύτητα. Να σημειωθεί ότι τα διαγράμματα ταχύτητας που απεικονίζονται στις εικόνες 16 και 17 έγιναν με λειτουργία των μοτέρ εν κενώ, δηλαδή το ρομπότ ήταν ανυψωμένο και οι ρόδες του γυρνούσαν δίχως να ακουμπάν στο έδαφος.

Το κάθε μοτέρ ελέγχεται από ξεχωριστή μονάδα ελέγχου PID. Για αυτό τον λόγο ο κάθε PID έπρεπε να ρυθμιστεί ώστε τα μοτέρ να δουλεύουν συγχρονισμένα για την ομαλή πορεία του ρομπότ. Οι στόχοι που τέθηκαν για την ρύθμιση των PID ήταν οι εξής:

- Η επιτάχυνση των μοτέρ να γίνεται ομαλά και με τον ίδιο ρυθμό
- Να μην γίνει υπερπήδηση του στόχου της ταχύτητας
- Η σταθεροποίηση της ταχύτητας να γίνεται την ίδια χρονική στιγμή.

Αν κάποιος από αυτούς τους στόχους δεν ικανοποιείται τότε η συμπεριφορά του ρομπότ είναι απρόβλεπτη. Αν για παράδειγμα η επιτάχυνση των δύο μοτέρ δεν γίνει με τον ίδιο ρυθμό αυτό θα έχει σαν αποτέλεσμα το ρομπότ να στρίβει προς την μεριά της ρόδας με την μικρότερη ταχύτητα και τελικά το ρομπότ να βγει εκτός πορείας. Οι τιμές για την ρύθμιση των PID βρέθηκαν πειραματικά και για το hardware που χρησιμοποιήθηκε είναι οι εξής:

- MotorA: KP = 30, KI = 12, KD = 0
- MotorB: KP = 25, KI = 14, KD = 0,08



Εικόνα 17: Διάγραμμα ταχύτητας με αφαίρεση θορύβου

Κεφάλαιο 9: Τεχνική ανάλυση κώδικα Arduino

Στο κεφάλαιο αυτό γίνεται επεξήγηση του κώδικα που αναπτύχθηκε στο Arduino. Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε είναι η C++ και εφαρμόστηκαν τεχνικές αντικειμενοστραφή προγραμματισμού. Για την ανάπτυξη του κώδικα χρησιμοποιήθηκαν δύο βιβλιοθήκες. Η πρώτη βιβλιοθήκη που χρησιμοποιήθηκε είναι η βιβλιοθήκη *SoftwareSerial.* Η βιβλιοθήκη αυτή δίνει την δυνατότητα της προσομοίωσης σειριακής επικοινωνίας σε δύο οποιαδήποτε ψηφιακά pins του Arduino. Με την βοήθεια της βιβλιοθήκης *SoftwareSerial* πραγματοποιήθηκε η επικοινωνία του Arduino με την πλακέτα διεπαφής Bluetooth. Η κύρια σειριακή επικοινωνία του μικροεπεξεργαστή χρησιμοποιήθηκε για τον προγραμματισμό του καθώς και για την αποσφαλμάτωση του κώδικα.

Η δεύτερη βιβλιοθήκη που χρησιμοποιήθηκε είναι η *atomic*. Με την χρήση της βιβλιοθήκης είναι δυνατό να χρησιμοποιηθεί η εντολή *ATOMIC_BLOCK()* η οποία είναι μία μάκρο-εντολή και προσφέρει την δυνατότητα να χαρακτηριστεί ένα κομμάτι κώδικα να εκτελεστεί ατομικά(δηλαδή χωρίς διακοπές). Η λειτουργία της εντολής είναι παρόμοια με τις συναρτήσεις που προσφέρονται από το αναπτυξιακό του Arduino *interrupts()* και *nointerrupts()* με την διαφορά ότι με την χρήση της παραμέτρου *ATOMIC_RESTORESTATE* όχι μόνο ενεργοποιούνται ξανά οι διακοπές αλλά, επαναφέρονται στην κατάσταση όπου βρίσκονταν προτού εκτελεστεί το κομμάτι του κώδικα που περικλείεται από την εντολή *ATOMIC_BLOCK*. Η βιβλιοθήκη *atomic* χρησιμοποιήθηκε διότι παρέχει έναν ασφαλέστερο και βελτιστοποιημένο τρόπο για την διαχείριση των διακοπών.

Για την υλοποίηση του κώδικα χρησιμοποιήθηκαν ρουτίνες και κλάσεις όπου παρέχονται από το αναπτυξιακό του Arduino καθώς επίσης αναπτύχθηκαν και κάποιες επιπλέον ρουτίνες και κλάσεις όπου εξηγούνται παρακάτω.

9.1 Επεξήγηση παγκόσμιων(global) μεταβλητών

9.1.1 Σταθερές ελέγχου πλακέτας L298P

Για να είναι δυνατός ο έλεγχος της πλακέτας οδήγησης μοτέρ L298P από τον Arduino ήταν απαραίτητο να καθοριστούν τα pins του μικροεπεξεργαστή για την σωστή επεξεργασία των πληροφοριών. Παρακάτω γίνεται η εξήγηση των σταθερών που χρησιμοποιήθηκαν.

Μοτέρ Α:

- #define ENC1_A 2: Έξοδος 1 κωδικοποιητή - ψηφιακό pin Arduino 2.
- #define ENC2_A 9: Έξοδος 2 κωδικοποιητή - ψηφιακό pin Arduino 9.
- #define PWM_A 10: Ρύθμιση ταχύτητας - ψηφιακό pin Arduino 10 με δυνατότητα PWM.
- #define IN_A 12: Ρύθμιση φοράς περιστροφής - ψηφιακό pin Arduino 12.

Μοτέρ Β:

- #define ENC1_B 3: Έξοδος 1 κωδικοποιητή - ψηφιακό pin Arduino 3.

- #define ENC2_B 8: Έξοδος 2 κωδικοποιητή - ψηφιακό pin Arduino 8.
- #define PWM_B 11: Ρύθμιση ταχύτητας - ψηφιακό pin Arduino 11 με δυνατότητα PWM.
- #define IN_B 13: Ρύθμιση φοράς περιστροφής - ψηφιακό pin Arduino 13.

9.1.2 Γενικές παγκόσμιες μεταβλητές

- #define PULSES_ONE_REVOLUTION 600: Σταθερά που χρησιμοποιείται για τον μετασχηματισμό των παλμών σε RPM
- *long prevTA = 0, prevTB = 0:* Μεταβλητές αποθήκευσης χρόνου κατά την προηγούμενη επανάληψη για κάθε μοτέρ - χρησιμοποιούνται για τον υπολογισμό της ταχύτητας.
- *int prevTempA =0, prevTempB = 0*: Μεταβλητές αποθήκευσης αριθμού παλμών κατά την προηγούμενη επανάληψη για κάθε μοτέρ - χρησιμοποιούνται για τον υπολογισμό της ταχύτητας.
- *volatile int senseA = 0, senseB = 0*: Μεταβλητές για την μέτρηση των παλμών για κάθε μοτέρ - χρησιμοποιούνται στις ρουτίνες διακοπής του κάθε μοτέρ.
- *float velFiltA = 0, velFiltB = 0*: Μεταβλητές για την αποθήκευση της ταχύτητας μετά την εφαρμογή του φίλτρου για κάθε μοτέρ.
- *float velPrevA = 0, velPrevB = 0*: Μεταβλητές για την αποθήκευση της τιμής της ταχύτητας πριν την εφαρμογή του φίλτρου κατά την προηγούμενη επανάληψη για κάθε μοτέρ.
- *int tempA = 0, tempB = 0*: Μεταβλητές όπου αντιγράφονται οι τιμές των *SenseA και SenseB* για την διεξαγωγή υπολογισμών στο κυρίως πρόγραμμα.
- *static int target = 0*: Μεταβλητή όπου αποθηκεύεται ο επιθυμητός αριθμός παλμών.
- *static int velTarget = 5*: Μεταβλητή καθορισμού επιθυμητής μέγιστης ταχύτητας κίνησης.
- *SoftwareSerial blue(Rx_pin, Tx_pin)*: Δημιουργία αντικειμένου κλάσης SoftwareSerial - χρησιμοποιείται για την επικοινωνία με την πλακέτα διεπαφής Bluetooth, όπου Rx_pin, Tx_pin τα επιθυμητά pins που θα προσομοιώσουν την αντίστοιχη λειτουργία.
- *MyPID pid_A, pid_B*: Δημιουργία αντικειμένων κλάσης MyPID για κάθε μοτέρ . Χρησιμοποιείται για την εφαρμογή ελέγχου PID.

9.2 Ρουτίνες Arduino IDE

Στο κεφάλαιο αυτό εξηγούνται οι ρουτίνες που χρησιμοποιήθηκαν οι οποίες προσφέρονται από το αναπτυξιακό του Arduino.

9.2.1 pinMode(pin, mode)

Καθορισμός συμπεριφοράς pin ως είσοδο ή ως έξοδο

Τιμές εισόδου:

pin: το επιθυμητό pin του Arduino προς καθορισμό.

mode: Ορισμός συμπεριφοράς του pin, πιθανές τιμές:

- INPUT - είσοδος.
- OUTPUT - έξοδος.
- INPUT_PULLUP - είσοδος με αντίσταση συνδεδεμένη στα +5V.

Τιμές επιστροφής:

Καμία.

9.2.2 attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)

Ενεργοποιεί την λειτουργία των εξωτερικών διακοπών για το συγκεκριμένο pin. Για τον Arduino Uno τα pins με αυτή τη δυνατότητα είναι το pin 2 και το pin 3. Η παράμετρος digitalPinToInterrupt χρησιμοποιείται διότι είναι ο συνιστώμενος τρόπος χρήσης της ρουτίνας.

Τιμές εισόδου:

pin: Επιθυμητό pin.

ISR: Επιθυμητή ρουτίνα που θα εκτελεστεί κατά την διακοπή του προγράμματος.

mode: Καθορίζει πότε θα εκτελεστεί η διακοπή, πιθανές τιμές:

- LOW - εκτέλεση της διακοπής όταν στο pin ανιχνεύεται χαμηλό δυναμικό.
- CHANGE - εκτέλεση της διακοπής όταν το pin αλλάζει τιμή.
- RISSING - εκτέλεση της διακοπής όταν το pin μεταβαίνει από χαμηλό δυναμικό σε υψηλό.
- FALLING - εκτέλεση της διακοπής όταν το pin μεταβαίνει από υψηλό δυναμικό σε χαμηλό.

Τιμές επιστροφής:

Καμία

9.2.3 digitalWrite(pin, value)

Αν το pin έχει οριστεί ως έξοδος, με την χρήση της pinMode, τότε ρυθμίζει την τάση του pin, 5V για HIGH, 0V για LOW. Αν το pin έχει οριστεί ως είσοδος τότε ενεργοποιεί (HIGH) ή απενεργοποιεί(LOW) την εσωτερική αντίσταση pullup του pin.

Τιμές εισόδου:

pin: Το pin του Arduino.

value: Επιθυμητή κατάσταση pin, πιθανές τιμές:

- HIGH.
- LOW.

Τιμές επιστροφής:

Καμία.

9.2.4 time = micros()

Επιστρέφει τον χρόνο σε μικρο δευτερόλεπτα από την στιγμή που ο Arduino ξεκίνησε την εκτέλεση του προγράμματος. Η τιμή κάνει υπερχείλιση και επιστρέφει στο μηδέν κάθε εβδομήντα λεπτά περίπου.

Τιμές εισόδου:

Καμία.

Τιμές επιστροφής:

time: Ο χρόνος σε μικρο δευτερόλεπτα από την εκκίνηση του προγράμματος.

9.2.5 analogWrite(pin, value)

Γράφει μια αναλογική τιμή(παλμοσειρά PWM) στην έξοδο.

Τιμές εισόδου:

pin: Το επιθυμητό pin του Arduino.

value: Ο κύκλος εργασίας(duty cycle) - η τιμή κυμαίνεται από 0 μέχρι 255.

Τιμές επιστροφής:

Καμία.

9.2.6 value = abs(number)

Επιστρέφει την απόλυτη τιμή ενός ακέραιου αριθμού.

Τιμές εισόδου:

number: Αριθμός για υπολογισμό απόλυτης τιμής.

Τιμές επιστροφής:

value: Η απόλυτη τιμή του αριθμού.

9.2.7 value = fabs(number)

Επιστρέφει την απόλυτη τιμή ενός δεκαδικού αριθμού.

Τιμές εισόδου:

number: Αριθμός για υπολογισμό απόλυτης τιμής.

Τιμές επιστροφής:

value: Η απόλυτη τιμή του αριθμού.

9.2.8 void setup()

Χρησιμοποιείται για να οριστεί ο τύπος λειτουργίας, είσοδος ή έξοδος, των pins του μικρο επεξεργαστή που θα χρησιμοποιηθούν, να γίνει η αρχικοποίηση μεταβλητών, έναρξη χρήσης βιβλιοθηκών κ.α. Εκτελείται μόνο μία φορά μετά από κάθε επανεκκίνηση του μικρο επεξεργαστή.

Τιμές εισόδου:

Καμία.

Τιμές επιστροφής:

Καμία.

9.2.9 void loop()

Εκτελείται πάντα μετά την *setup.* Είναι ο κύριος βρόχος επανάληψης του μικρο επεξεργαστή όπου τρέχουν οι επιθυμητοί αλγόριθμοι ελέγχου.

Τιμές εισόδου:

Καμία.

Τιμές επιστροφής:

Καμία.

9.3 Κλάσεις Arduino IDE

Για την ανάπτυξη του κώδικα χρησιμοποιήθηκαν ορισμένες κλάσεις που παρέχονται από το αναπτυξιακό του Arduino. Οι κλάσεις αυτές είναι οι *Serial* και *SoftwareSerial* και παρέχουν μεθόδους για την αρχικοποίηση και ανταλλαγή δεδομένων μέσο σειριακής επικοινωνίας. Η διαφορά των δύο είναι ότι η πρώτη, *Serial*, ελέγχει την κύρια σειριακή επικοινωνία του μικρο επεξεργαστή, που αντιστοιχεί στα pins 0(Rx) και 1(Tx) και έχει την δυνατότητα ανταλλαγής πληροφοριών και μέσω της ενσωματωμένης διεπαφής USB που έχει ο μικρο επεξεργαστής. Η δεύτερη, *SoftwareSerial,* παρέχει την δυνατότητα της προσομοίωσης σειριακής επικοινωνίας σε δύο οποιαδήποτε ψηφιακά pins του Arduino και δεν παρέχει την δυνατότητα επικοινωνίας μέσω της ενσωματωμένης διεπαφής USB του μικρο επεξεργαστή. Η *SoftwareSerial* χρησιμοποιήθηκε διότι η κύρια σειριακή διεπαφή του μικρο επεξεργαστή χρησιμοποιείται για την φόρτωση του προγράμματος και η επικοινωνία με κάποιο άλλο περιφερειακό μπορεί να δημιουργήσει πρόβλημα. Οι δύο αυτές κλάσεις προσφέρουν τις ίδιες μεθόδους και ο διαχωρισμός γίνεται από το πρόθεμα, για παράδειγμα η *Serial.begin(9600)*, αναφέρεται στην κύρια μονάδα σειριακής επικοινωνίας, ενώ η *X.begin(9600)*, όπου X το όνομα που δόθηκε κατά την δημιουργία αντικειμένου *SoftwareSerial*, αναφέρεται στην προσομοιωμένη σειριακή επικοινωνία. Παρακάτω γίνεται η επεξήγηση των μεθόδων αυτών.

9.3.1 begin(speed)

Χρησιμοποιείται για την αρχικοποίηση της σειριακής επικοινωνίας στην επιθυμητή ταχύτητα.

Τιμές εισόδου:

speed: Η επιθυμητή ταχύτητα επικοινωνίας.

Τιμές επιστροφής:

Καμία.

9.3.2 println(val)

Εκτύπωση δεδομένων στην σειριακή θύρα σε μορφή ASCII και στην συνέχεια στέλνει τους χαρακτήρες επιστροφής και νέας γραμμής.

Τιμές εισόδου:

val: Τα δεδομένα προς αποστολή.

Τιμές επιστροφής:

Ο αριθμός των bytes που γράφτηκαν.

9.3.3 available()

Επιστρέφει τον αριθμό των bytes που είναι διαθέσιμα για ανάγνωση από την σειριακή θύρα.

Τιμές εισόδου:

Καμία

Τιμές επιστροφής:

Αριθμός bytes προς ανάγνωση από την σειριακή θύρα.

9.3.4 readBytes(buffer, length)

Διαβάζει χαρακτήρες από την σειριακή θύρα και τους αποθηκεύει σε συγκεκριμένη θέση μνήμης. Η μέθοδος σταματάει το διάβασμα όταν έχει φτάσει το επιθυμητό μέγεθος μνήμης.

Τιμές εισόδου:

buffer: Η επιθυμητή θέση μνήμης.

length: Το επιθυμητό μέγεθος προς ανάγνωση.

Τιμές επιστροφής:

Ο αριθμός των bytes που τοποθετήθηκαν στην μνήμη.

9.3.5 parseInt()

Περιμένει τον επόμενο έγκυρο ακέραιο αριθμό που θα ληφθεί από την σειριακή επικοινωνία.

Τιμές εισόδου:

Καμία.

Τιμές επιστροφής:

Ο επόμενος έγκυρος ακέραιος αριθμός.

9.4 Κώδικας που αναπτύχθηκε

Για την υλοποίηση του συστήματος ήταν απαραίτητο να δημιουργηθούν κάποιες συναρτήσεις και κλάσεις πέραν από αυτές που παρέχονται με το αναπτυξιακό του Arduino. Παρακάτω γίνεται η επεξήγηση του κώδικα που αναπτύχθηκε.

9.4.1 void start(char dir)

Η συνάρτηση αυτή ευθύνεται για να ξεκινήσει το ρομπότ να κινείται προς την επιθυμητή κατεύθυνση και να σταματήσει μόλις έχει μετρηθεί ότι ο αριθμός περιστροφών που έχουν εκτελέσει οι ρόδες είναι ίσος ή μεγαλύτερος από τον αριθμό που έχει ληφθεί μέσω της διεπαφής Bluetooth.

Τιμές εισόδου:

dir: Η επιθυμητή πορεία που θα εκτελέσει το ρομπότ, πιθανές τιμές:

- f: κίνηση ευθεία μπροστά.
- b: κίνηση ευθεία προς τα πίσω.
- l: στροφή προς τα αριστερά.
- r: στροφή προς τα δεξιά.

Τιμές επιστροφής:

Καμία.

9.4.2 void setDirection(char dir)

Η συνάρτηση χρησιμοποιείται για την ρύθμιση του οδηγού L298P ώστε οι ρόδες να έχουν την κατάλληλη φορά περιστροφής για να εκτελεστεί η επιθυμητή πορεία.

Τιμές εισόδου:

dir: Η επιθυμητή πορεία που θα εκτελέσει το ρομπότ, πιθανές τιμές:

- f: κίνηση ευθεία μπροστά.
- b: κίνηση ευθεία προς τα πίσω.
- l: στροφή προς τα αριστερά.
- r: στροφή προς τα δεξιά.

Τιμές επιστροφής:

Καμία.

9.4.3 void SensorA()

Ρουτίνα διακοπής που μετράει των αριθμό περιστροφών του μοτέρ Α.

Τιμές εισόδου:

Καμία.

Τιμές επιστροφής:

Καμία.

9.4.4 void SensorB()

Ρουτίνα διακοπής που μετράει των αριθμό περιστροφών του μοτέρ Β.

Τιμές εισόδου:

Καμία.

Τιμές επιστροφής:

Καμία.

9.4.5 Κλάση MyPid

Η κλάση MyPid δημιουργήθηκε ώστε να γίνει έλεγχος της ταχύτητας των μοτέρ του ρομπότ με την μέθοδο PID. Παρακάτω αναλύονται τα μέλη της κλάσης.

9.4.5.1 Ιδιωτικές μεταβλητές κλάσης

- *float kp, ki, kd*: Μεταβλητές αποθήκευσης ποσοστού ενίσχυσης P, I και D αντίστοιχα
- *float signalMax*: μέγιστη τιμή σήμα ελέγχου ταχύτητας
- *float errorPrev*: Σφάλμα κατά την προηγούμενη εκτέλεση του αλγορίθμου PID - χρησιμοποιείται για τον υπολογισμό του σφάλματος διαφόρισης.
- *float errorInteg*: Χρησιμοποιείται για τον υπολογισμό του σφάλματος ολοκλήρωσης I

9.4.5.2 MyPID(): kp(1), ki(0), kd(0), signalMax(255), errorInteg(0.0)

Δομητής κλάσης, χρησιμοποιείται κατά την δημιουργία αντικειμένου MyPID. Κατά την δημιουργία του αντικειμένου γίνεται η αρχικοποίηση των παρακάτω μεταβλητών:

- *kp* = 1.
- *ki = 0.*
- *kd* = 0.
- *signalMax* = 255.
- *errorInteg* = 0,0.

Τιμές εισόδου:

Καμία.

Τιμές εξόδου:

Καμία.

9.4.5.3 void setParams(float kpIn, float kiIn, float kdIn, float signalMaxIn)

Μέθοδος για ορισμό επιθυμητών τιμών *kp, ki, kd* και *signalMax*.

Τιμές εισόδου:

kpIn: Επιθυμητή τιμή kp.

kiIn: Επιθυμητή τιμή ki.

kdIn: Επιθυμητή τιμή kd.

signalMaxIn: Επιθυμητή τιμή signalMax.

Τιμές εξόδου:

Καμία.

9.4.5.4 void clearErrors(void)

Μέθοδος για μηδενισμό σφαλμάτων παραγώγισης και ολοκλήρωσης.

Τιμές εισόδου:

Καμία.

Τιμές επιστροφής:

Καμία.

9.4.5.5 void evalu (int value, int target, float deltaT, int &power)

Μέθοδος όπου υπολογίζει το σήμα ελέγχου της ταχύτητας περιστροφής του μοτέρ ώστε να επιτευχθεί η επιθυμητή ταχύτητα με τη βοήθεια του αλγορίθμου PID.

Τιμές εισόδου:

value: Η τρέχουσα ταχύτητα - χρησιμοποιείται για τον υπολογισμό του αναλογικού σφάλματος.

target: Η επιθυμητή ταχύτητα - χρησιμοποιείται για τον υπολογισμό του αναλογικού σφάλματος.

deltaT: Χρόνος που μεσολάβησε από την προηγούμενη εκτέλεση της μεθόδου. Χρησιμοποιείται για τον υπολογισμό των σφαλμάτων διαφόρισης και ολοκλήρωσης.

Τιμές επιστροφής:

power: Σήμα ελέγχου ταχύτητας, εύρος τιμών [0, 255].

9.5 Περιγραφή κώδικα

Στο κεφάλαιο αυτό γίνεται η ανάλυση του κώδικα που αναπτύχθηκε με το αναπτυξιακό Arduino IDE. Αφού έχει γίνει η εισαγωγή των βιβλιοθηκών και η δημιουργία των κατάλληλων μεταβλητών και κλάσεων ξεκινάει το κυρίως μέρος του κώδικα. Το κύριο μέρος του κώδικα απαρτίζεται από δύο μέρη, αυτό της αρχικοποίησης και ο βρόχος επανάληψης όπου μέσα του γίνεται η ανάπτυξη του επιθυμητού αλγόριθμου ελέγχου. Το μέρος της αρχικοποίησης εκτελείται με την συνάρτηση *void setup()*. Αρχικά αρχικοποιούνται οι σειριακές επικοινωνίες του επεξεργαστή όπως φαίνεται παρακάτω,

```
Serial.begin(9600);
blue.begin(9600);
```

Το πρόθεμα *Serial* αντιστοιχεί στην κύρια σειριακή επικοινωνία του επεξεργαστή, που χρησιμοποιείται για αποσφαλμάτωση του κώδικα και τον προγραμματισμό του επεξεργαστή και το πρόθεμα *blue* αντιστοιχεί στην προσομοιωμένη σειριακή επικοινωνία που χρησιμοποιείται για την επικοινωνία του επεξεργαστή με την διεπαφή bluetooth. Έπειτα γίνεται ο ορισμός της συμπεριφοράς των pins που ευθύνονται για τον έλεγχο των μοτέρ,

```
//Αρχικοποίηση pins μοτέρA
pinMode(ENC1_A, INPUT);
pinMode(ENC2_A, INPUT);
pinMode(PWM_A, OUTPUT);
pinMode(IN_A, OUTPUT);

//Αρχικοποίηση pins μοτέρB
pinMode(ENC1_B, INPUT);
pinMode(ENC2_B, INPUT);
pinMode(PWM_B, OUTPUT);
pinMode(IN_B, OUTPUT);
```

Στο επόμενο βήμα γίνεται η αρχικοποίηση της λειτουργίας εξωτερικής διακοπής που παρέχουν τα pins *ENC1_A* και *ENC1_B* με την χρήση των παρακάτω εντολών:

```
attachInterrupt(digitalPinToInterrupt(ENC1_A), SensorA, RISING);
attachInterrupt(digitalPinToInterrupt(ENC1_B), SensorB, RISING);
```

Με τον τρόπο αυτόν όταν εντοπίζεται ανερχόμενη παρυφή, από ψηφιακή στάθμη 0 σε ψηφιακή στάθμη 1, σε ένα από τα pins *ENC1_A* ή *ENC1_B* διακόπτεται η κανονική εκτέλεση του κύριου βρόχου του προγράμματος και μεταβαίνει στην εκτέλεση της ρουτίνας που αντιστοιχεί σε κάθε pin, *SensorA* και *SensorB* αντίστοιχα. Οι ρουτίνες *SensorA* και *SensorB* αυξάνουν κατά ένα τον αριθμό των παλμών που έχουν εκτελέσει τα αντίστοιχα μοτέρ όπως φαίνεται παρακάτω:

```
void SensorA()
```

```
{
  senseA = senseA + 1;
}

void SensorB()
{
  senseB = senseB + 1;
}
```

Στην συνέχεια της αρχικοποίησης ορίζεται η αρχική φορά περιστροφής των μοτέρ, σύμφωνα με τον Πίνακα 2, με τις παρακάτω εντολές:

```
digitalWrite(IN_A, HIGH);
digitalWrite(IN_B, HIGH);
```

Τέλος γίνεται η αρχικοποίηση των παραμέτρων PID για το κάθε μοτέρ με τις εντολές:

```
pid_A.setParams(30, 12, 0, 255);
pid_B.setParams(25, 14, 0.08, 255);
```

Για το μοτέρ Α οι παράμετροι ορίζονται ως εξής KP = 30, KI = 12, KD = 0, signalMax = 255. Για το μοτέρ Β οι παράμετροι ορίζονται ως εξής KP = 25, KI = 14, KD = 0.08, signalMax = 255.

Αφού έχει ολοκληρωθεί το μέρος της αρχικοποίησης σειρά έχει το μέρος του βρόχου επανάληψης. Ο βρόχος επανάληψης εκτελείται με την συνάρτηση *void loop()*. Αρχικά δημιουργείται μία τοπική μεταβλητή όπου θα χρησιμοποιηθεί για να αποθηκευτεί η πληροφορία της κατεύθυνσης του ρομπότ.

```
char command;
```

Στην συνέχεια στέλνεται η τιμή '1' μέσω της διεπαφής bluetooth, όπου σηματοδοτεί ότι το ρομπότ αναμένει εντολή κατεύθυνσης και περιμένει την απάντηση.

```
blue.println(1, DEC);
while (!blue.available());
```

Μόλις η διεπαφή bluetooth λάβει την πληροφορία κατεύθυνσης τότε αυτή διαβάζεται από τον επεξεργαστή με την εντολή *blue.readBytes(&command, 1);* και τα δεδομένα αποθηκεύονται στην μεταβλητή *command*. Στην συνέχεια σύμφωνα με την τιμή της μεταβλητής *command* ορίζεται η μέγιστη επιθυμητή ταχύτητα, 4 για ευθεία κίνηση 2 για στροφή, και εκτελείται η ρουτίνα *start*, όπου δέχεται ως είσοδο την τιμή της κατεύθυνσης, όπως φαίνεται στο παρακάτω κομμάτι κώδικα.

```
 switch (command)
 {
   case 'f':
```

```
    {
      velTarget = 4;
      start(command);
      break;
    }
  case 'b':
    {
      velTarget = 4;
      start(command);
      break;
    }
  case 'r':
    {
      velTarget = 2;
      start(command);
      break;
    }
  case 'l':
    {
      velTarget = 2;
      start(command);
      break;
    }
  default:
    {
      blue.println("Wrong Input");
      break;
    }
}
```

Οι διαφορετικές ταχύτητες οφείλονται στο γεγονός ότι παρατηρήθηκε πειραματικά πως το ρομπότ είχε μεγαλύτερη ακρίβεια στην εκτέλεση περιστροφικής κίνησης με χαμηλότερη ταχύτητα.

Μόλις ξεκινήσει η εκτέλεση της ρουτίνας *start* στέλνεται η τιμή '2' μέσω της διεπαφής

bluetooth, όπου σηματοδοτεί ότι το ρομπότ αναμένει των αριθμό παλμών που πρέπει να εκτελεστούν και περιμένει για απάντηση.

```
blue.println(2, DEC);
while (!blue.available());
```

Μόλις η διεπαφή bluetooth λάβει την πληροφορία για τον αριθμό παλμών τότε αυτή διαβάζεται από τον επεξεργαστή με την εντολή *target = blue.parseInt();* και τα δεδομένα αποθηκεύονται στην μεταβλητή *target.* Έπειτα καλείται η ρουτίνα *setDirection(dir);* όπου ρυθμίζει τα pins ελέγχου του οδηγού μοτέρ L298P ώστε το ρομπότ να κινηθεί προς την επιθυμητή κατεύθυνση. Στη συνέχεια γίνεται ο μηδενισμός των μεταβλητών που ευθύνονται για τον έλεγχο της ταχύτητας του ρομπότ όπως φαίνεται παρακάτω.

```
ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
{
  senseA = 0;
  senseB = 0;
}
tempA = 0;
tempB = 0;
velFiltA = 0;
velPrevA = 0;
velFiltB = 0;
velPrevB = 0;
prevTA = micros();
pid_A.clearErrors();
pid_B.clearErrors();
```

Η μεταβλητή *PrevTA* παίρνει την τιμή του χρόνου που πέρασε από την αρχή της εκτέλεσης του προγράμματος, *micros()*, διότι αν έχει την τιμή 0 δεν γίνεται σωστός υπολογισμός της ταχύτητας κατά την πρώτη εκτέλεση του βρόχου επανάληψης do-while. Στο επόμενο βήμα υπάρχει ένας βρόχος επανάληψης τύπου do-while όπου εκτελείται όσο ο αριθμός παλμών που έχουν εκτελέσει τα μοτέρ Α και μοτέρ Β είναι μικρότερος από τον επιθυμητό αριθμό παλμών. Εντός του βρόχου επανάληψης αρχικά δημιουργούνται τέσσερις τοπικές μεταβλητές όπως φαίνεται παρακάτω:

```
int pwrA = 0, pwrB = 0;
float velocityA = 0, velocityB = 0;
```

όπου *pwrA* και *pwrB* είναι το σήμα ελέγχου ταχύτητας για τα μοτέρ Α και μοτέρ Β αντίστοιχα. Οι μεταβλητές *velocityA* και *velocityB* χρησιμοποιούνται για τον υπολογισμό της ταχύτητας πριν την χρήση του χαμηλοπερατού φίλτρου. Έπειτα γίνεται η αντιγραφή των δεδομένων των μεταβλητών που χρησιμοποιούν οι ρουτίνες διακοπής σε προσωρινές

μεταβλητές ώστε να γίνει ο υπολογισμός της νέας ταχύτητας από το κυρίως πρόγραμμα όπως φαίνεται παρακάτω:

```
ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
{
  tempA = senseA;
  tempB = senseB;
}
```

Στη συνέχεια δημιουργούνται ακόμα δύο τοπικές μεταβλητές:

```
long currT = micros();
float deltaT = ((float) (currT - prevTA)) / 1.0e6;
```

όπου αποθηκεύονται, ο τρέχων χρόνος που πέρασε από την αρχή της εκτέλεσης του προγράμματος, *currT* και, η διαφορά χρόνου μεταξύ δύο διαδοχικών εκτελέσεων του βρόχου do-while, *deltaT.* Στο επόμενο βήμα γίνεται ο υπολογισμός της ταχύτητας περιστροφής του κάθε μοτέρ

```
velocityA = (tempA - prevTempA) / deltaT;
velocityB = (tempB - prevTempB) / deltaT;
```

και έπειτα αποθηκεύονται οι τρέχουσες τιμές του χρόνου και του αριθμού παλμών που εκτέλεσε το κάθε μοτέρ για την διεξαγωγή των υπολογισμών στην επόμενη εκτέλεση του βρόχου do-while.

```
prevTA = currT;
prevTempA = tempA;
prevTempB = tempB;
```

Πριν γίνει η εφαρμογή του φίλτρου γίνεται πρώτα η μετατροπή της ταχύτητας σε περιστροφές ανά λεπτό, PRM,

```
float velA = velocityA / 137;
float velB = velocityB / 137;
```

και στην συνέχεια εφαρμόζεται το φίλτρο για την ταχύτητα του κάθε μοτέρ

```
velFiltA = 0.854 * velFiltA + 0.0728 * velA + 0.0728 * velPrevA;
velPrevA = velA;


velFiltB = 0.854 * velFiltB + 0.0728 * velB + 0.0728 * velPrevB;
velPrevB = velB;
```

Αφού πλέον είναι γνωστές οι ταχύτητες και έχει εφαρμοστεί το φίλτρο μπορεί να γίνει εφαρμογή του ελεγκτή PID

```
pid_A.evalu(velFiltA, velTarget, deltaT, pwrA);
```

pid_B.evalu(velFiltB, velTarget, deltaT, pwrB);

και να εφαρμοστούν οι καινούργιες τιμές ταχύτητας σε κάθε μοτέρ

analogWrite(PWM_A, pwrA);

analogWrite(PWM_B, pwrB);

Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου και τα δύο μοτέρ να έχουν εκτελέσει περισσότερους παλμούς από τους επιθυμητούς για την διεξαγωγή της κίνησης.

while (tempB < target && tempA < target);

Το γεγονός ότι τα μοτέρ εκτελούν περισσότερους παλμούς από το επιθυμητό δεν δημιουργεί σοβαρό πρόβλημα στο σύστημα καθώς η απόκλιση από τον στόχο είναι μερικές δεκάδες παλμών και ο αριθμός μίας πλήρους περιστροφής είναι 600 παλμοί.

Αφού ολοκληρωθεί η εκτέλεση του βρόχου do-while, σταματάει η κίνηση των μοτέρ

analogWrite(PWM_A, 0);

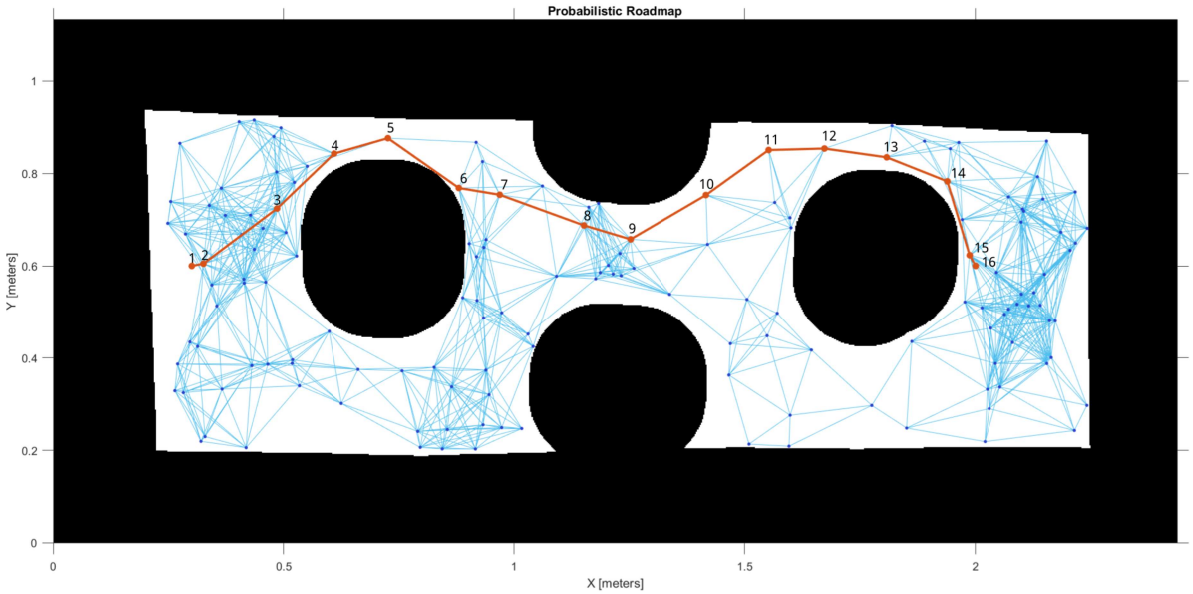
analogWrite(PWM_B, 0);

και γίνεται ο μηδενισμός των μεταβλητών που ευθύνονται για τον έλεγχο της ταχύτητας του ρομπότ

```
ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
{
  senseA = 0;
  senseB = 0;
}
tempA = 0;
tempB = 0;
prevTempA = 0;
prevTempB = 0;
velFiltA = 0;
velPrevA = 0;
velFiltB = 0;
velPrevB = 0;
```

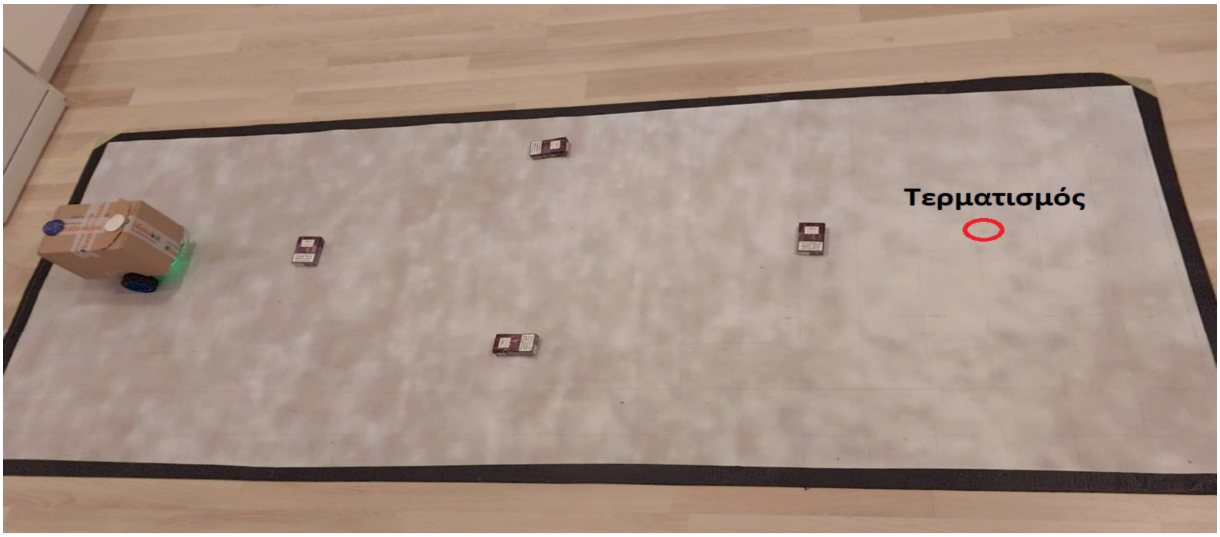
Έπειτα η συνάρτηση *loop()* εκτελείται ξανά από την αρχή.

Κεφάλαιο 10: Αποτελέσματα

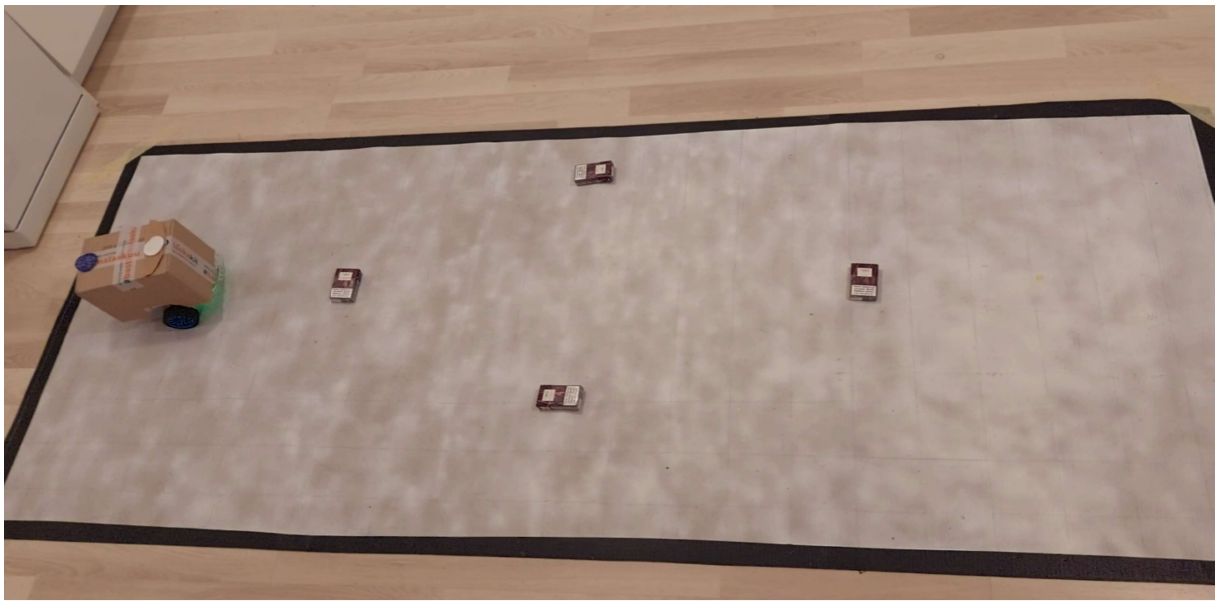


Εικόνα 18: Διαδρομή 1

Η Εικόνα 18 απεικονίζει την διαδρομή που υπολογίστηκε από την μέθοδο PRM για να μπορέσει το ρομπότ να φτάσει από το σημείο 1, που είναι η αφετηρία, στο σημείο 16, τον τερματισμό. Τα ενδιάμεσα σημεία, 2-15, είναι κόμβοι όπου το ρομπότ πρέπει να αλλάξει την κατεύθυνσή του. Στην Εικόνα 19 φαίνεται η αρχική τοποθέτηση του ρομπότ στον χώρο κίνησης καθώς και το σημείο του τερματισμού.

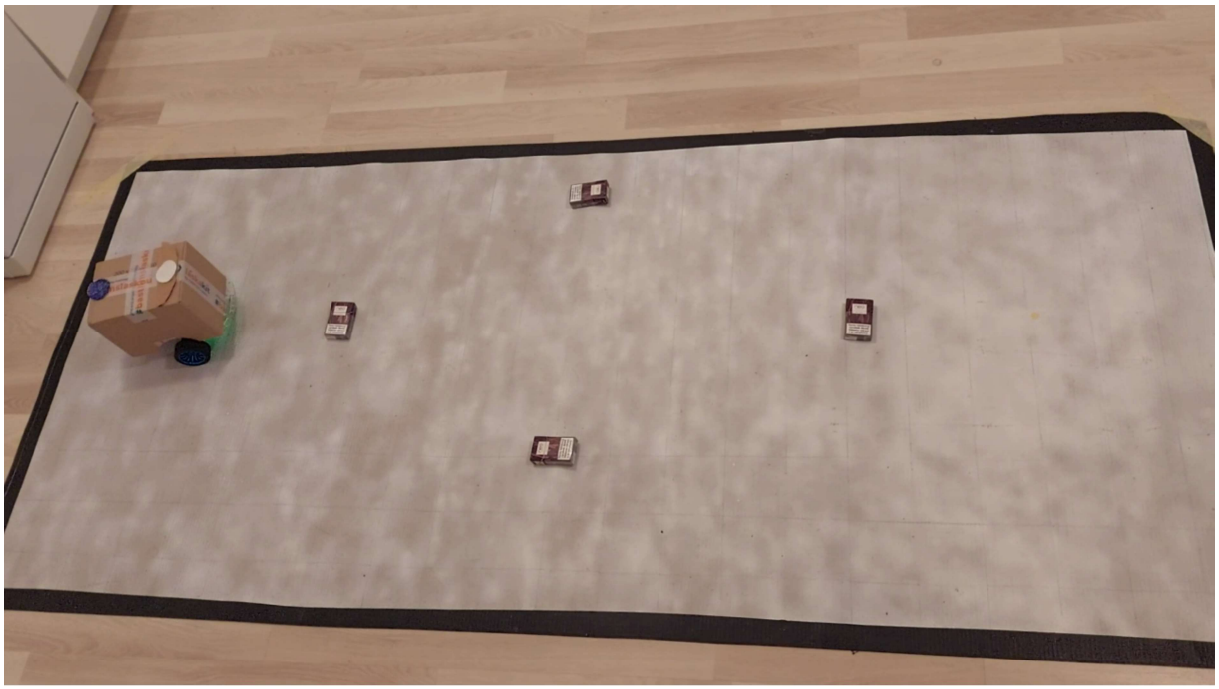


Εικόνα 19: Αρχική θέση ρομπότ, διαδρομή 1

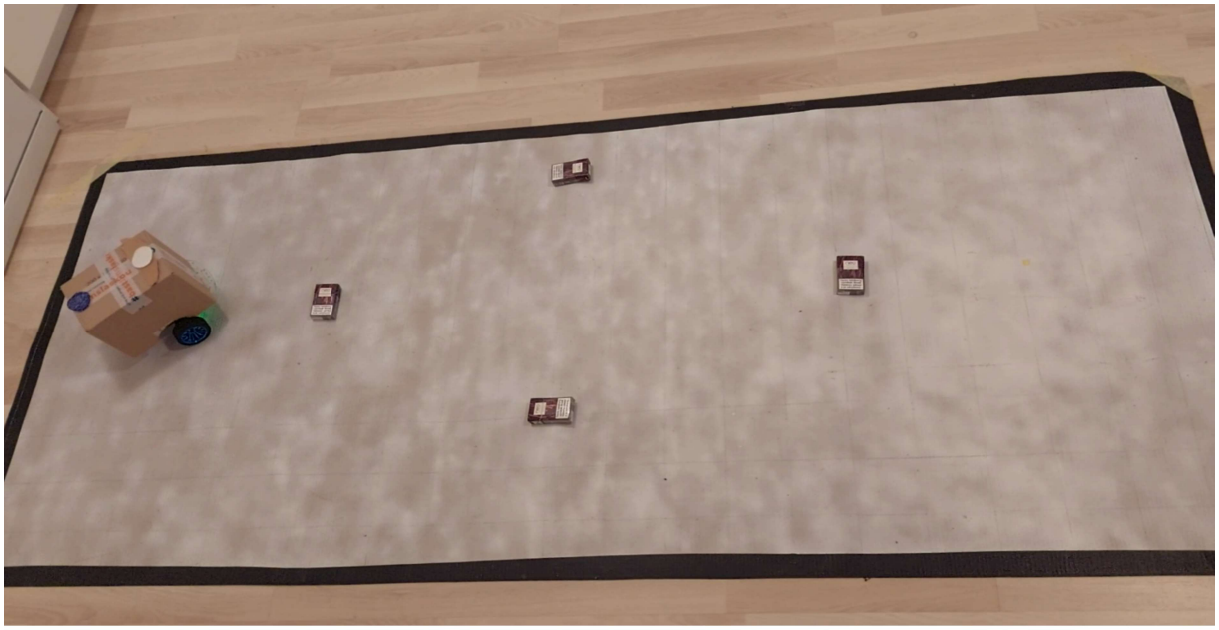


Εικόνα 20: Αλλαγή κατεύθυνσης ρομπότ για σημείο 2, διαδρομή 1

Στην Εικόνα 20 φαίνεται η αλλαγή της κατεύθυνσης που χρειάζεται να κάνει το ρομπότ για να φτάσει στο σημείο 2 και η Εικόνα 21 δείχνει το ρομπότ που έχει φτάσει το σημείο αυτό.

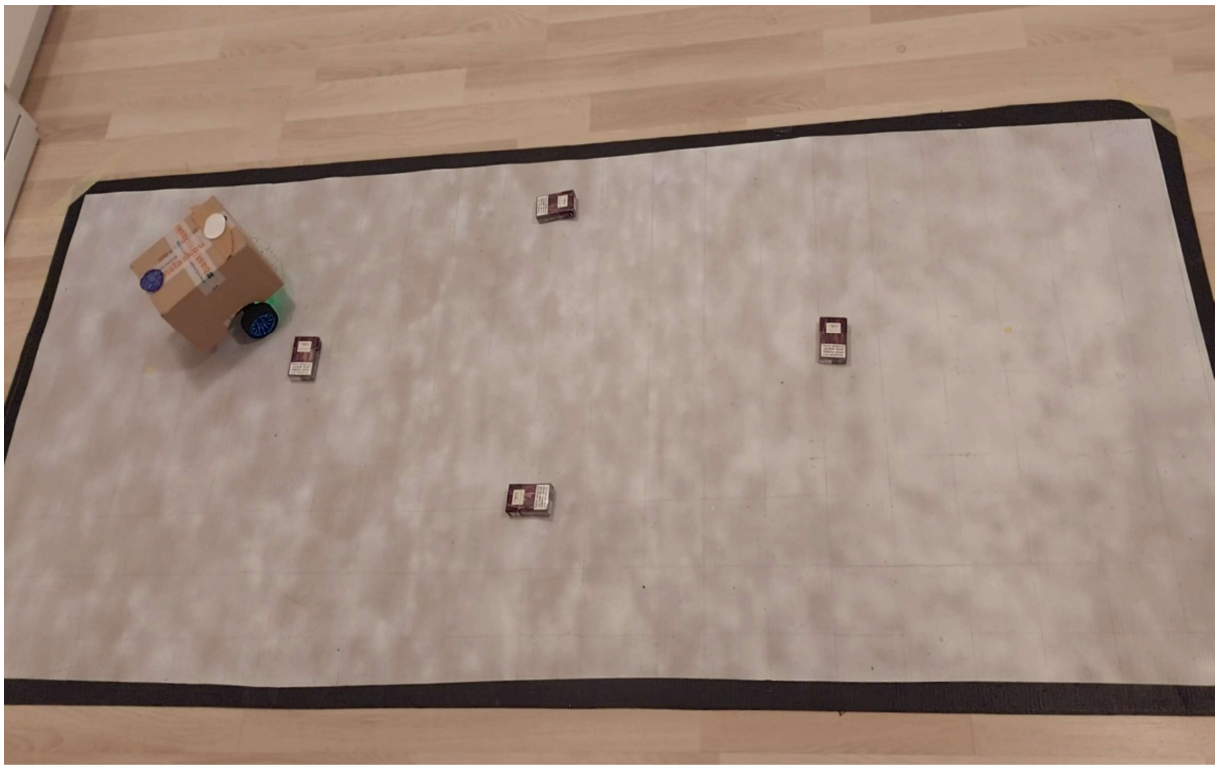


Εικόνα 21: Ρομπότ στο σημείο 2, διαδρομή 1

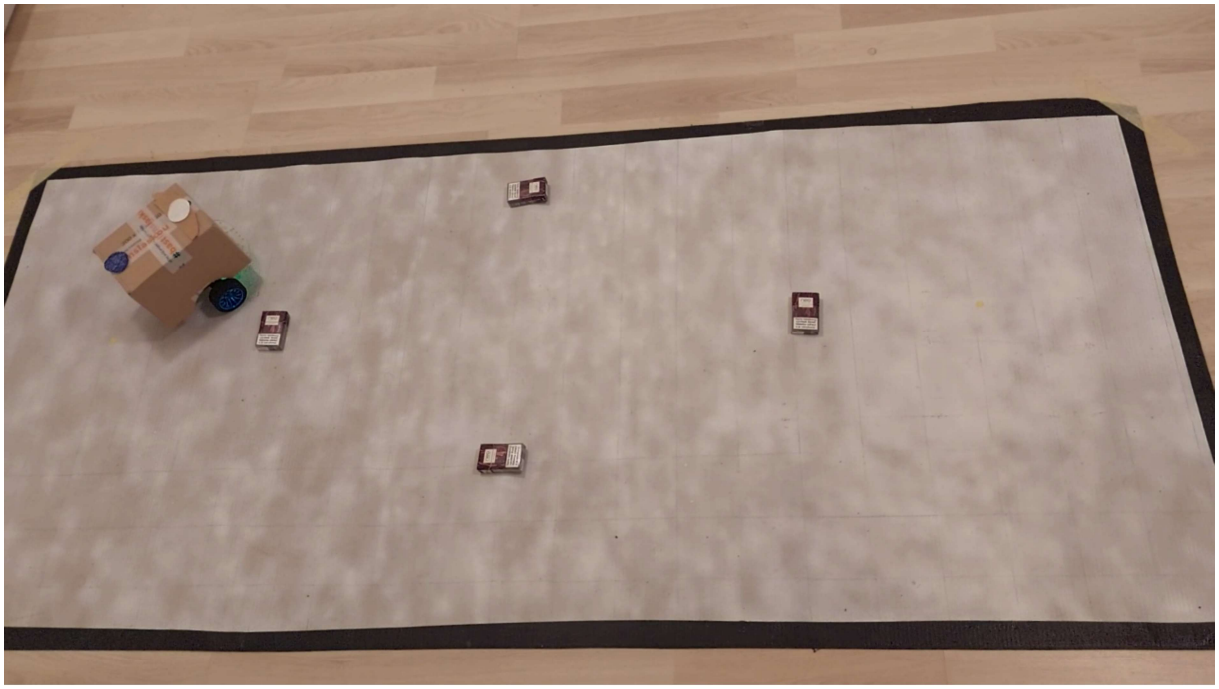


Εικόνα 22: Αλλαγή κατεύθυνσης ρομπότ για σημείο 3, διαδρομή 1

Στην Εικόνα 22 φαίνεται η αλλαγή της κατεύθυνσης που χρειάζεται να κάνει το ρομπότ για να φτάσει στο σημείο 3 και η Εικόνα 23 δείχνει το ρομπότ που έχει φτάσει το σημείο αυτό.

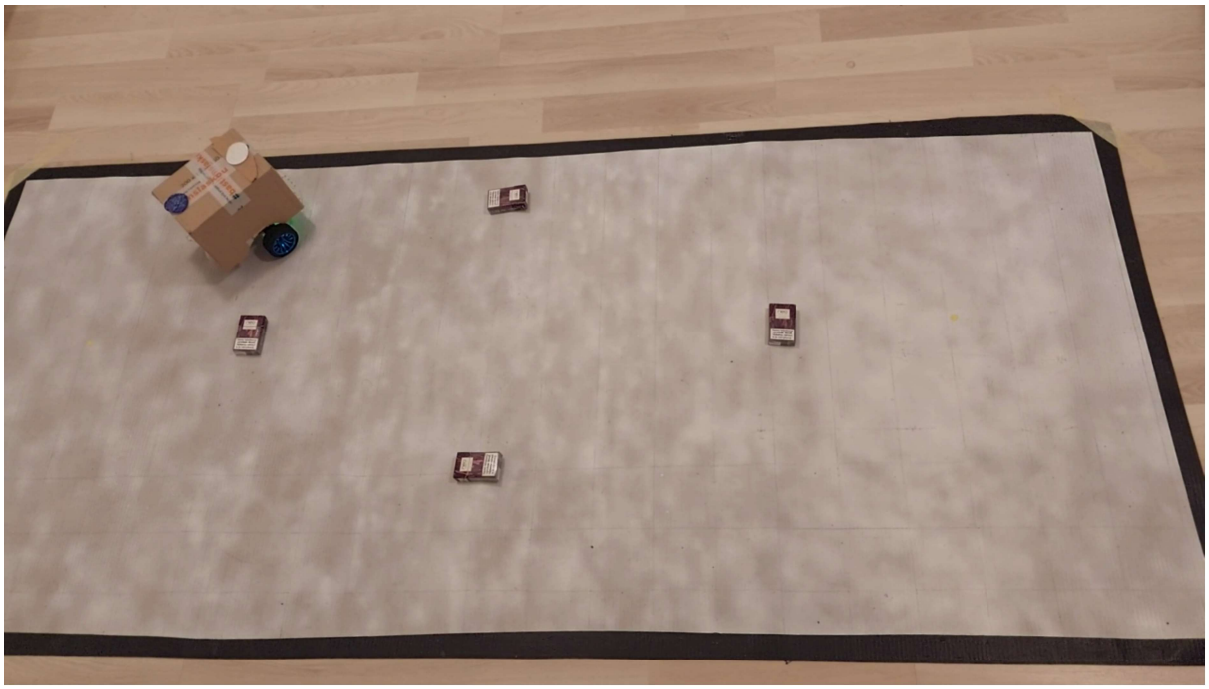


Εικόνα 23: Ρομπότ στο σημείο 3, διαδρομή 1

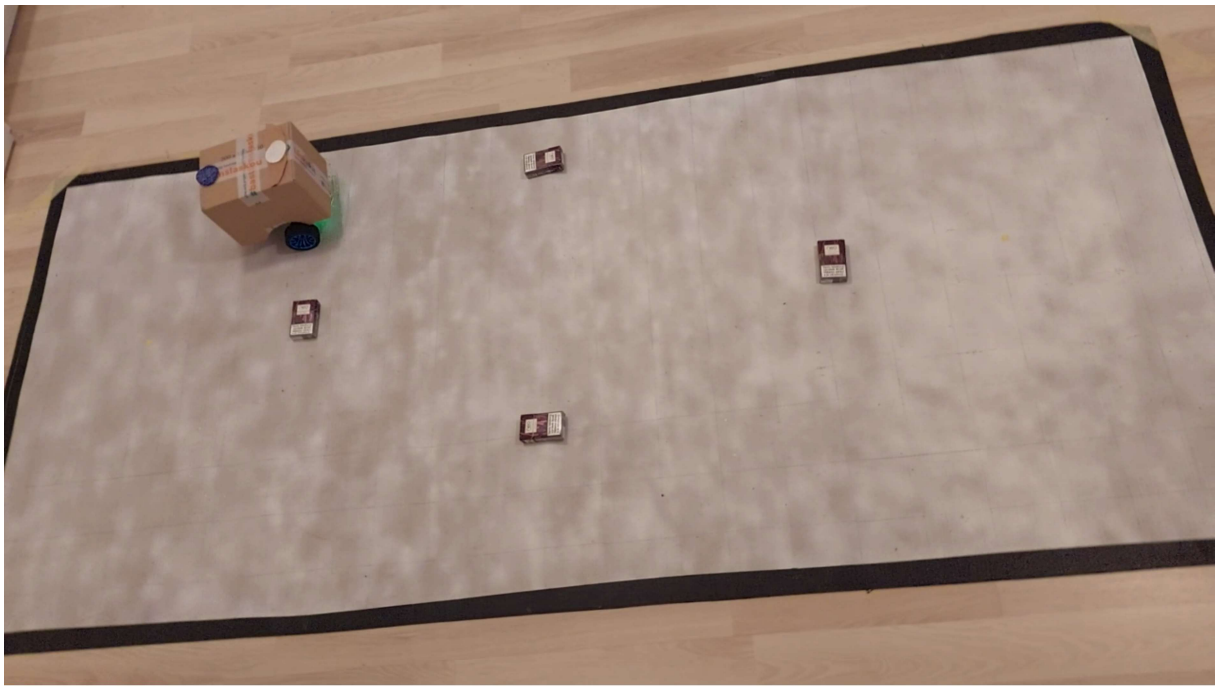


Εικόνα 24: Αλλαγή κατεύθυνσης ρομπότ για σημείο 4, διαδρομή 1

Στην Εικόνα 24 φαίνεται η αλλαγή της κατεύθυνσης που χρειάζεται να κάνει το ρομπότ για να φτάσει στο σημείο 4 και η Εικόνα 25 δείχνει το ρομπότ που έχει φτάσει το σημείο αυτό.

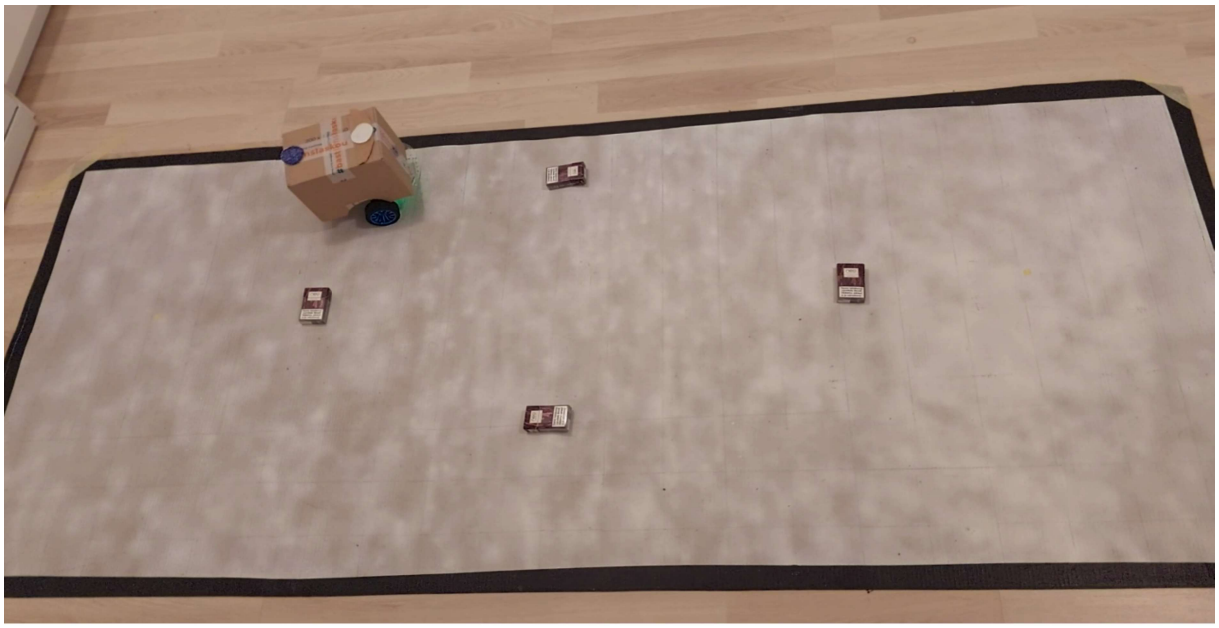


Εικόνα 25: Ρομπότ στο σημείο 4, διαδρομή 1

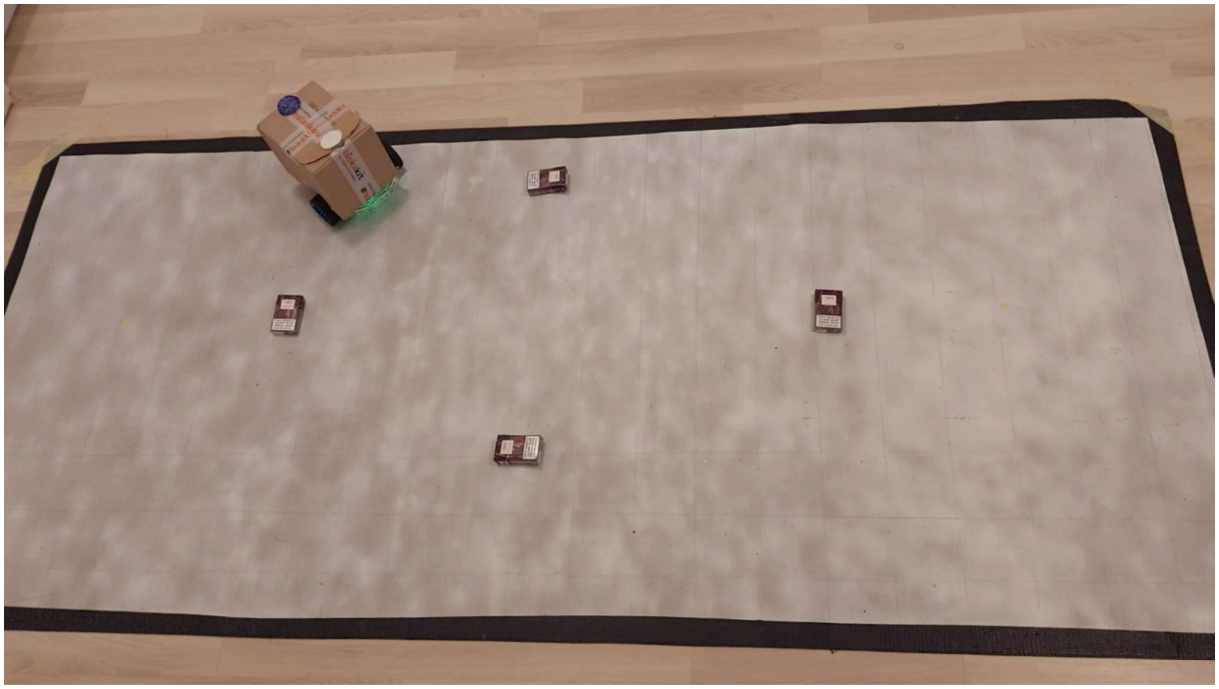


Εικόνα 26: Αλλαγή κατεύθυνσης ρομπότ για σημείο 5, διαδρομή 1

Στην Εικόνα 26 φαίνεται η αλλαγή της κατεύθυνσης που χρειάζεται να κάνει το ρομπότ για να φτάσει στο σημείο 5 και η Εικόνα 27 δείχνει το ρομπότ που έχει φτάσει το σημείο αυτό.

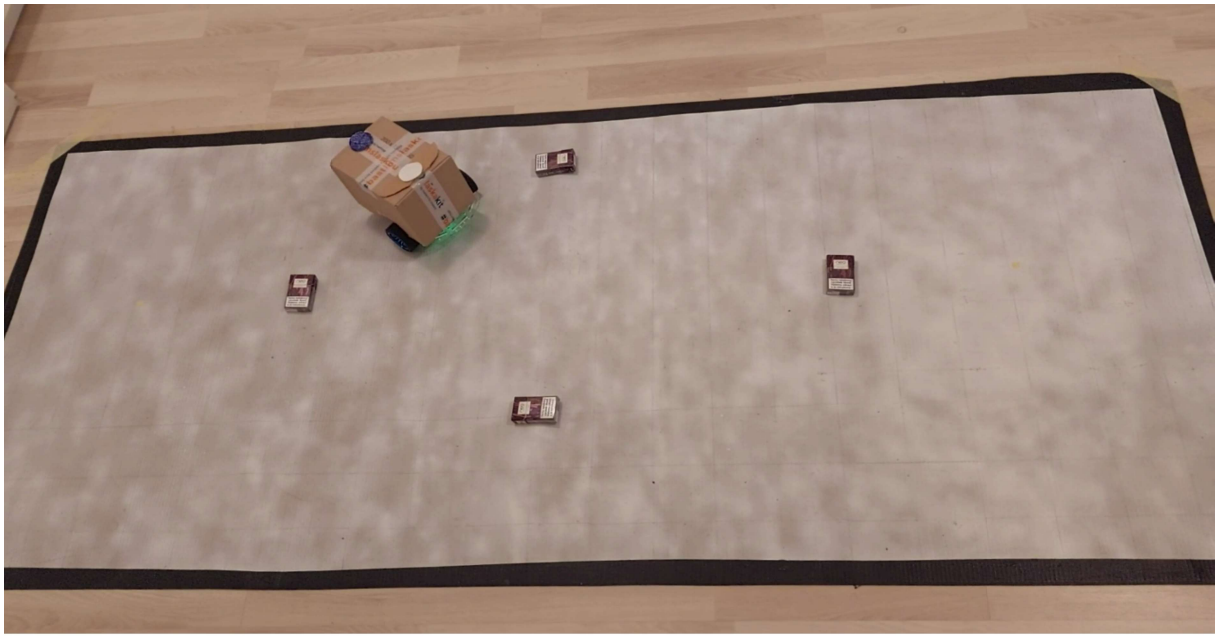


Εικόνα 27: Ρομπότ στο σημείο 5, διαδρομή 1

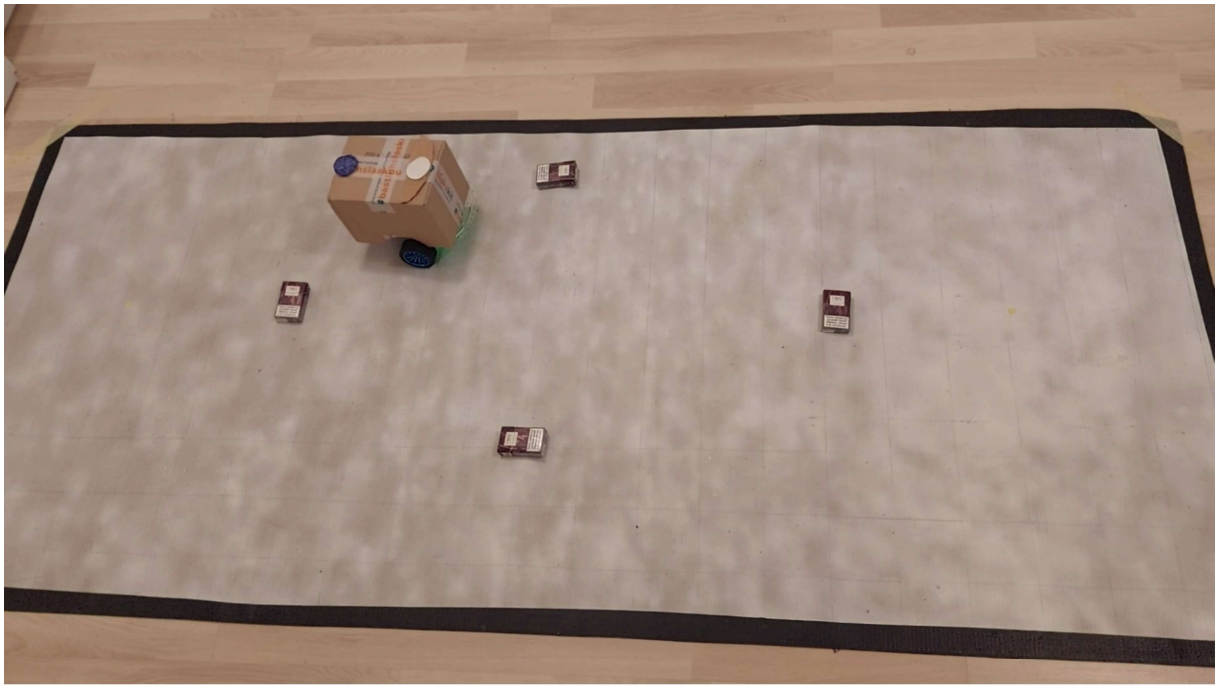


Εικόνα 28: Αλλαγή κατεύθυνσης ρομπότ για σημείο 6, διαδρομή 1

Στην Εικόνα 28 φαίνεται η αλλαγή της κατεύθυνσης που χρειάζεται να κάνει το ρομπότ για να φτάσει στο σημείο 6 και η Εικόνα 29 δείχνει το ρομπότ που έχει φτάσει το σημείο αυτό.

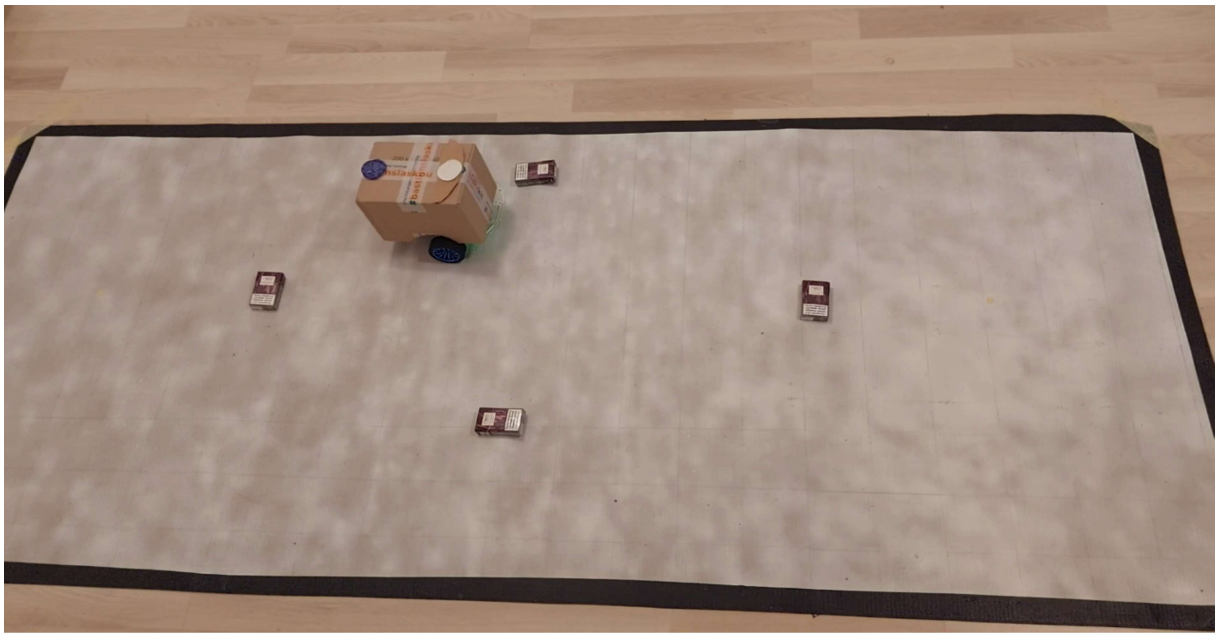


Εικόνα 29: Ρομπότ στο σημείο 6, διαδρομή 1

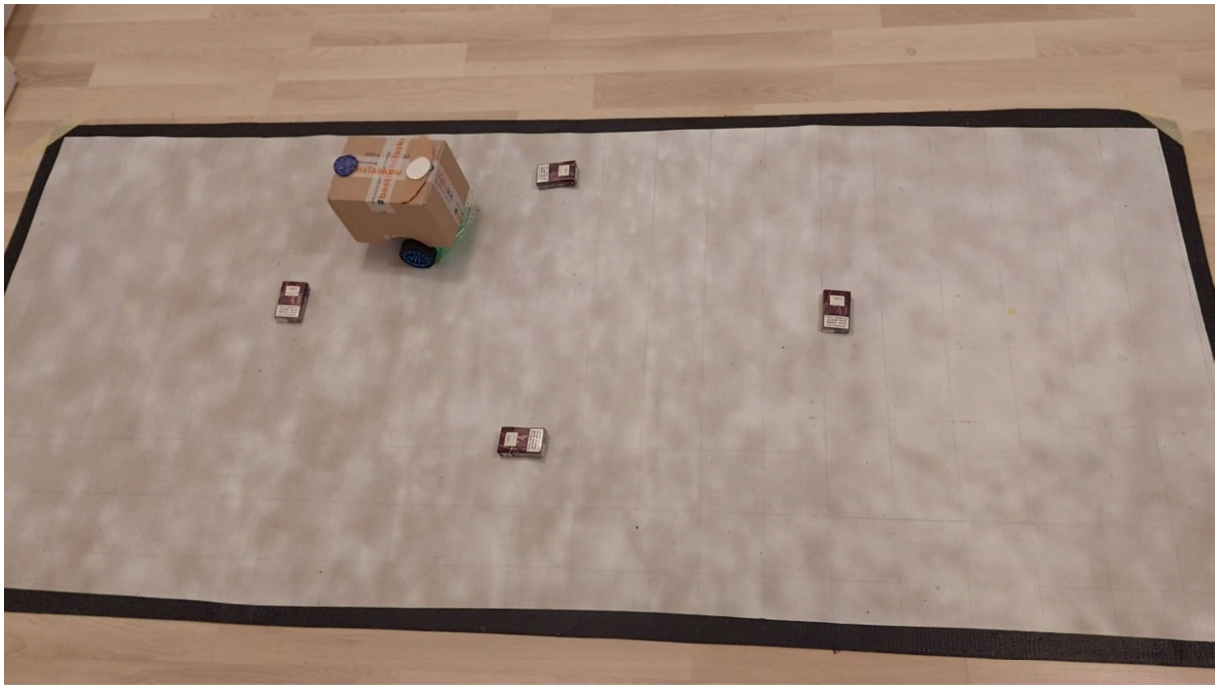


Εικόνα 30: Αλλαγή κατεύθυνσης ρομπότ για σημείο 7, διαδρομή 1

Στην Εικόνα 30 φαίνεται η αλλαγή της κατεύθυνσης που χρειάζεται να κάνει το ρομπότ για να φτάσει στο σημείο 7 και η Εικόνα 31 δείχνει το ρομπότ που έχει φτάσει το σημείο αυτό.

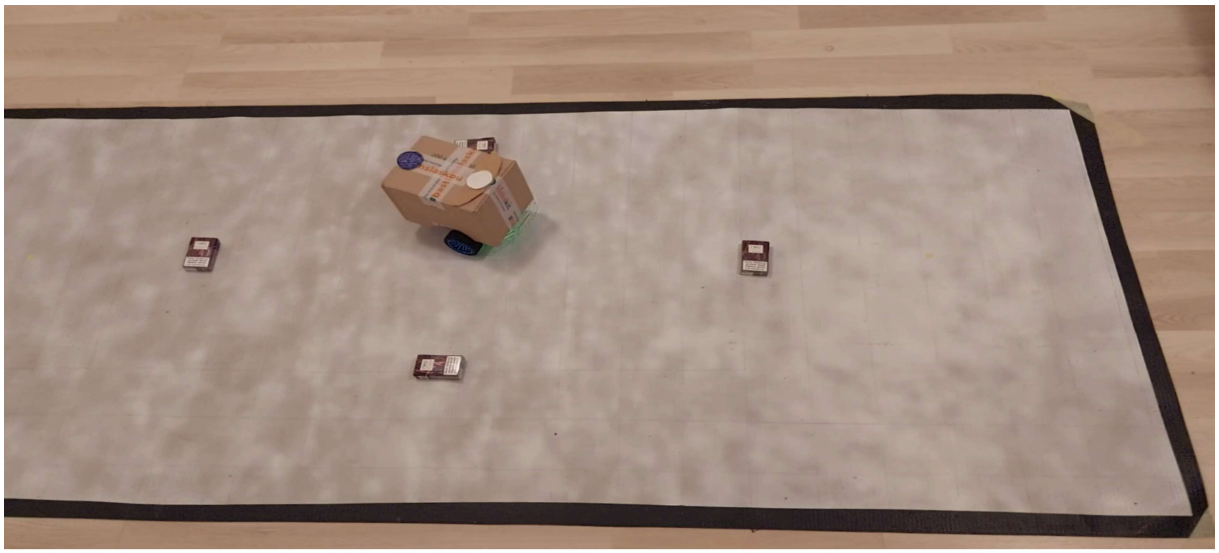


Εικόνα 31: Ρομπότ στο σημείο 7, διαδρομή 1

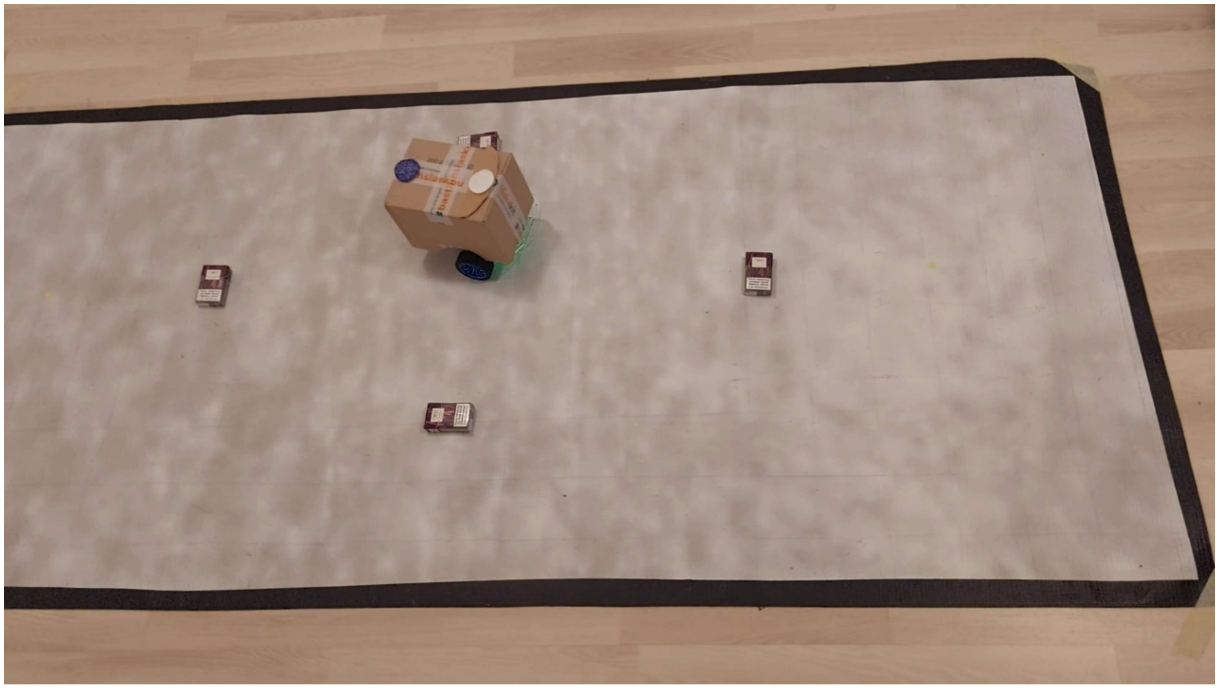


Εικόνα 32: Αλλαγή κατεύθυνσης ρομπότ για σημείο 7, διαδρομή 1

Στην Εικόνα 32 φαίνεται η αλλαγή της κατεύθυνσης που χρειάζεται να κάνει το ρομπότ για να φτάσει στο σημείο 8 και η Εικόνα 33 δείχνει το ρομπότ που έχει φτάσει το σημείο αυτό.

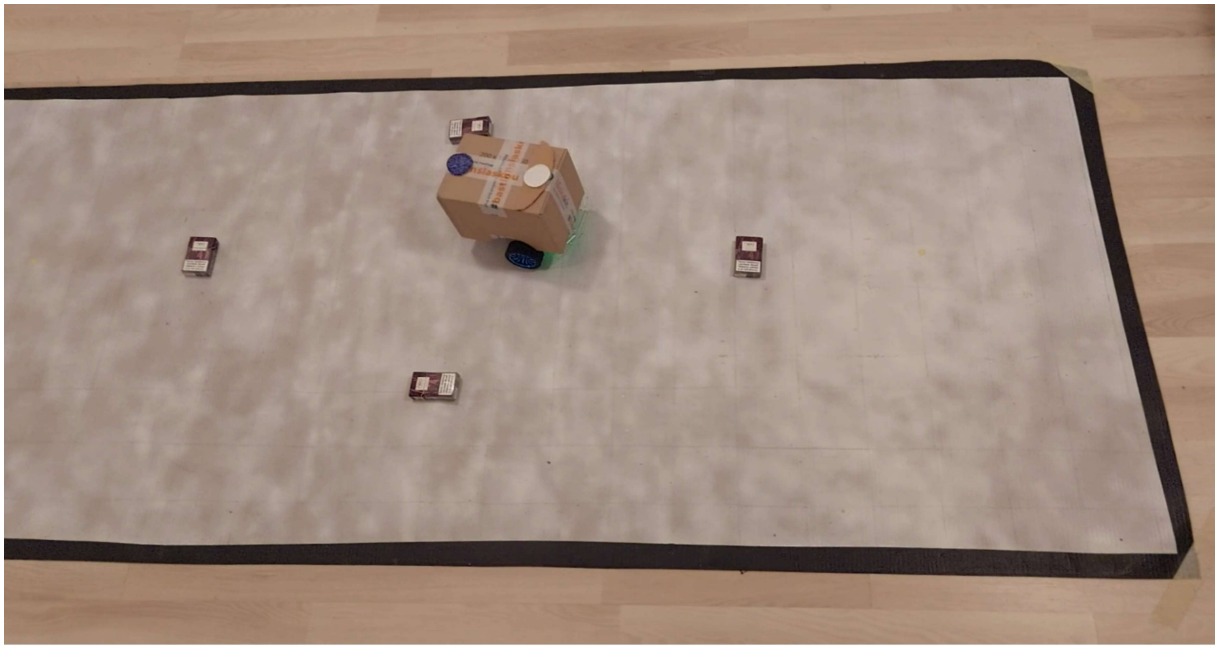


Εικόνα 33: Ρομπότ στο σημείο 8 , διαδρομή 1

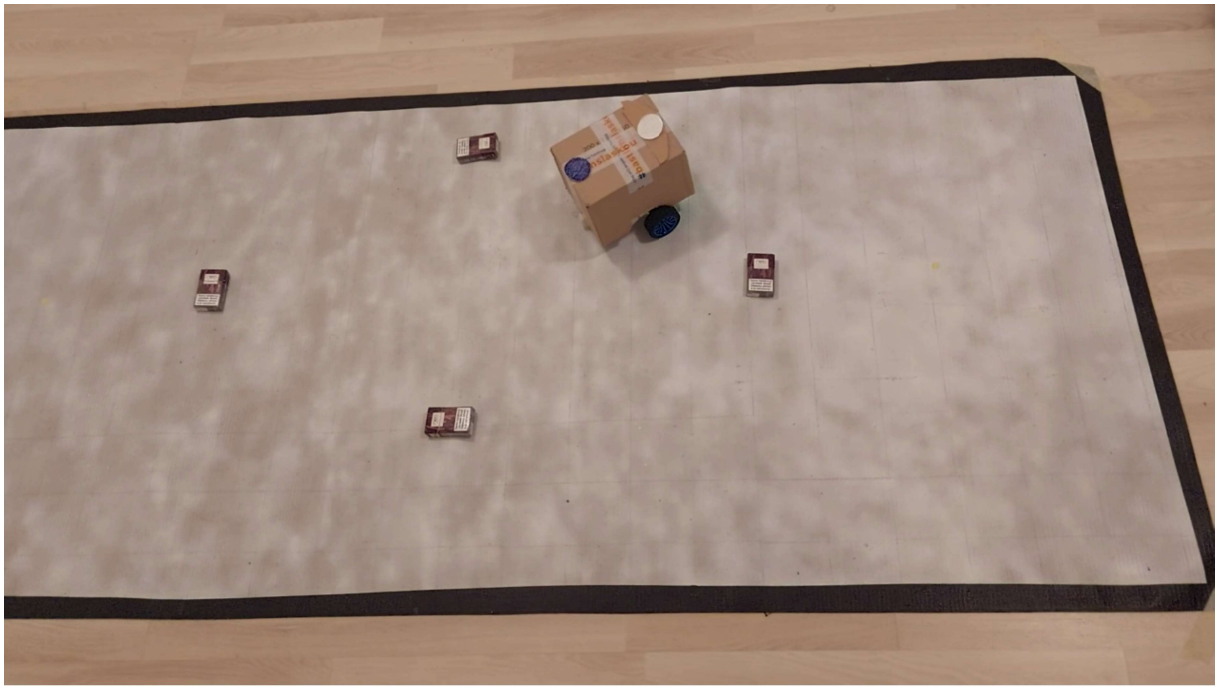


Εικόνα 34: Αλλαγή κατεύθυνσης ρομπότ για σημείο 9, διαδρομή 1

Στην Εικόνα 34 φαίνεται η αλλαγή της κατεύθυνσης που χρειάζεται να κάνει το ρομπότ για να φτάσει στο σημείο 9 και η Εικόνα 35 δείχνει το ρομπότ που έχει φτάσει το σημείο αυτό.

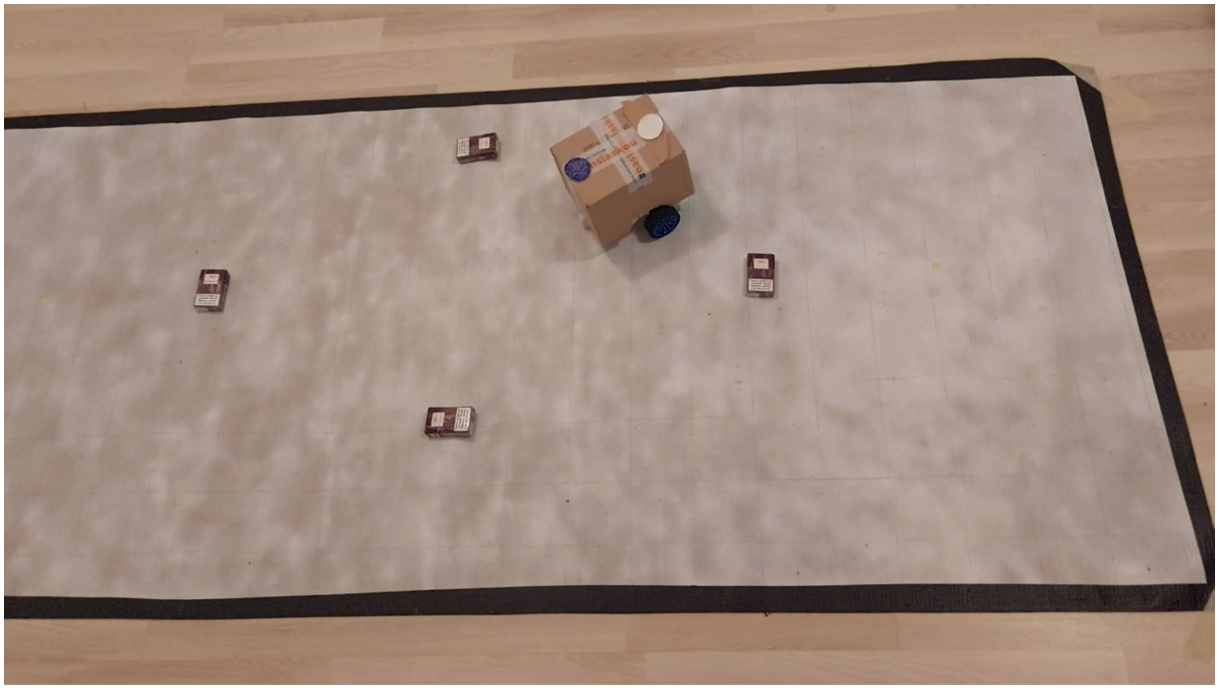


Εικόνα 35: Ρομπότ στο σημείο 9, διαδρομή 1

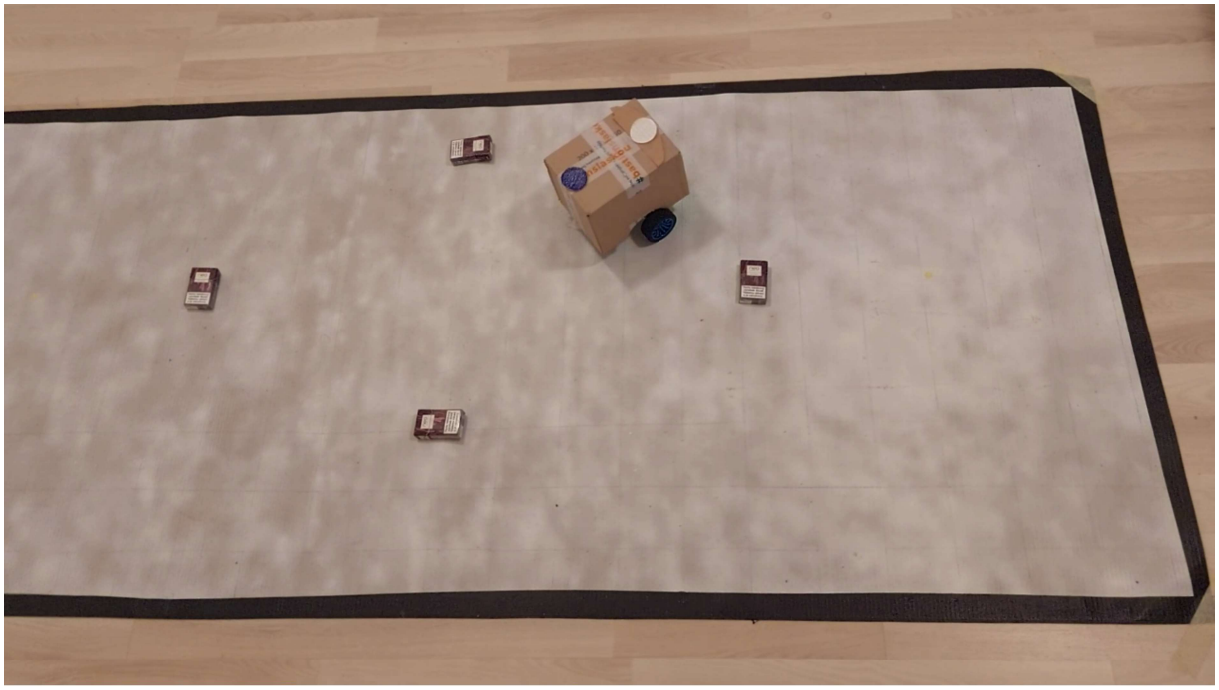


Εικόνα 36: Αλλαγή κατεύθυνσης ρομπότ για σημείο 10, διαδρομή 1

Στην Εικόνα 36 φαίνεται η αλλαγή της κατεύθυνσης που χρειάζεται να κάνει το ρομπότ για να φτάσει στο σημείο 10 και η Εικόνα 37 δείχνει το ρομπότ που έχει φτάσει το σημείο αυτό.

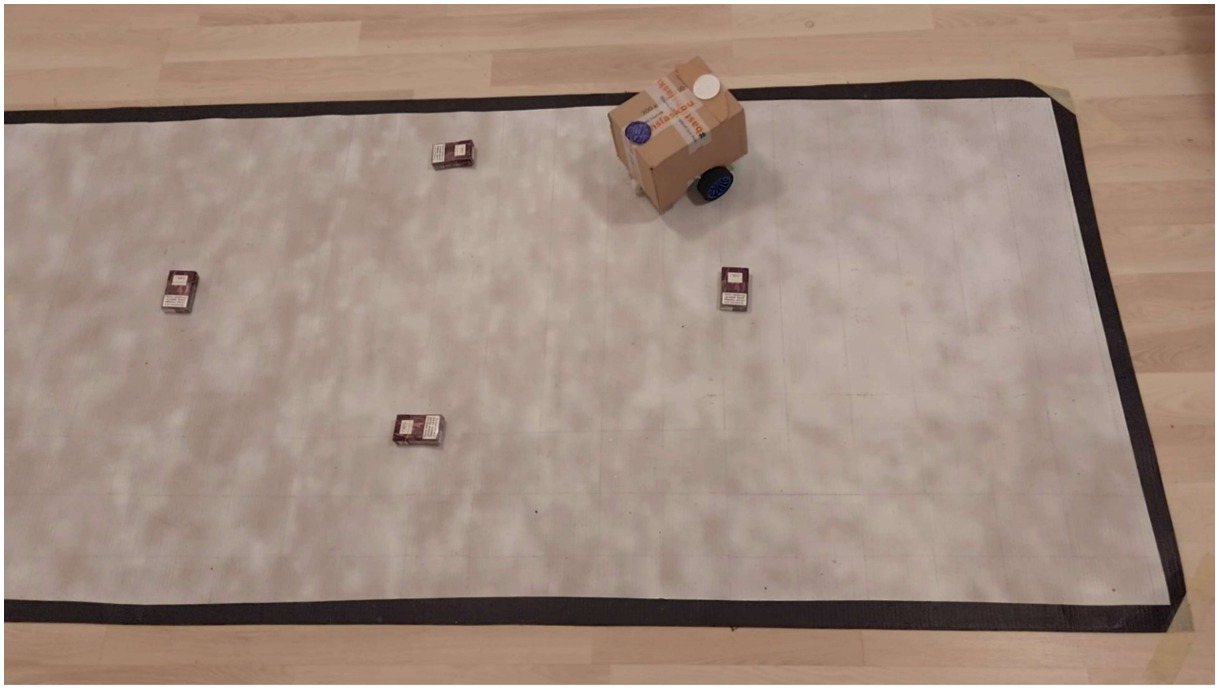


Εικόνα 37: Ρομπότ στο σημείο 10, διαδρομή 1

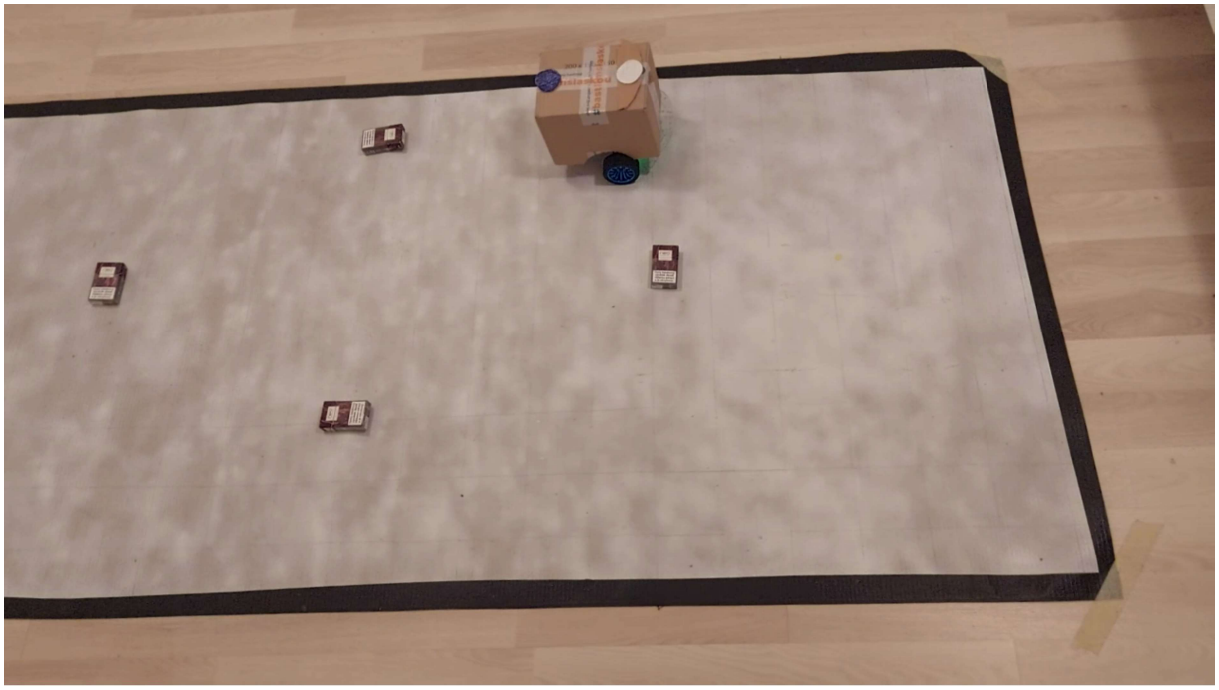


Εικόνα 38: Αλλαγή κατεύθυνσης ρομπότ για σημείο 11, διαδρομή 1

Στην Εικόνα 38 φαίνεται η αλλαγή της κατεύθυνσης που χρειάζεται να κάνει το ρομπότ για να φτάσει στο σημείο 11 και η Εικόνα 39 δείχνει το ρομπότ που έχει φτάσει το σημείο αυτό.

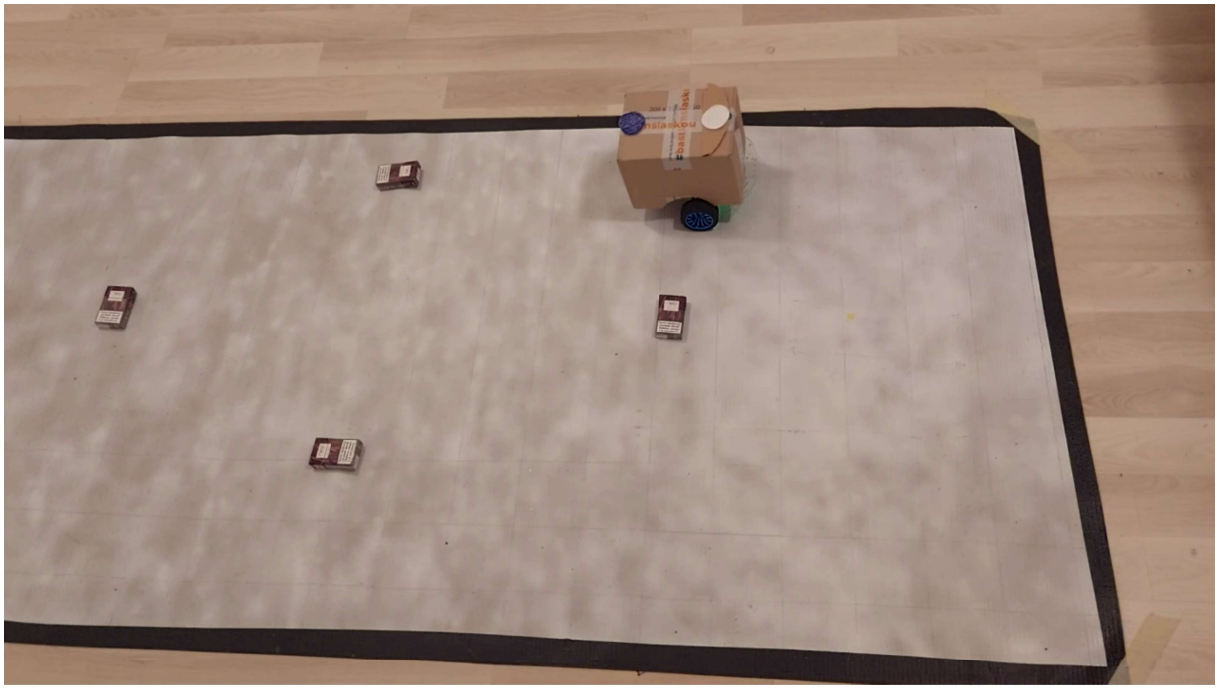


Εικόνα 39: Ρομπότ στο σημείο 11, διαδρομή 1

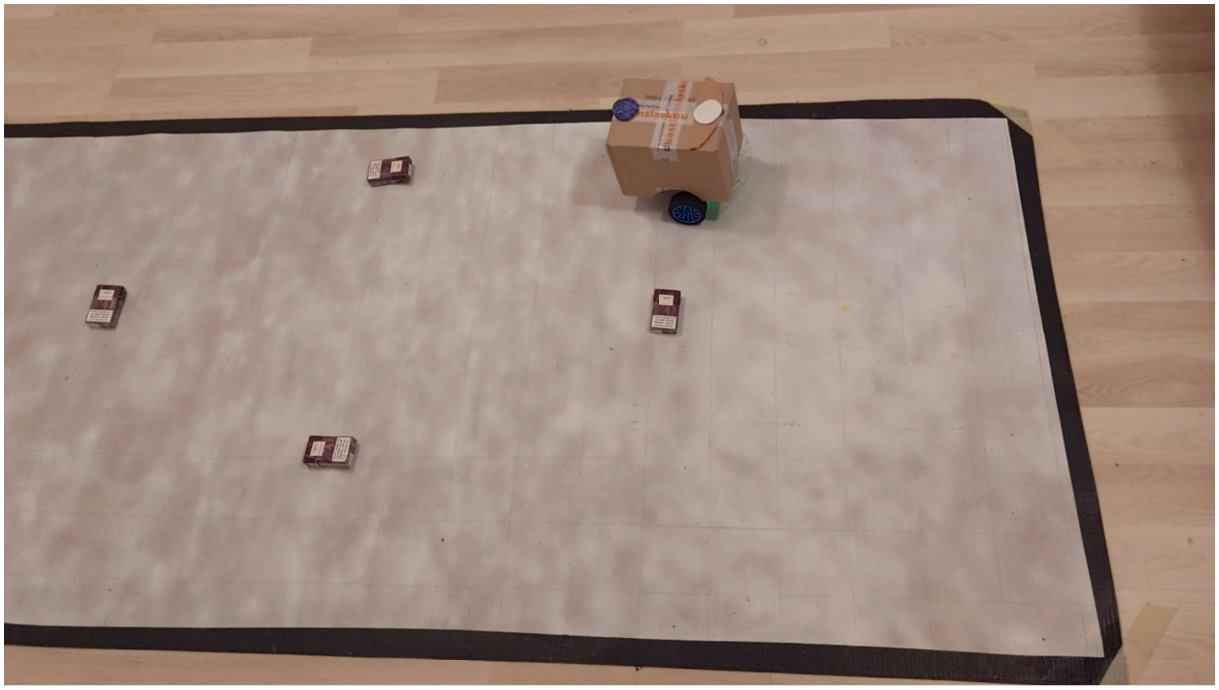


Εικόνα 40: Αλλαγή κατεύθυνσης ρομπότ για σημείο 12, διαδρομή 1

Στην Εικόνα 40 φαίνεται η αλλαγή της κατεύθυνσης που χρειάζεται να κάνει το ρομπότ για να φτάσει στο σημείο 12 και η Εικόνα 42 δείχνει το ρομπότ που έχει φτάσει το σημείο αυτό.

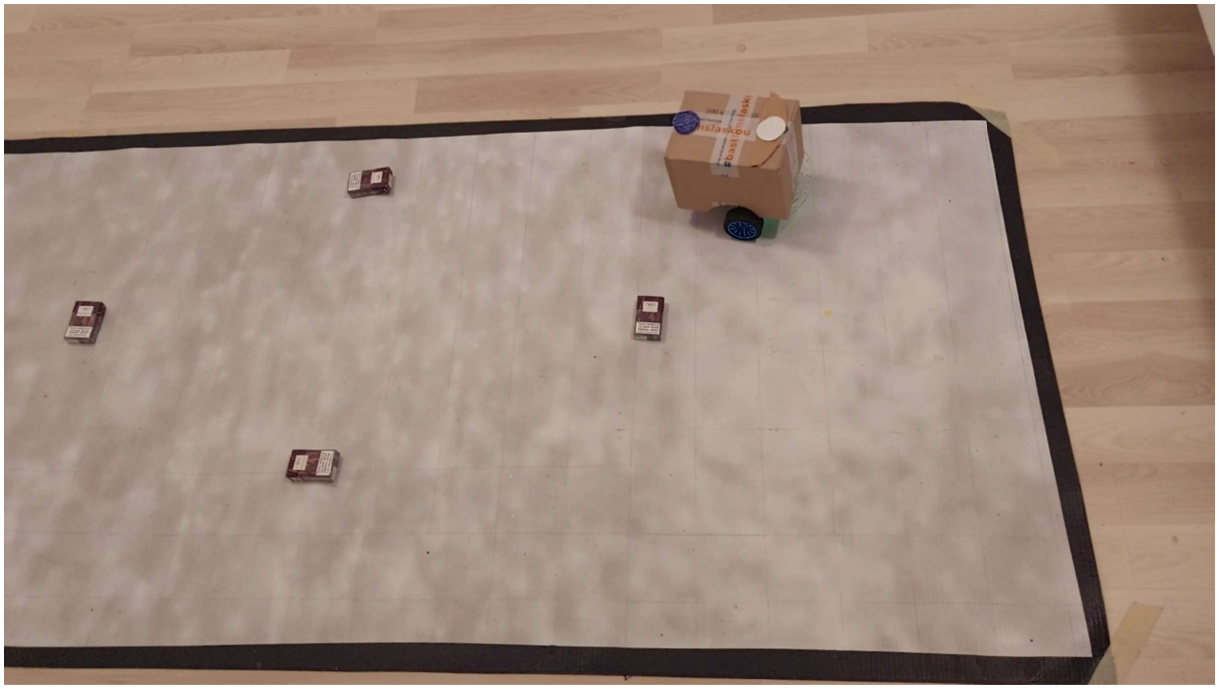


Εικόνα 41: Ρομπότ στο σημείο 12, διαδρομή 1

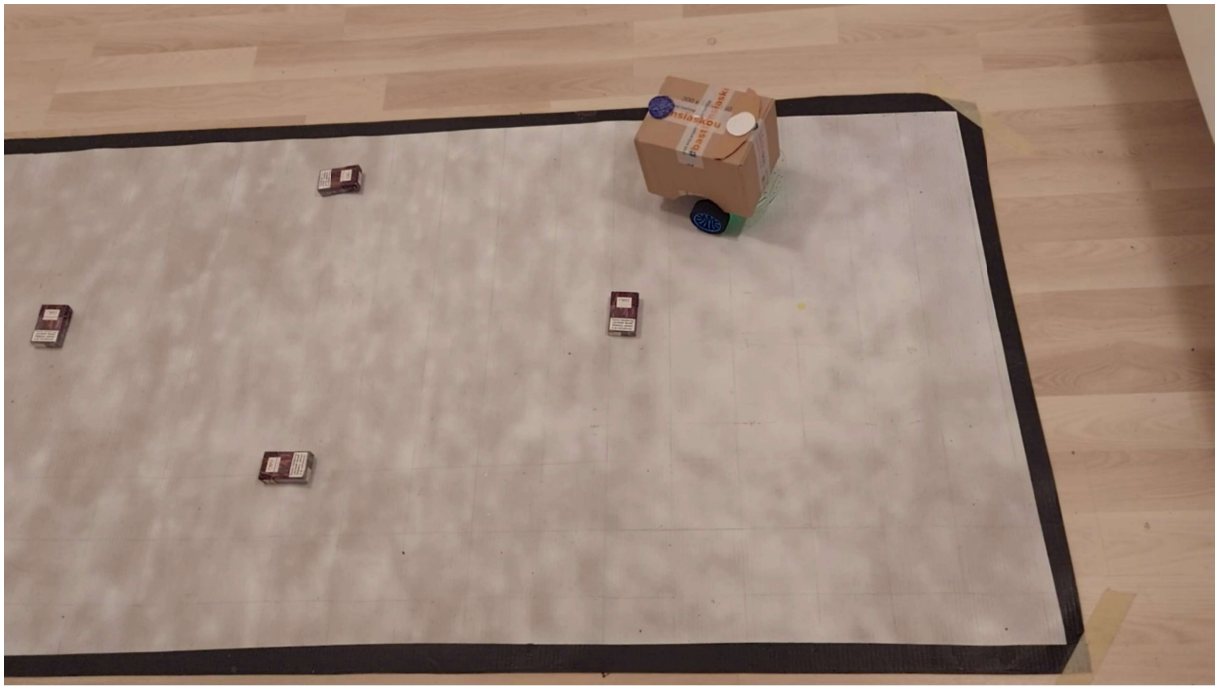


Εικόνα 42: Αλλαγή κατεύθυνσης ρομπότ για σημείο 13, διαδρομή 1

Στην Εικόνα 42 φαίνεται η αλλαγή της κατεύθυνσης που χρειάζεται να κάνει το ρομπότ για να φτάσει στο σημείο 13 και η Εικόνα 43 δείχνει το ρομπότ που έχει φτάσει το σημείο αυτό.

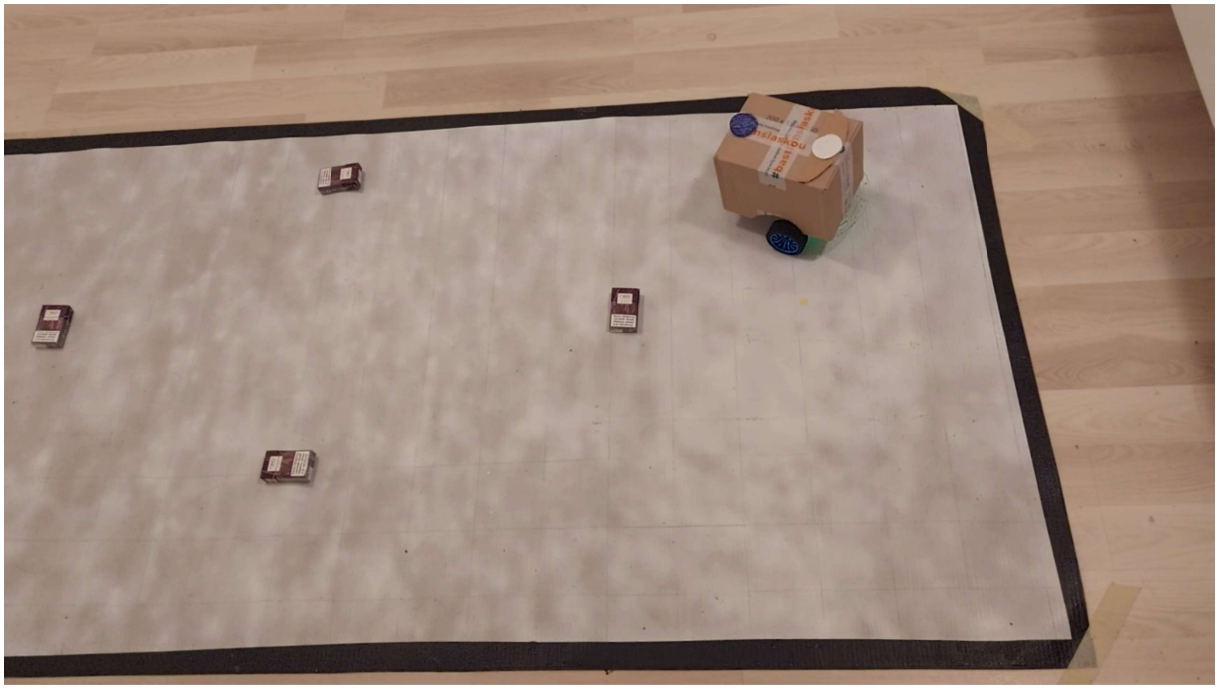


Εικόνα 43: Ρομπότ στο σημείο 13, διαδρομή 1

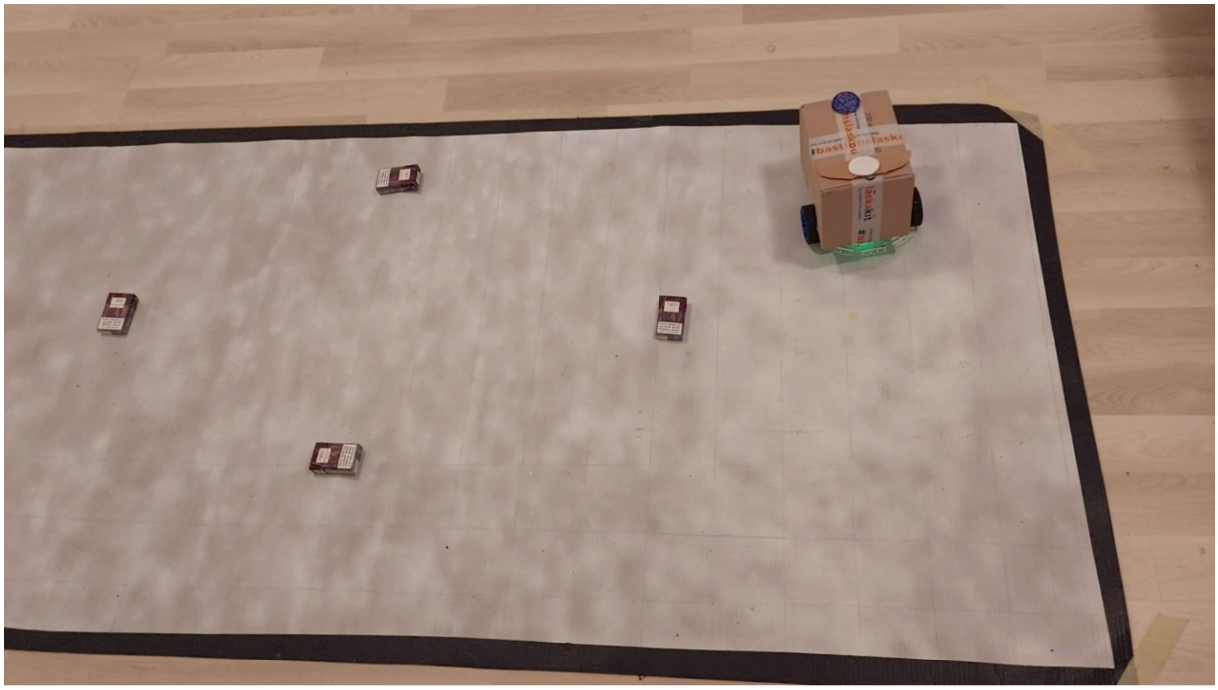


Εικόνα 44: Αλλαγή κατεύθυνσης ρομπότ για σημείο 14, διαδρομή 1

Στην Εικόνα 44 φαίνεται η αλλαγή της κατεύθυνσης που χρειάζεται να κάνει το ρομπότ για να φτάσει στο σημείο 14 και η Εικόνα 45 δείχνει το ρομπότ που έχει φτάσει το σημείο αυτό.

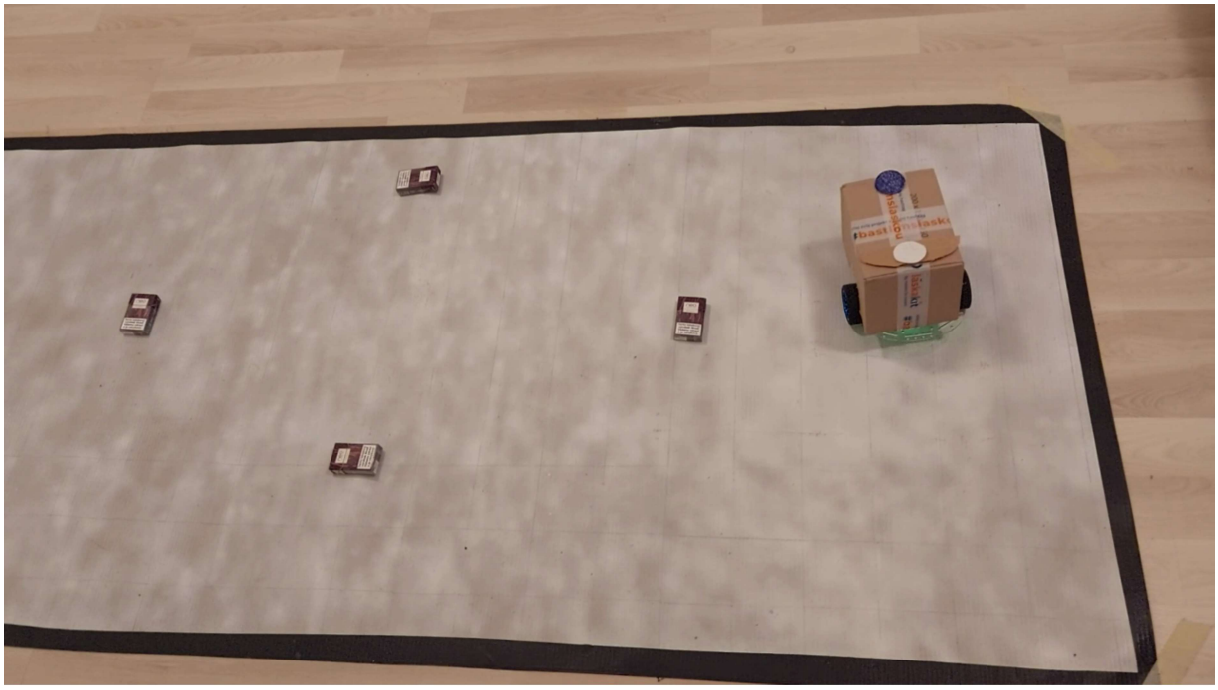


Εικόνα 45: Ρομπότ στο σημείο 14, διαδρομή 1

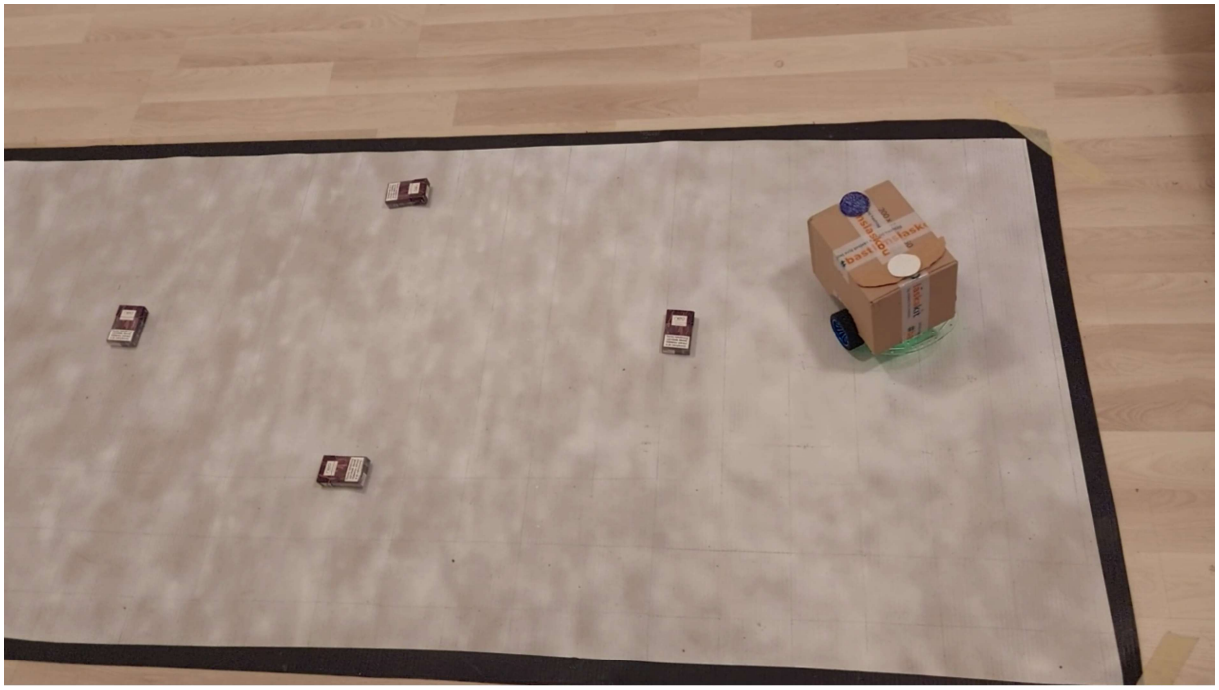


Εικόνα 46: Αλλαγή κατεύθυνσης ρομπότ για σημείο 14, διαδρομή 1

Στην Εικόνα 46 φαίνεται η αλλαγή της κατεύθυνσης που χρειάζεται να κάνει το ρομπότ για να φτάσει στο σημείο 15 και η Εικόνα 47 δείχνει το ρομπότ που έχει φτάσει το σημείο αυτό.

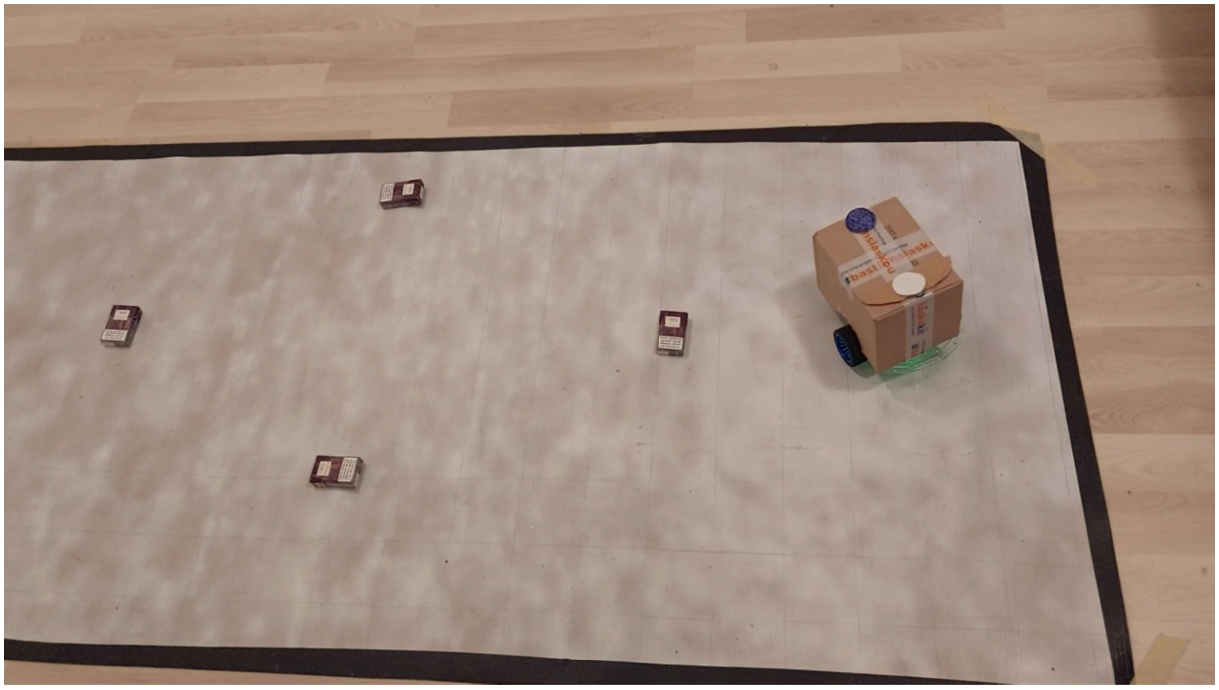


Εικόνα 47: Ρομπότ στο σημείο 15, διαδρομή 1



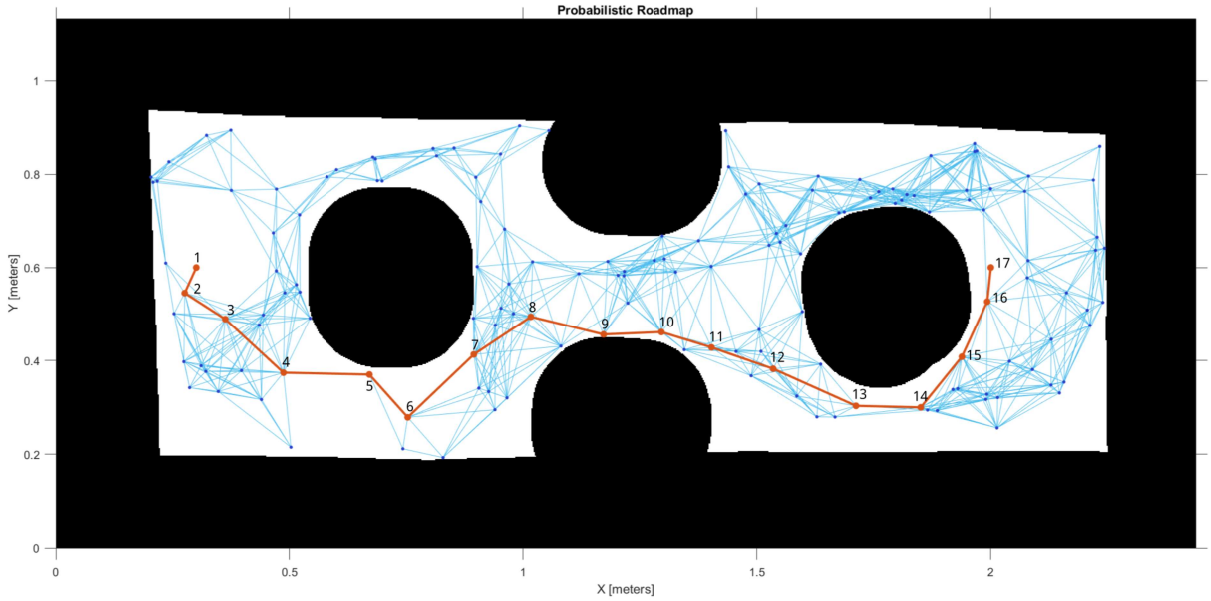
Εικόνα 48: Αλλαγή κατεύθυνσης ρομπότ για σημείο 16, διαδρομή 1

Στην Εικόνα 48 φαίνεται η αλλαγή της κατεύθυνσης που χρειάζεται να κάνει το ρομπότ για να φτάσει στο σημείο 16 και η Εικόνα 49 δείχνει το ρομπότ που έχει φτάσει το σημείο αυτό.



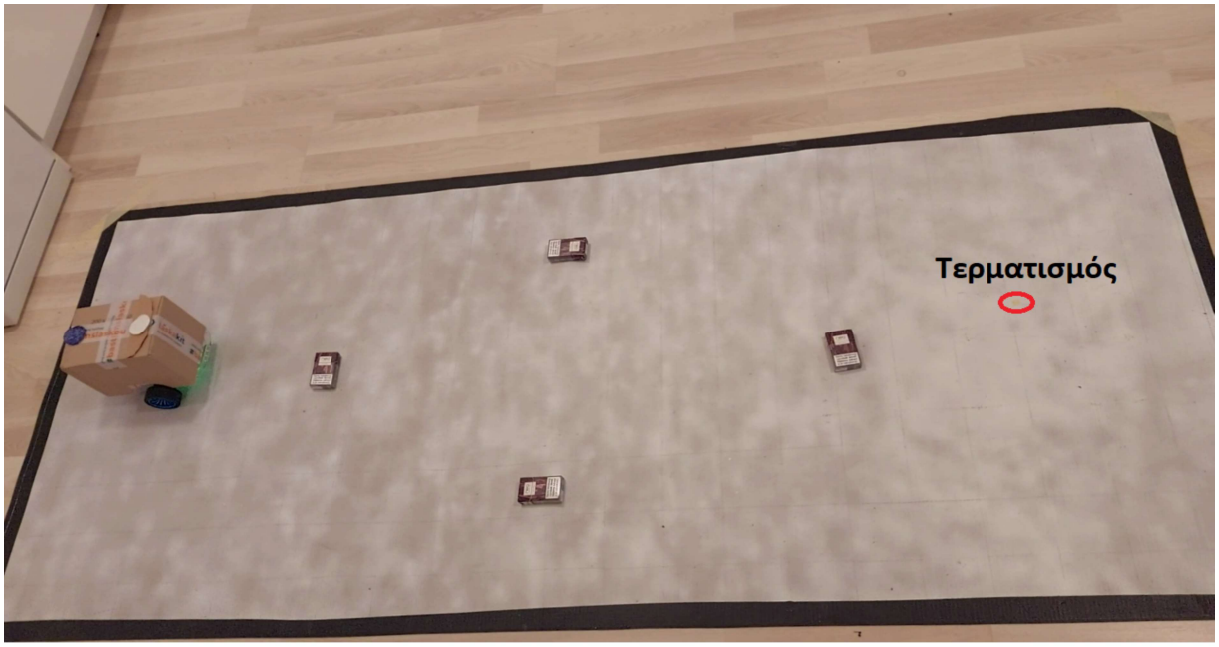
Εικόνα 49: Ρομπότ στο σημείο 16, διαδρομή 1

Η Εικόνα 49 δείχνει το ρομπότ στην τελική του θέση. Όπως φαίνεται από τις εικόνες το ρομπότ κατάφερε να ακολουθήσει την διαδρομή που υπολόγισε η μέθοδος PRM και να σταματήσει σε απόσταση μικρότερη των 10cm από τον τερματισμό. Για επιβεβαίωση των αποτελεσμάτων πραγματοποιήθηκε και δεύτερη δοκιμή όπου η διαδρομή που πρέπει να ακολουθήσει το ρομπότ φαίνεται στην Εικόνα 50.

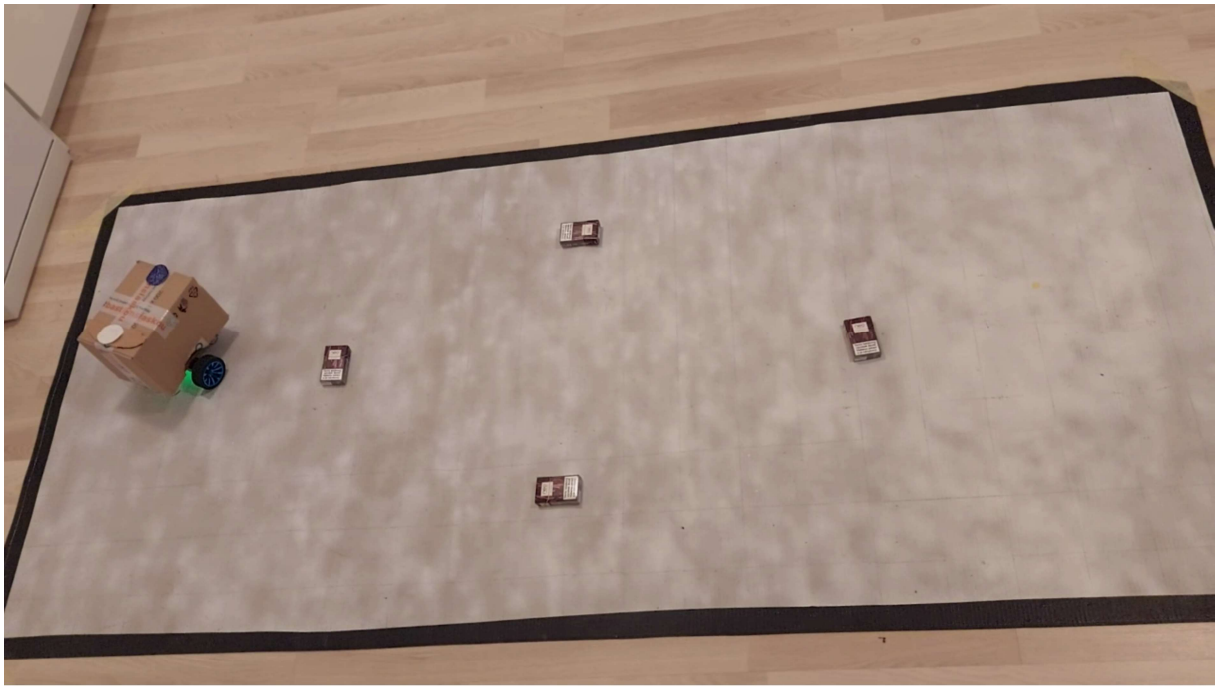


Εικόνα 50: Διαδρομή 2

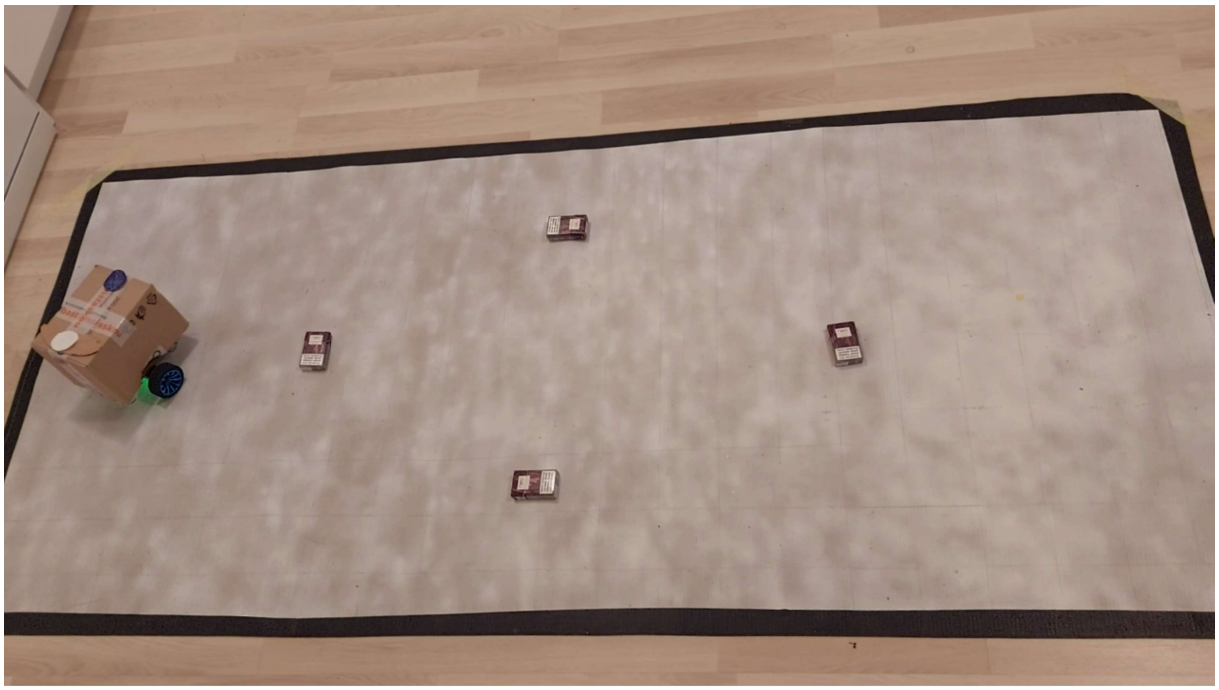
Στην Εικόνα 51 φαίνεται η αρχική τοποθέτηση του ρομπότ στον χώρο κίνησης καθώς και το σημείο του τερματισμού. Οι εικόνες 52 - 83 δείχνουν την διαδρομή που έκανε το ρομπότ.



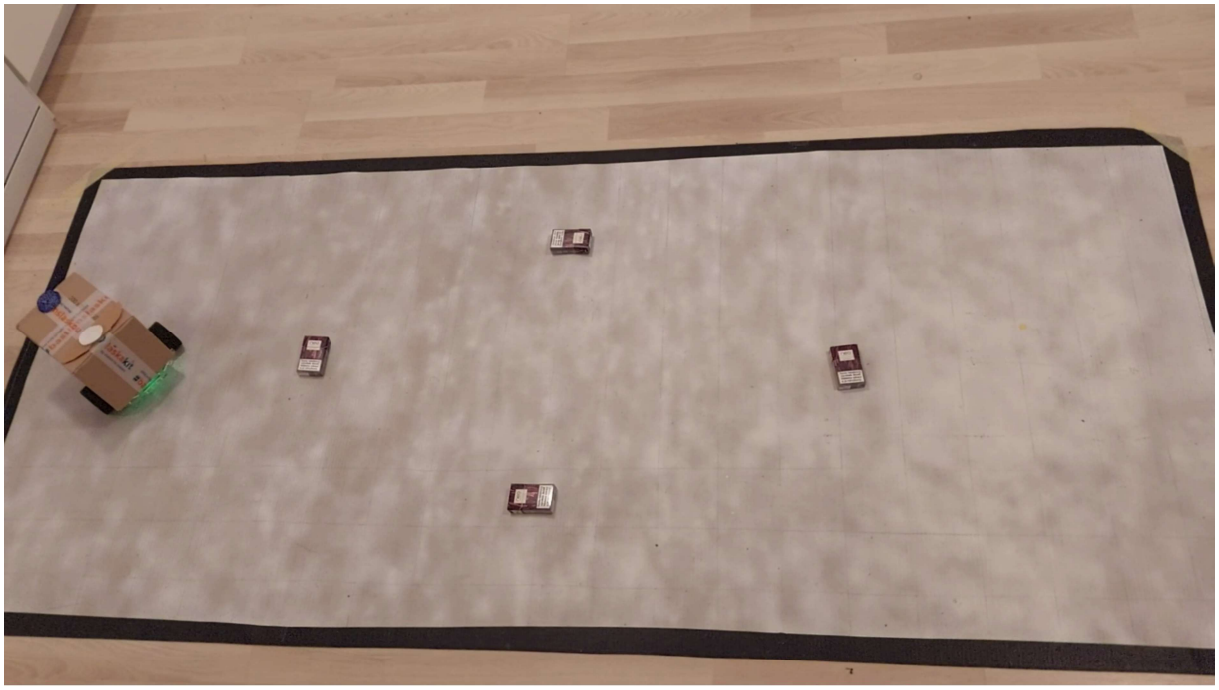
Εικόνα 51: Αρχική θέση ρομπότ, διαδρομή 2



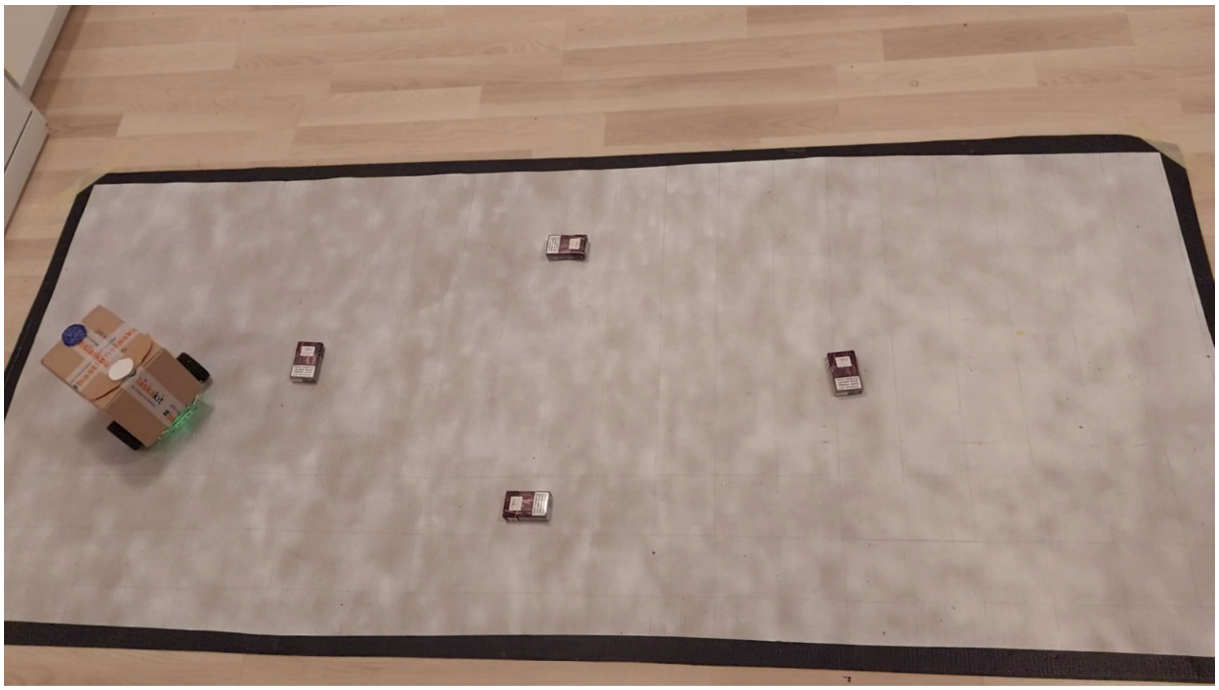
Εικόνα 52: Αλλαγή κατεύθυνσης ρομπότ για σημείο 2, διαδρομή 2



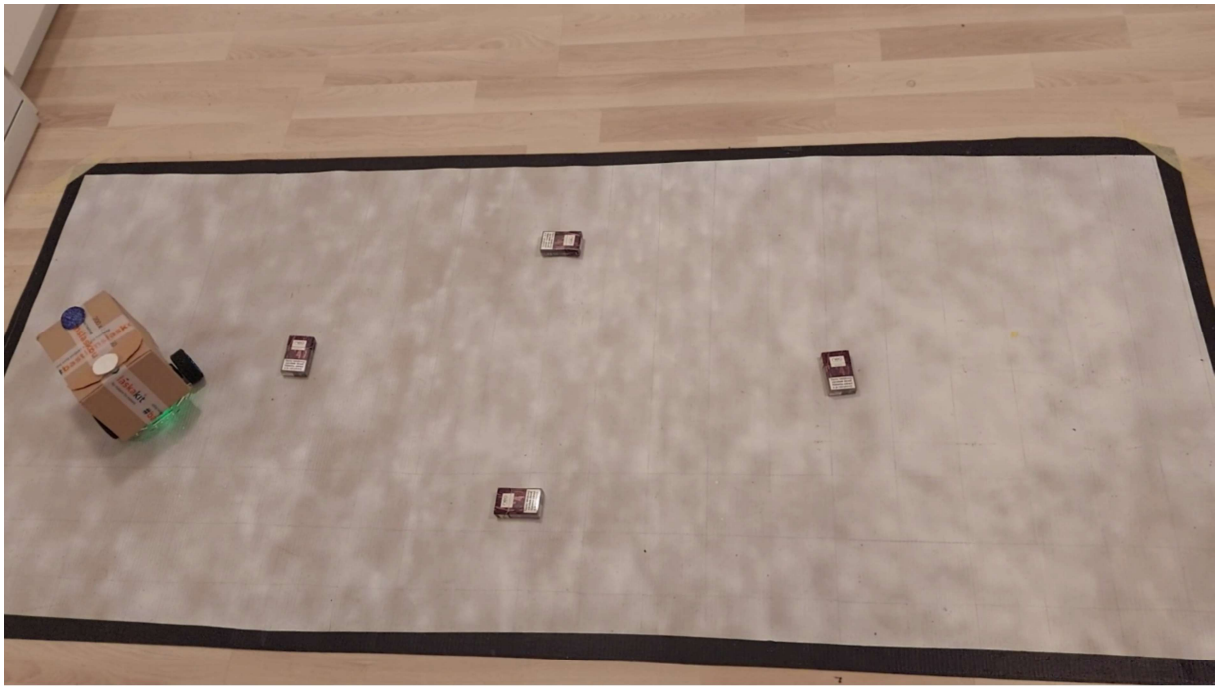
Εικόνα 53: Ρομπότ στο σημείο 2, διαδρομή 2



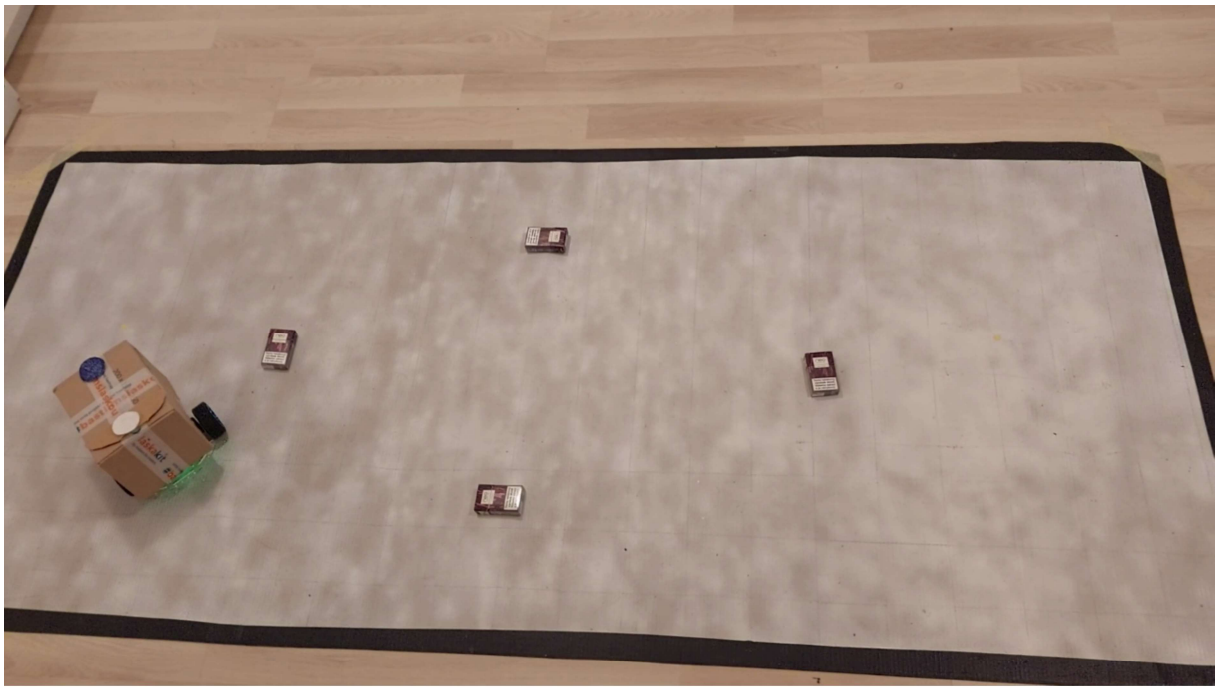
Εικόνα 54: Αλλαγή κατεύθυνσης ρομπότ για σημείο 3, διαδρομή 2



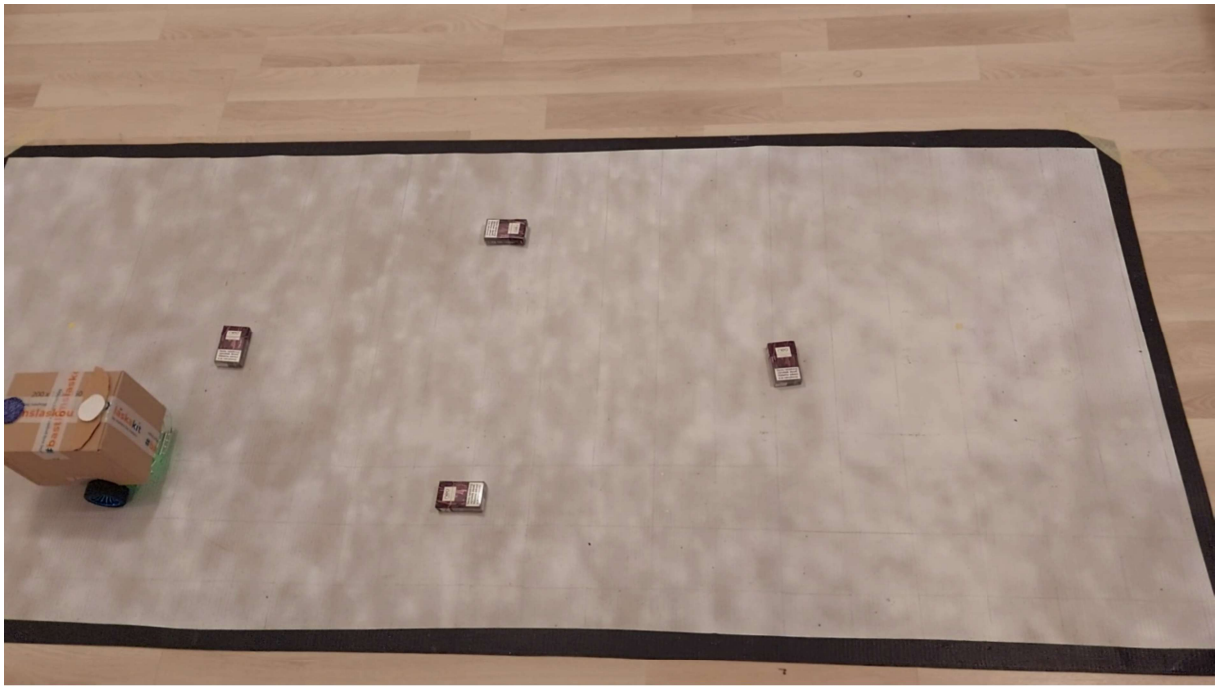
Εικόνα 55: Ρομπότ στο σημείο 3, διαδρομή 2



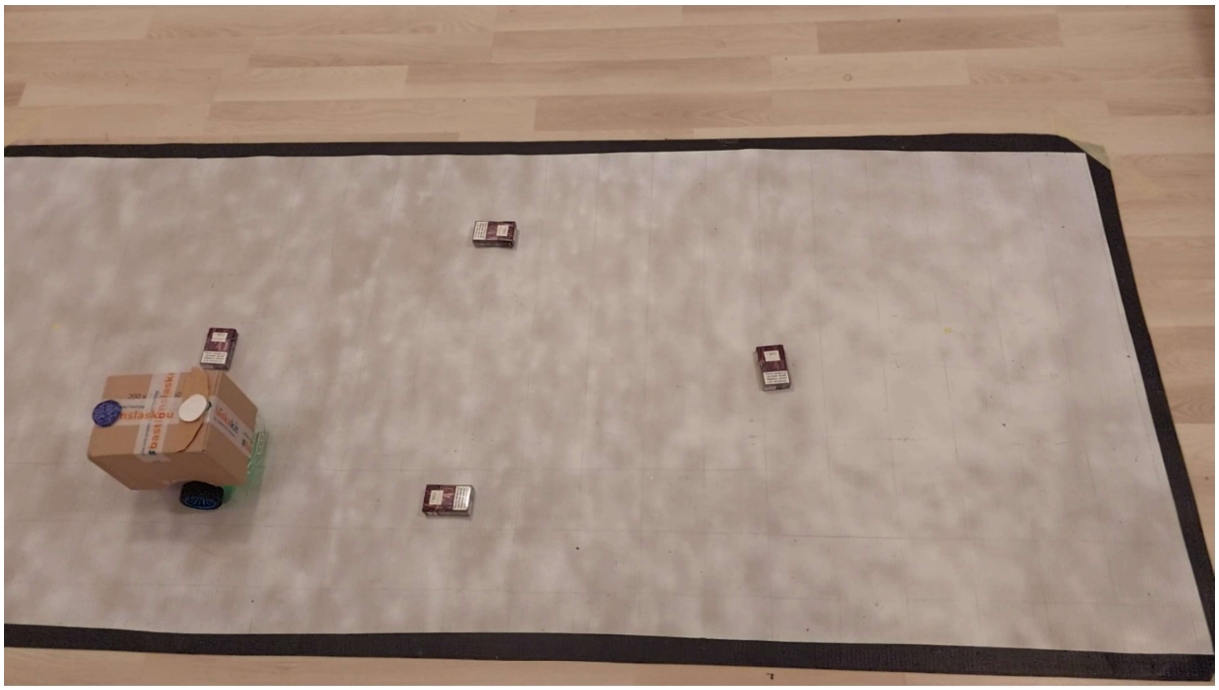
Εικόνα 56: Αλλαγή κατεύθυνσης ρομπότ για σημείο 4, διαδρομή 2



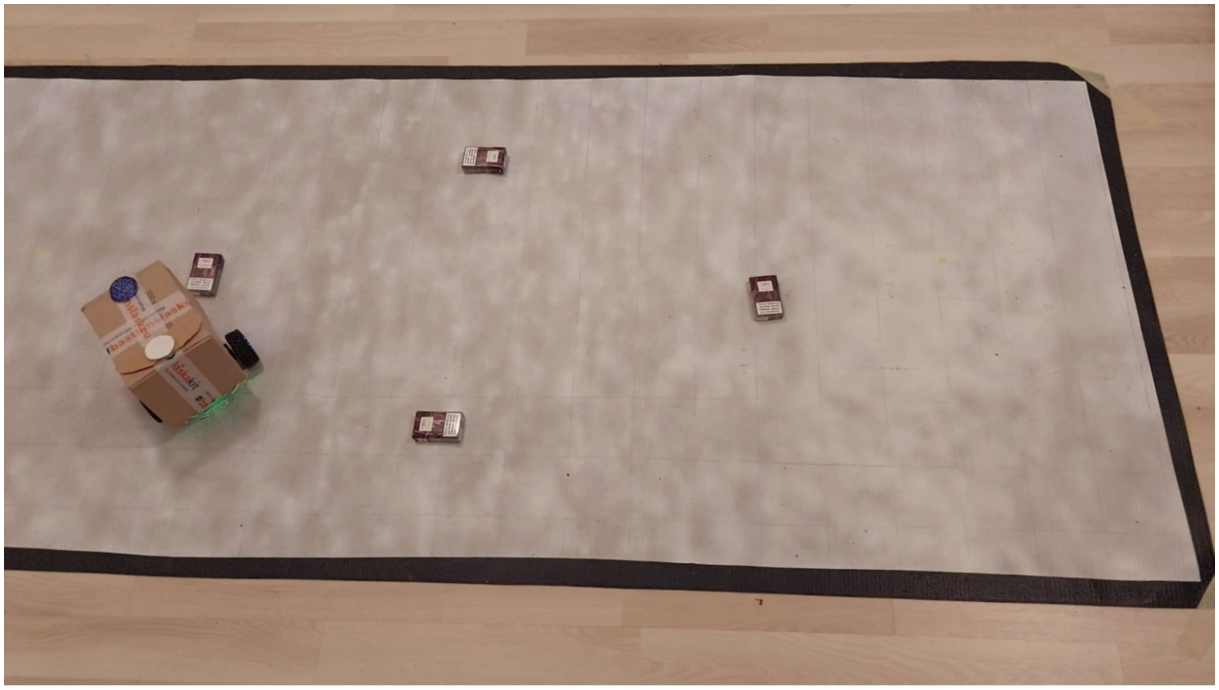
Εικόνα 57: Ρομπότ στο σημείο 4, διαδρομή 2



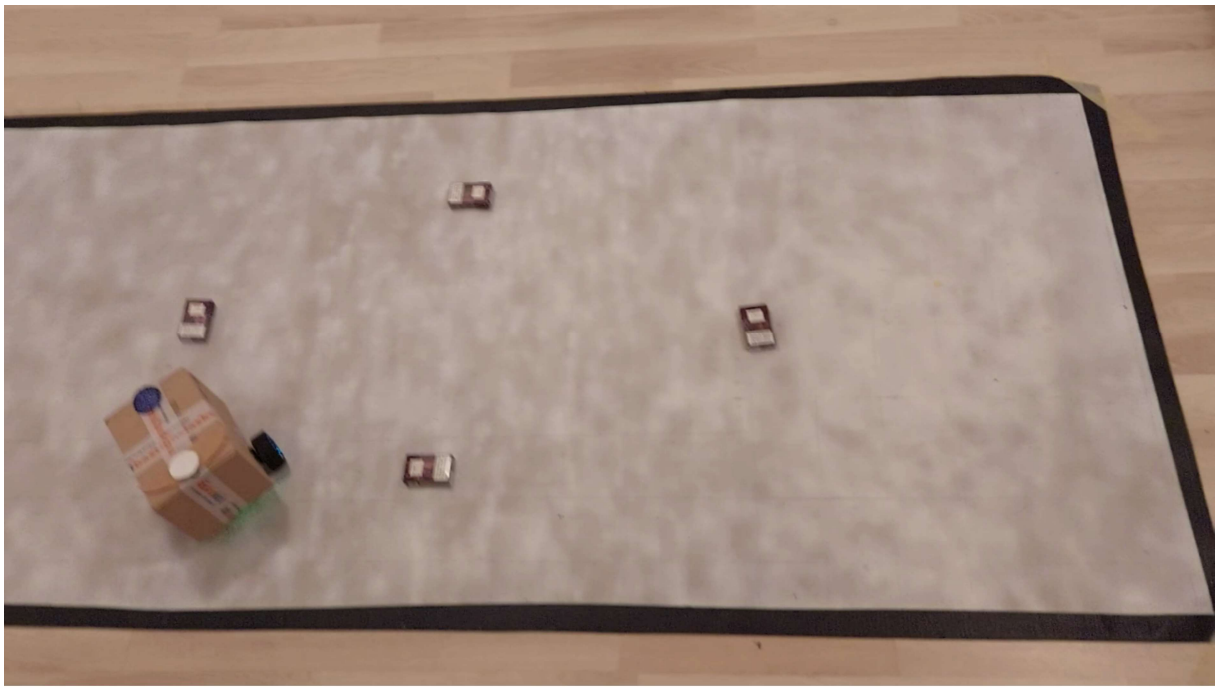
Εικόνα 58: Αλλαγή κατεύθυνσης ρομπότ για σημείο 5, διαδρομή 2



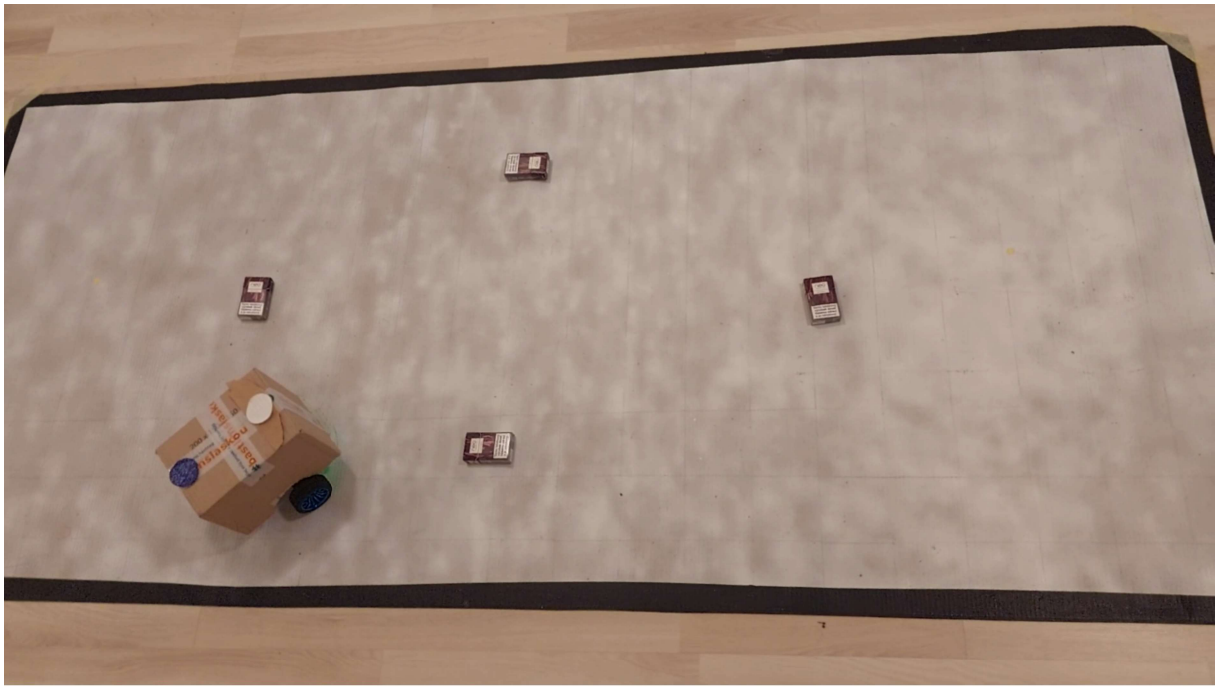
Εικόνα 59: Ρομπότ στο σημείο 5, διαδρομή 2



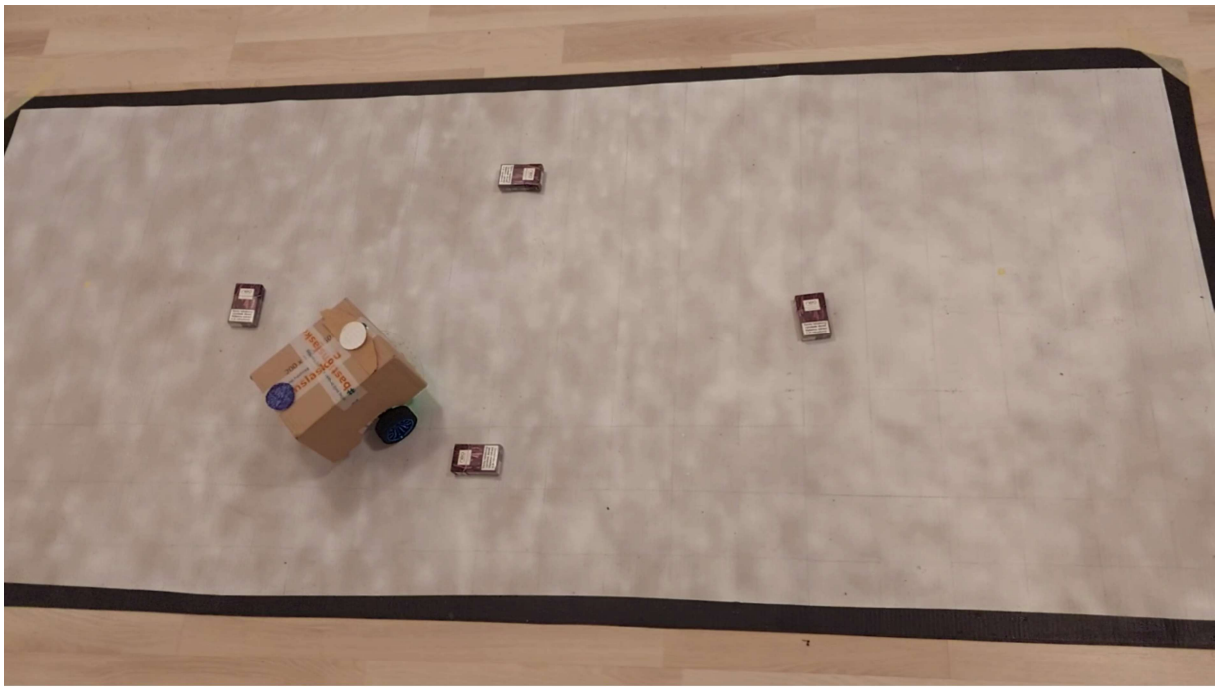
Εικόνα 60: Αλλαγή κατεύθυνσης ρομπότ για σημείο 6, διαδρομή 2



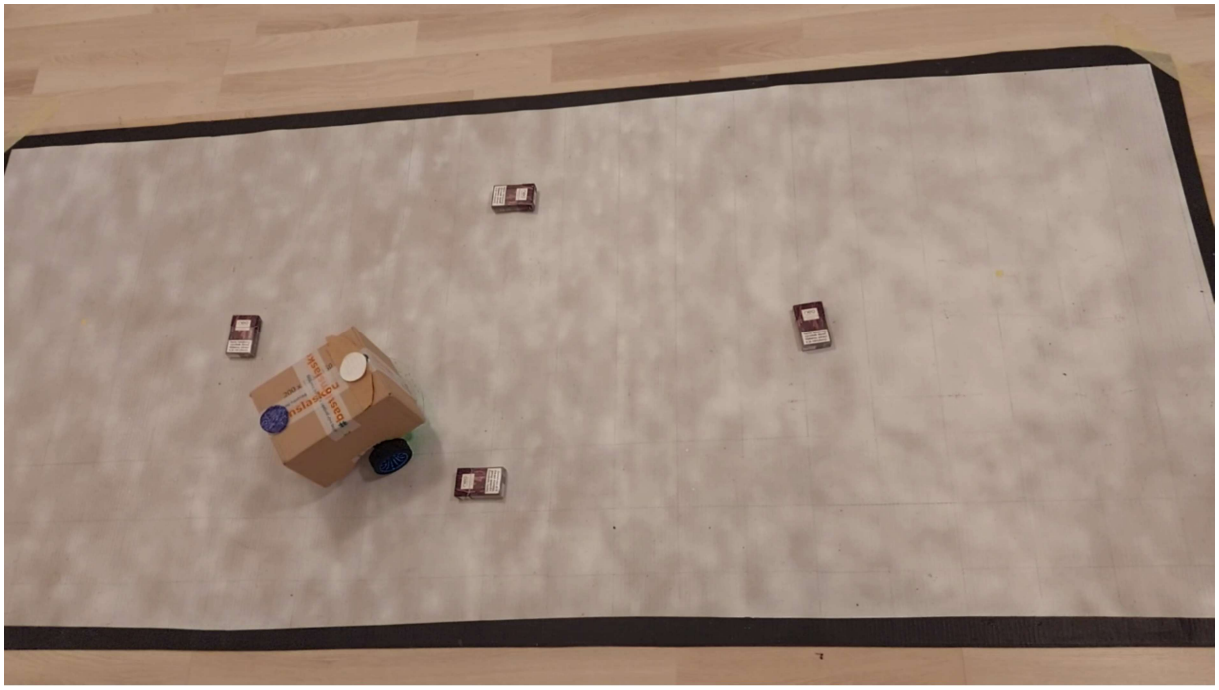
Εικόνα 61: Ρομπότ στο σημείο 6, διαδρομή 2



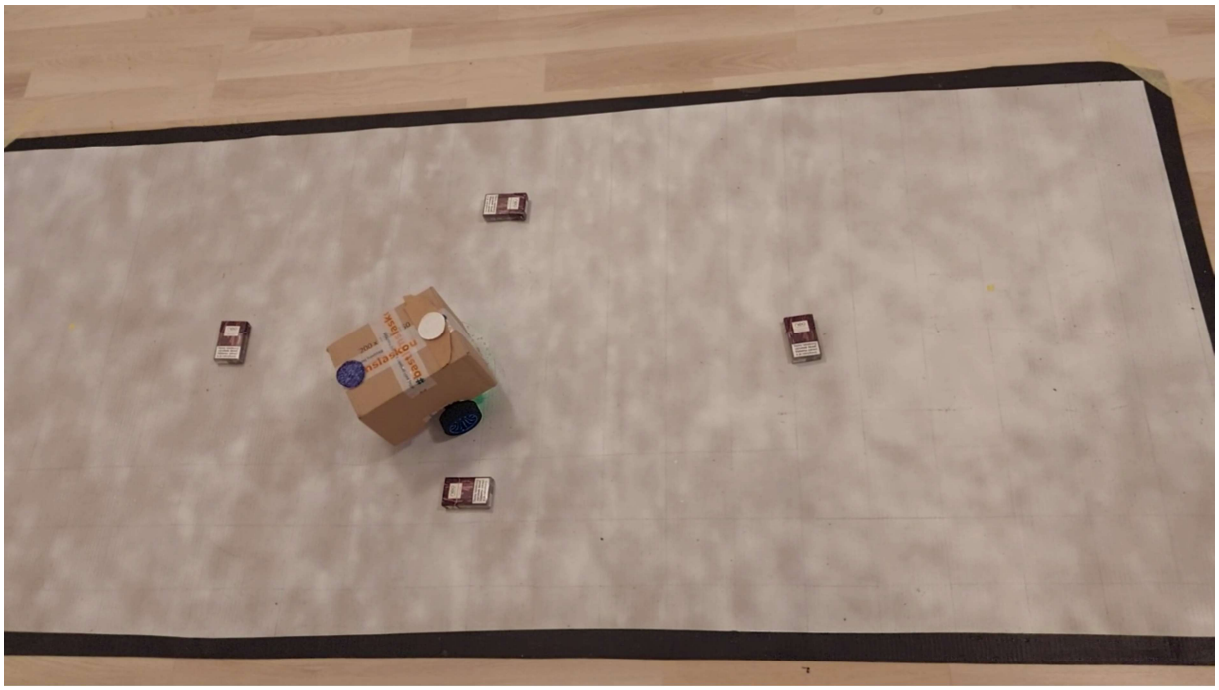
Εικόνα 62: Αλλαγή κατεύθυνσης ρομπότ για σημείο 7, διαδρομή 2



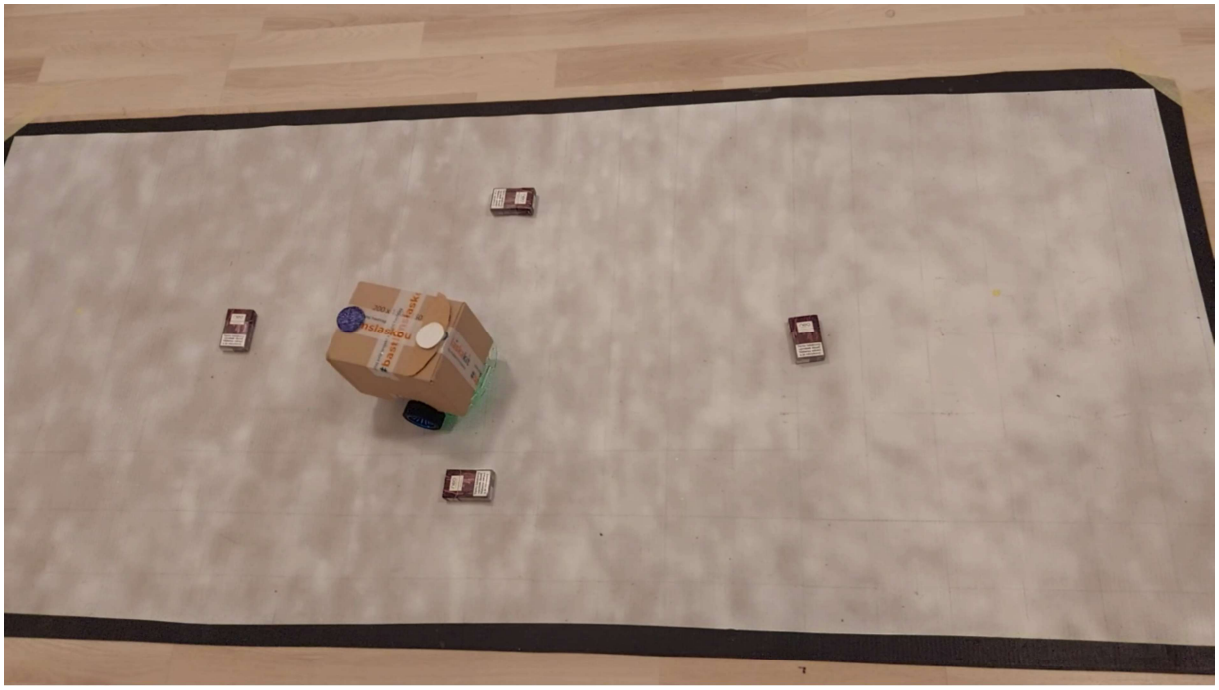
Εικόνα 63: Ρομπότ στο σημείο 7, διαδρομή 2



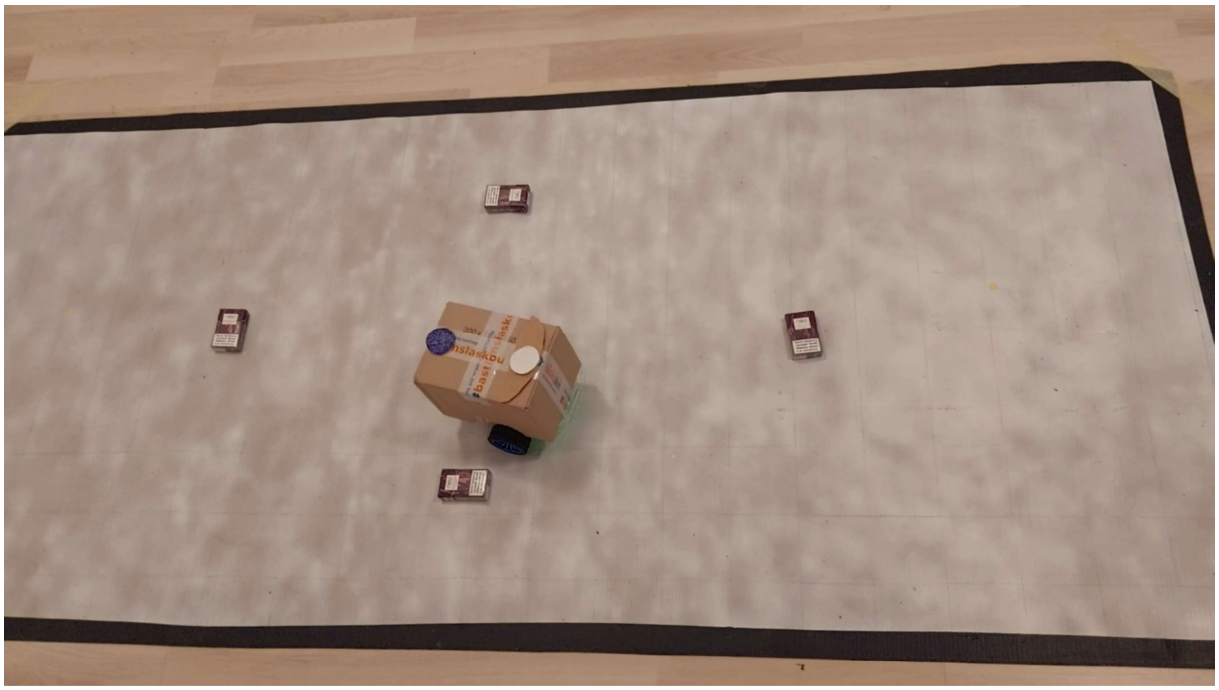
Εικόνα 64: Αλλαγή κατεύθυνσης ρομπότ για σημείο 8, διαδρομή 2



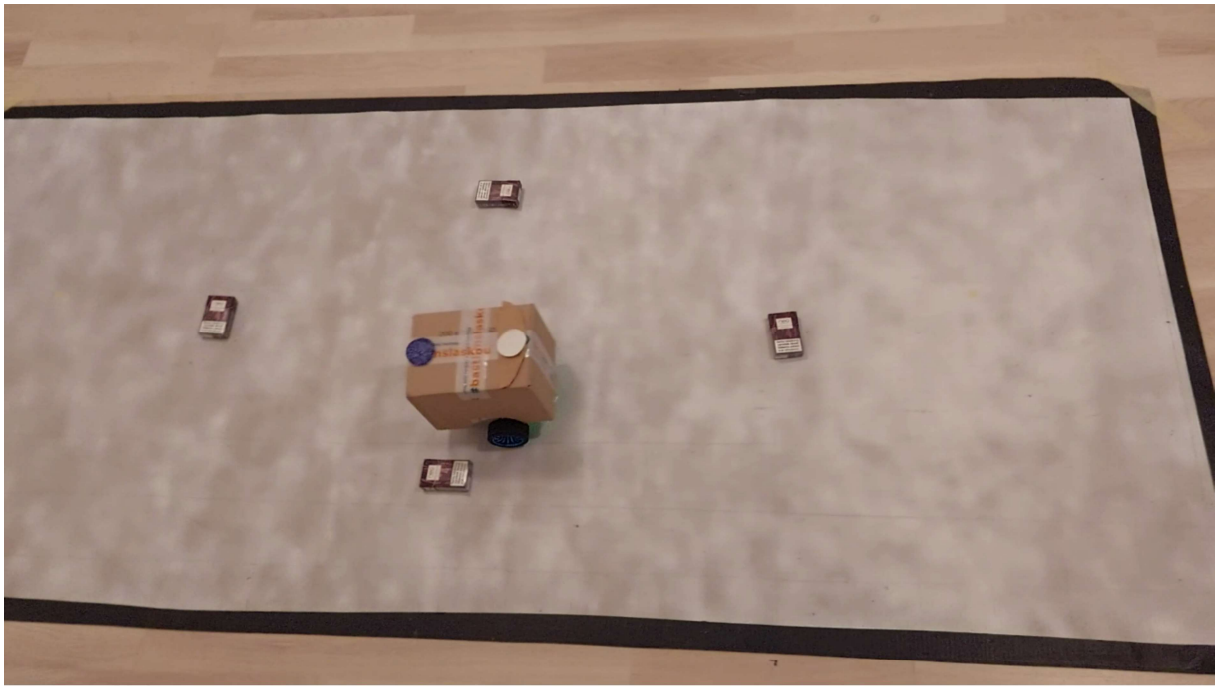
Εικόνα 65: Ρομπότ στο σημείο 8, διαδρομή 2



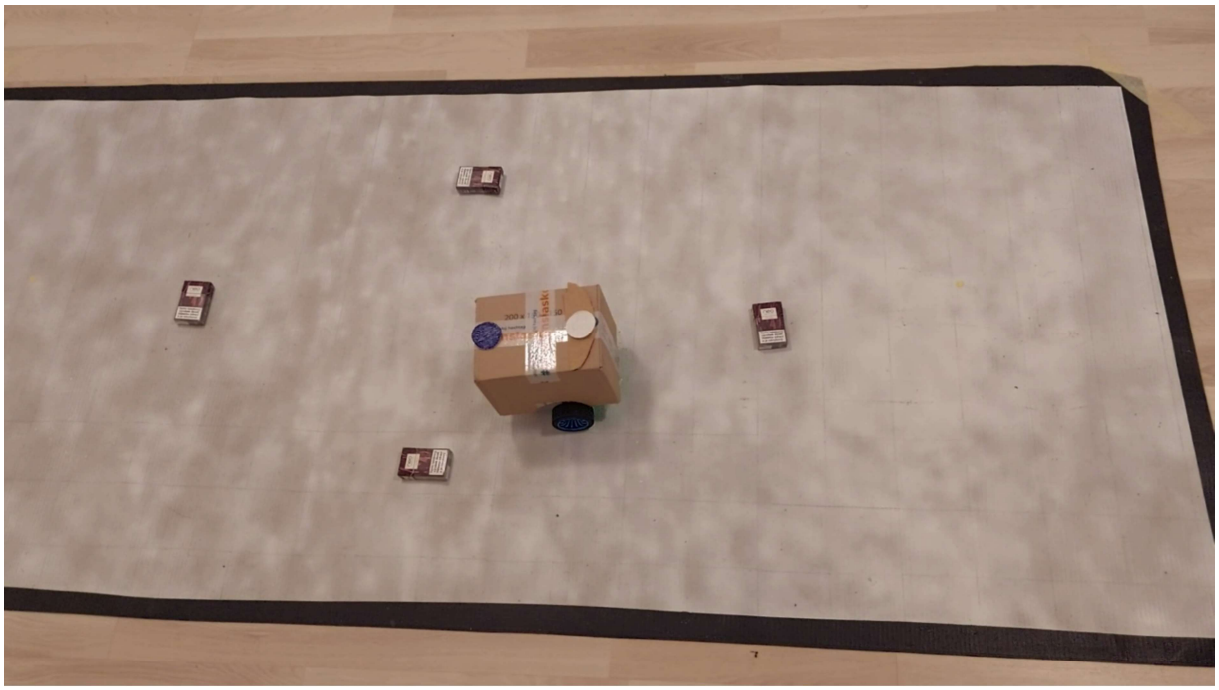
Εικόνα 66: Αλλαγή κατεύθυνσης ρομπότ για σημείο 9, διαδρομή 2



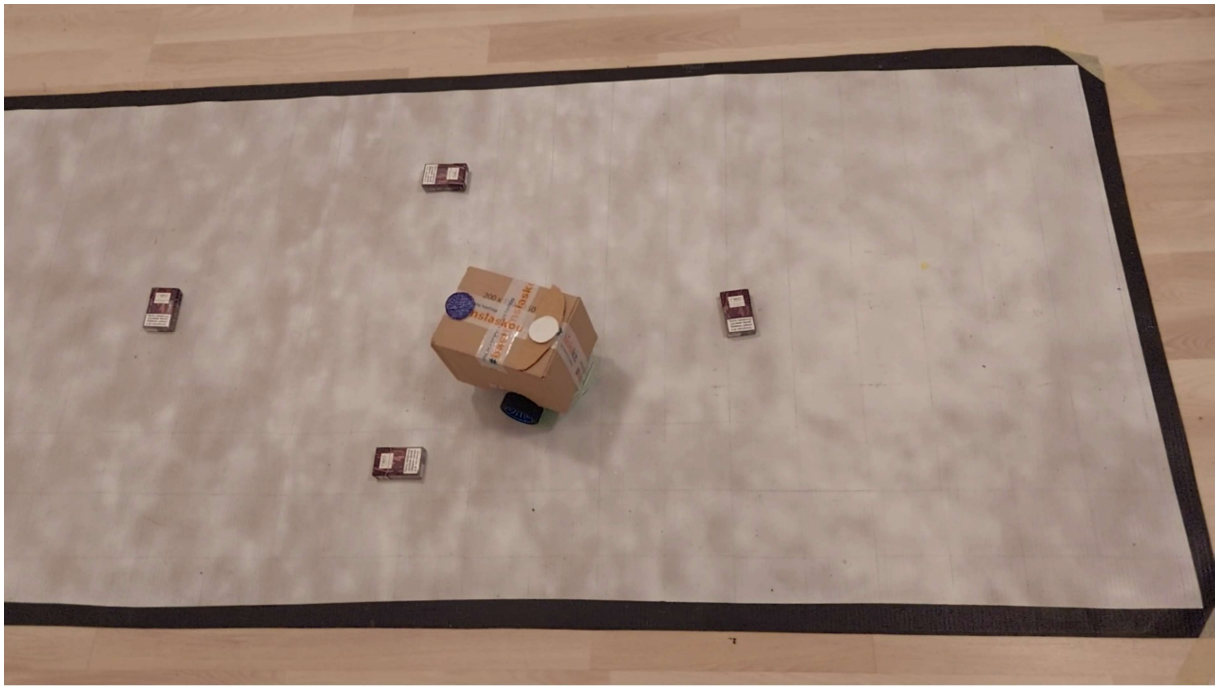
Εικόνα 67: Ρομπότ στο σημείο 9, διαδρομή 2



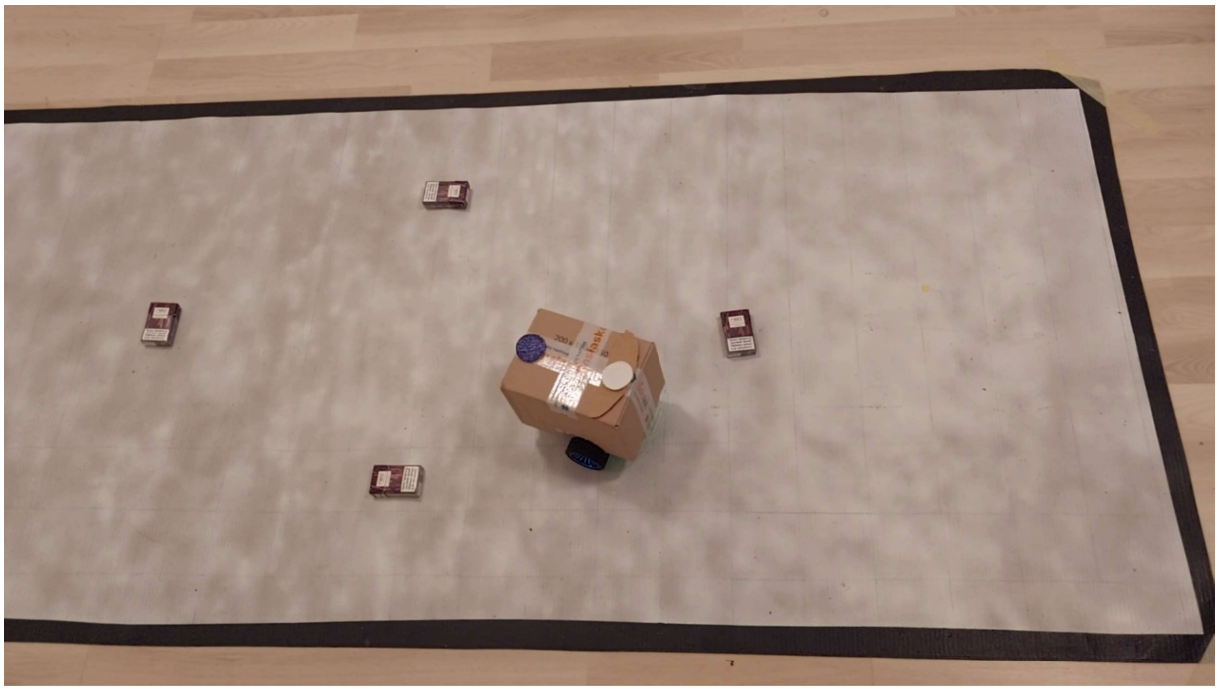
Εικόνα 68: Αλλαγή κατεύθυνσης ρομπότ για σημείο 10, διαδρομή 2



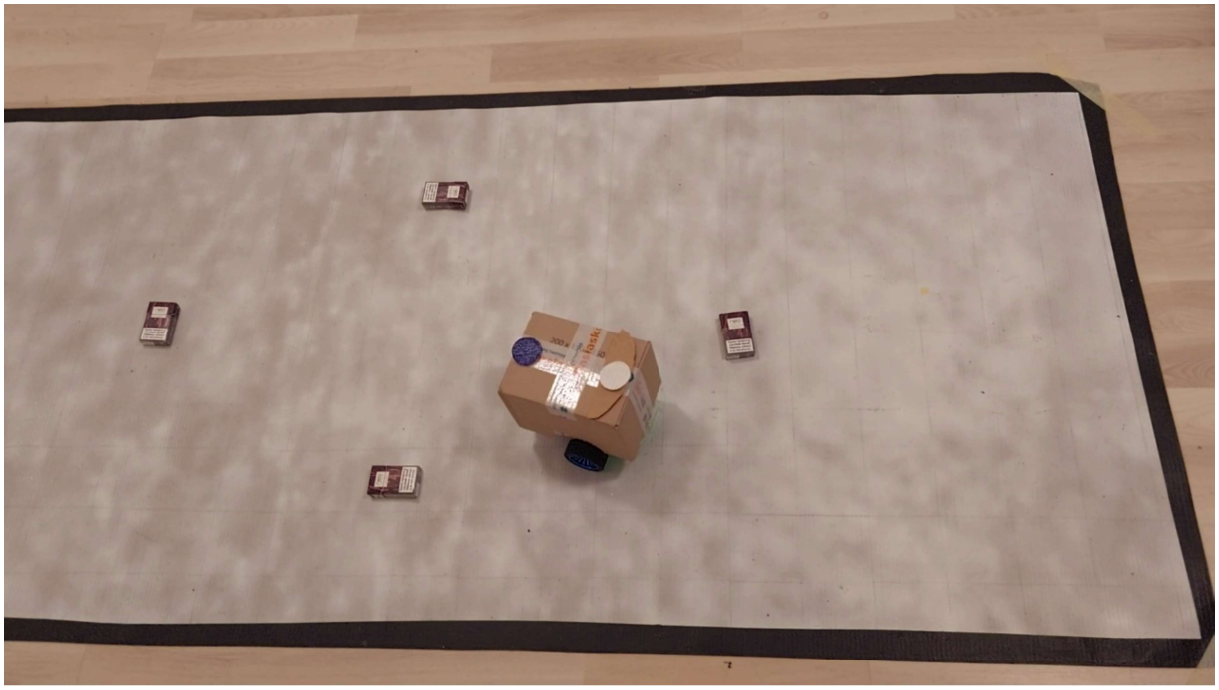
Εικόνα 69: Ρομπότ στο σημείο 10, διαδρομή 2



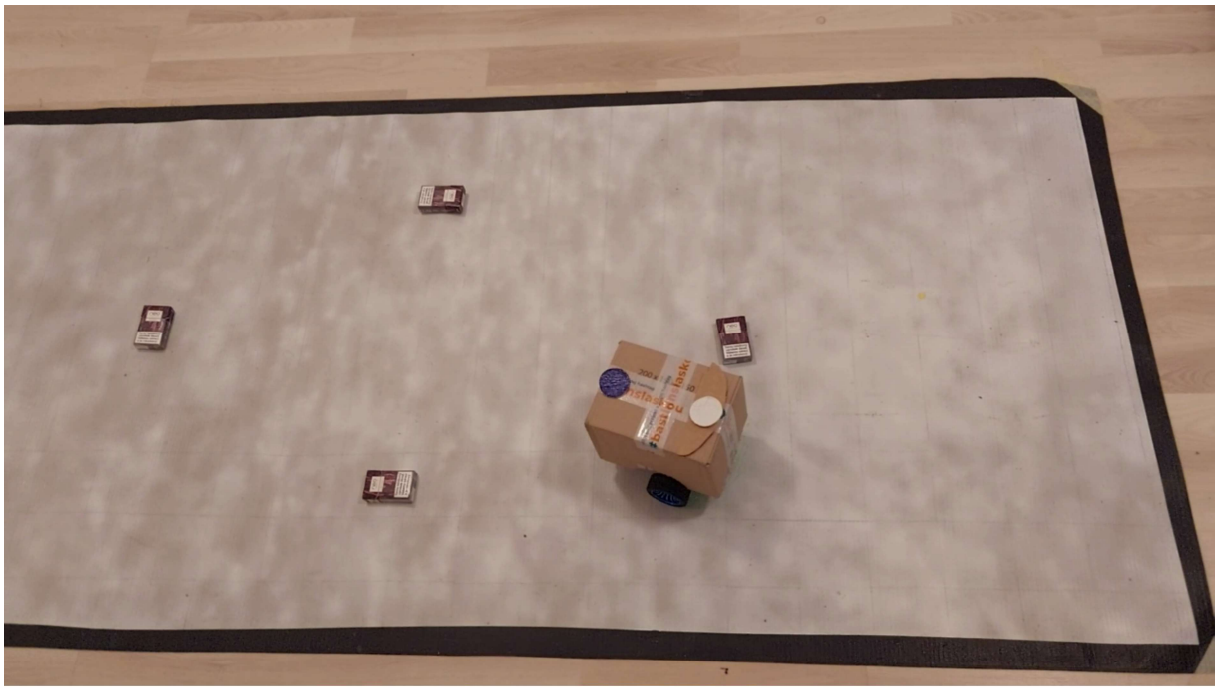
Εικόνα 70: Αλλαγή κατεύθυνσης ρομπότ για σημείο 11, διαδρομή 2



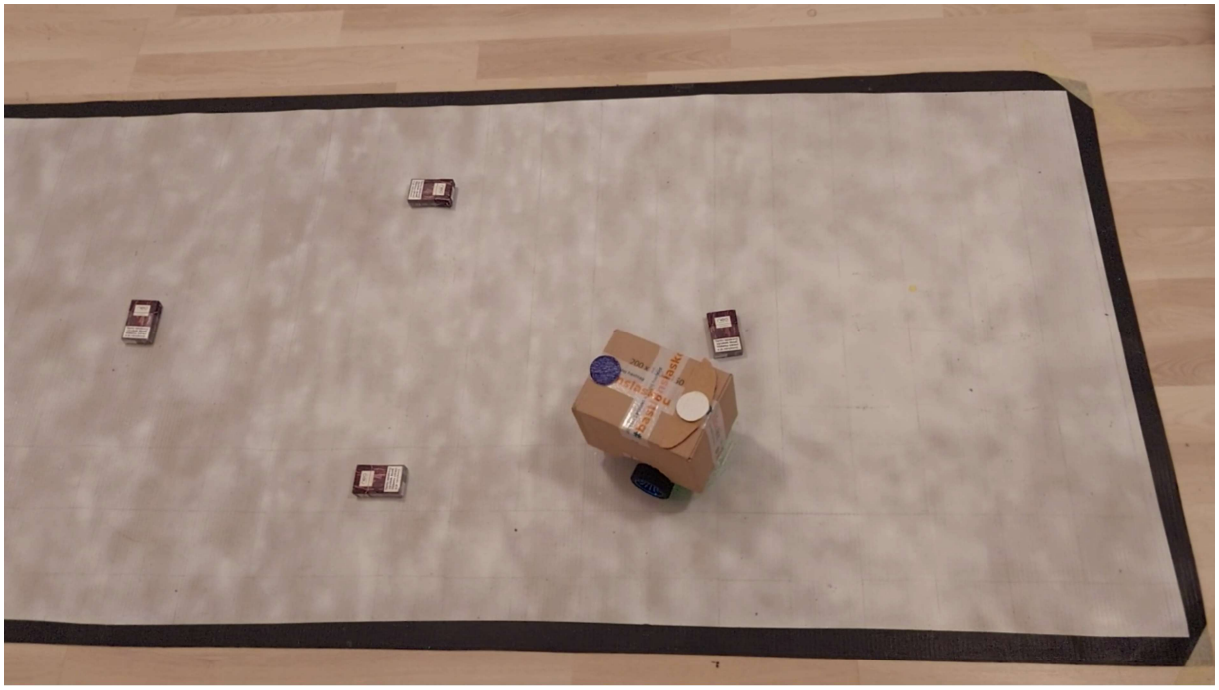
Εικόνα 71: Ρομπότ στο σημείο 11, διαδρομή 2



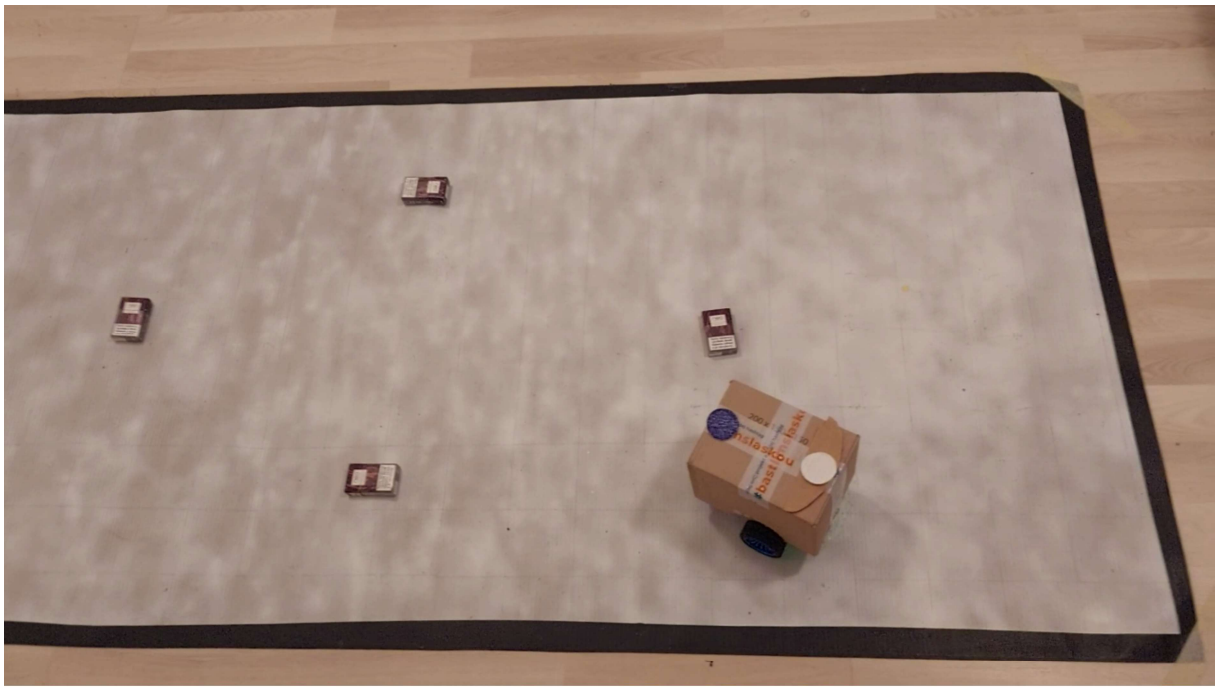
Εικόνα 72: Αλλαγή κατεύθυνσης ρομπότ για σημείο 12, διαδρομή 2



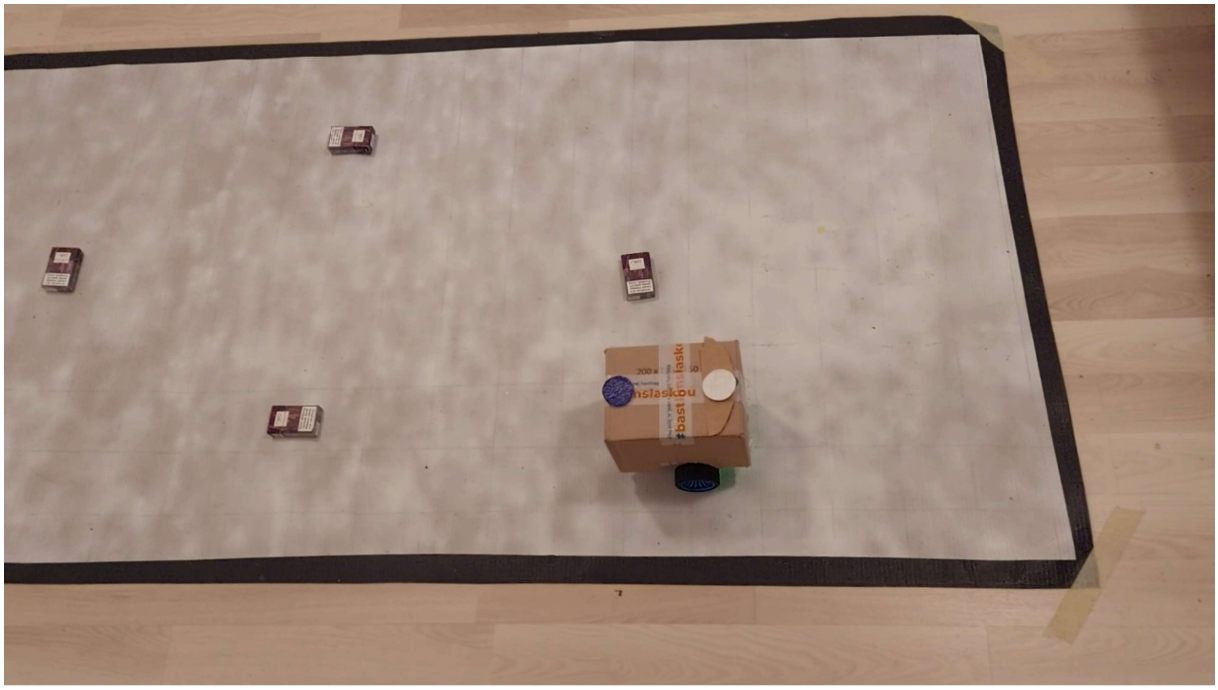
Εικόνα 73: Ρομπότ στο σημείο 12, διαδρομή 2



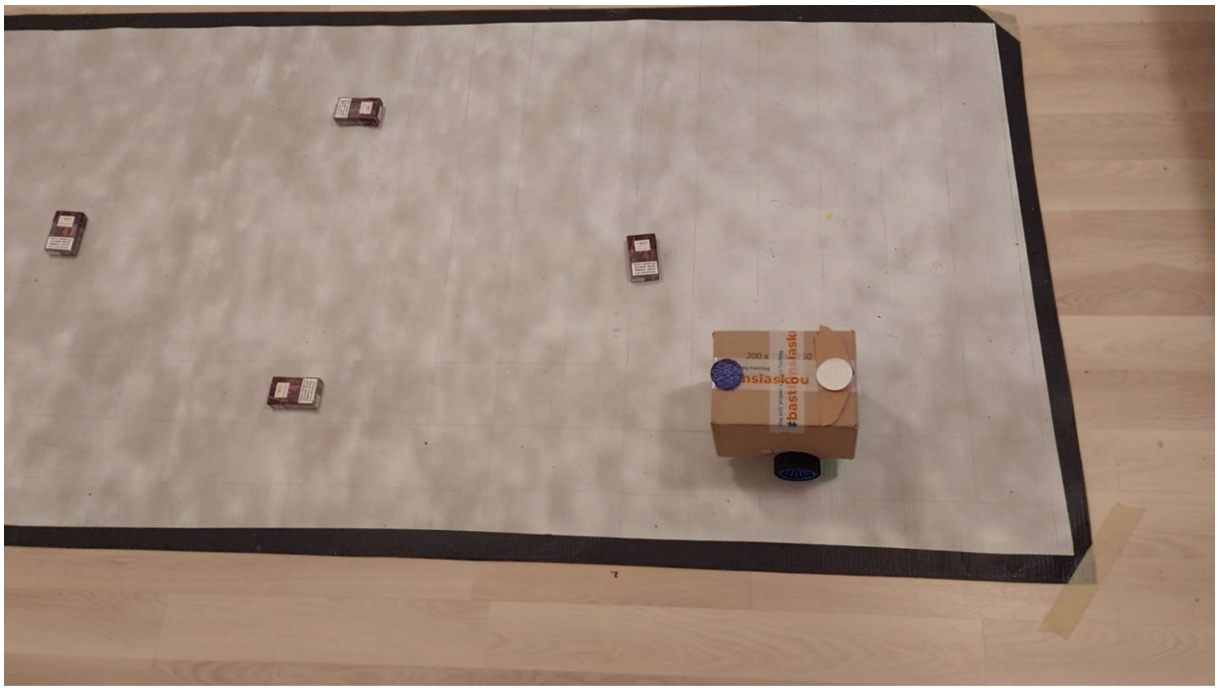
Εικόνα 74: Αλλαγή κατεύθυνσης ρομπότ για σημείο 13, διαδρομή 2



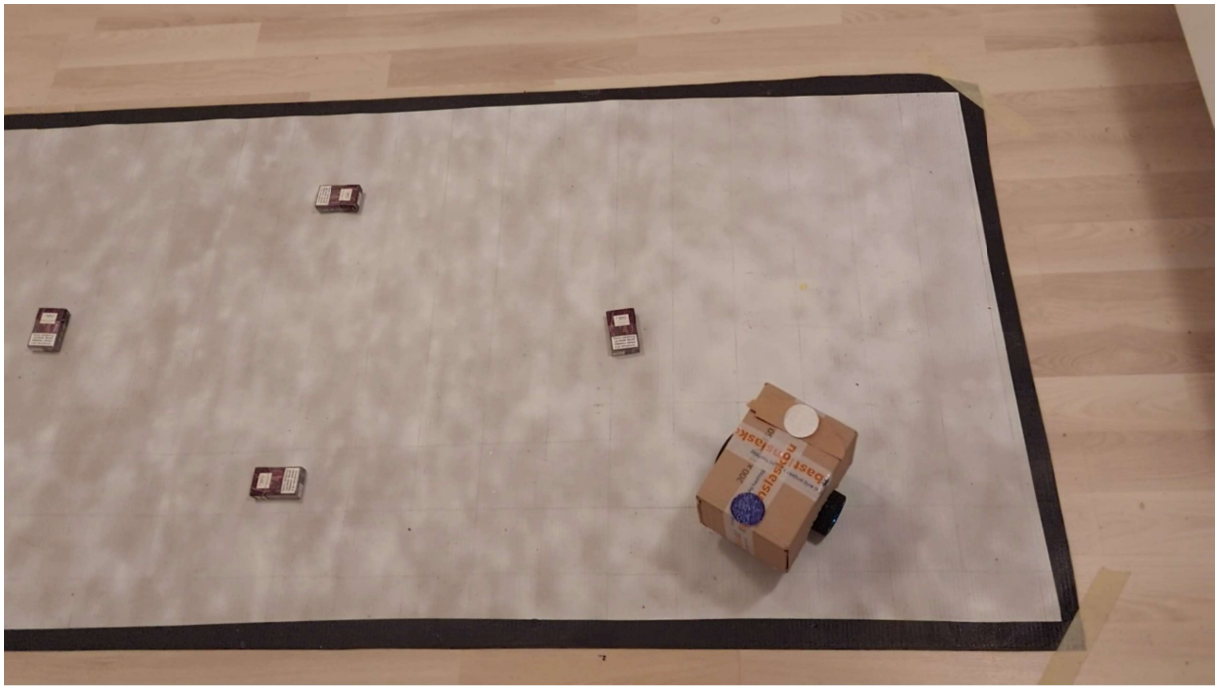
Εικόνα 75: Ρομπότ στο σημείο 13, διαδρομή 2



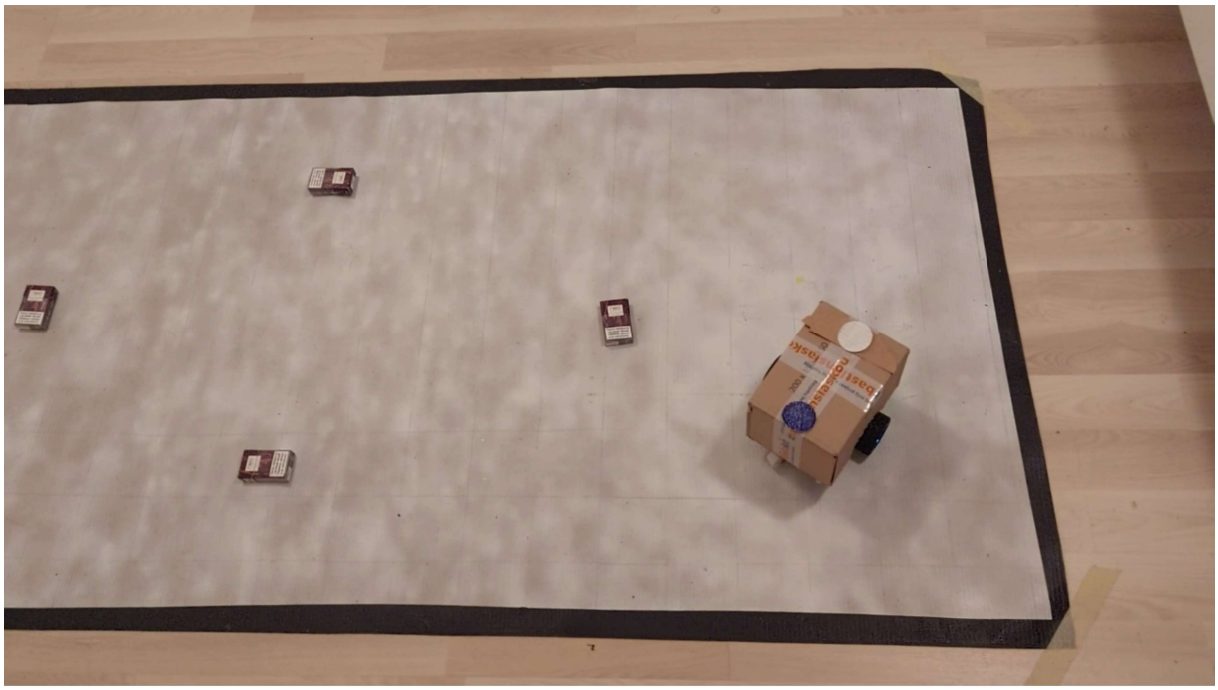
Εικόνα 76: Αλλαγή κατεύθυνσης ρομπότ για σημείο 14, διαδρομή 2



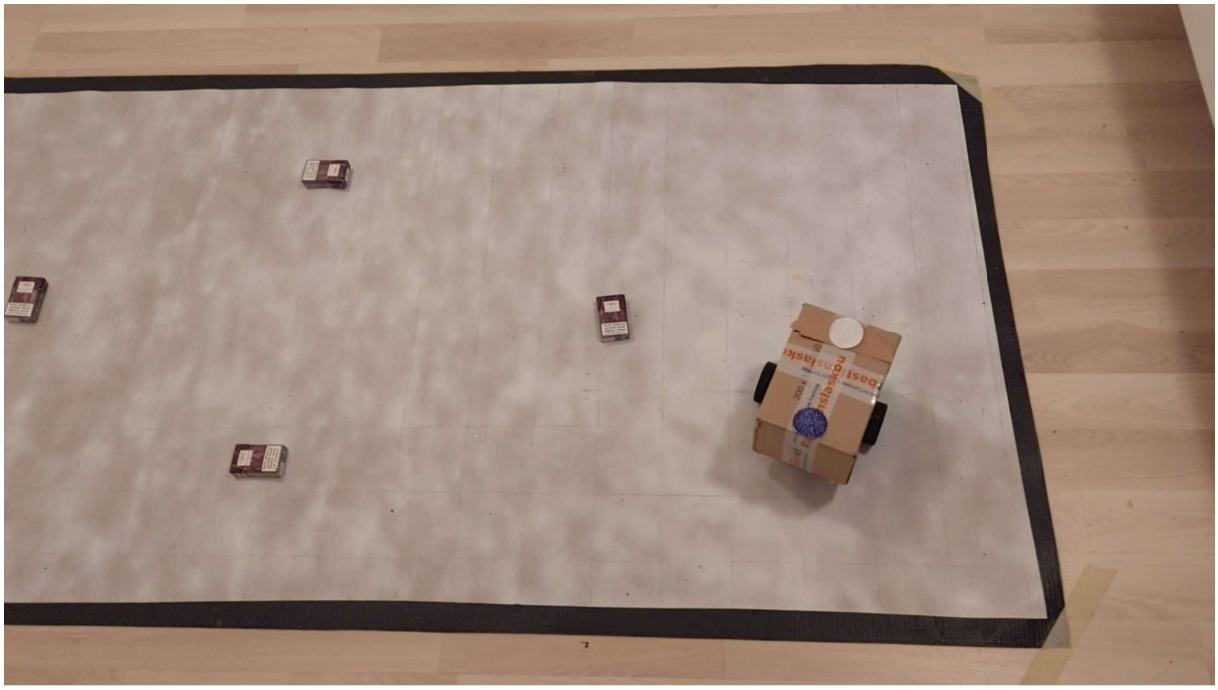
Εικόνα 77: Ρομπότ στο σημείο 14, διαδρομή 2



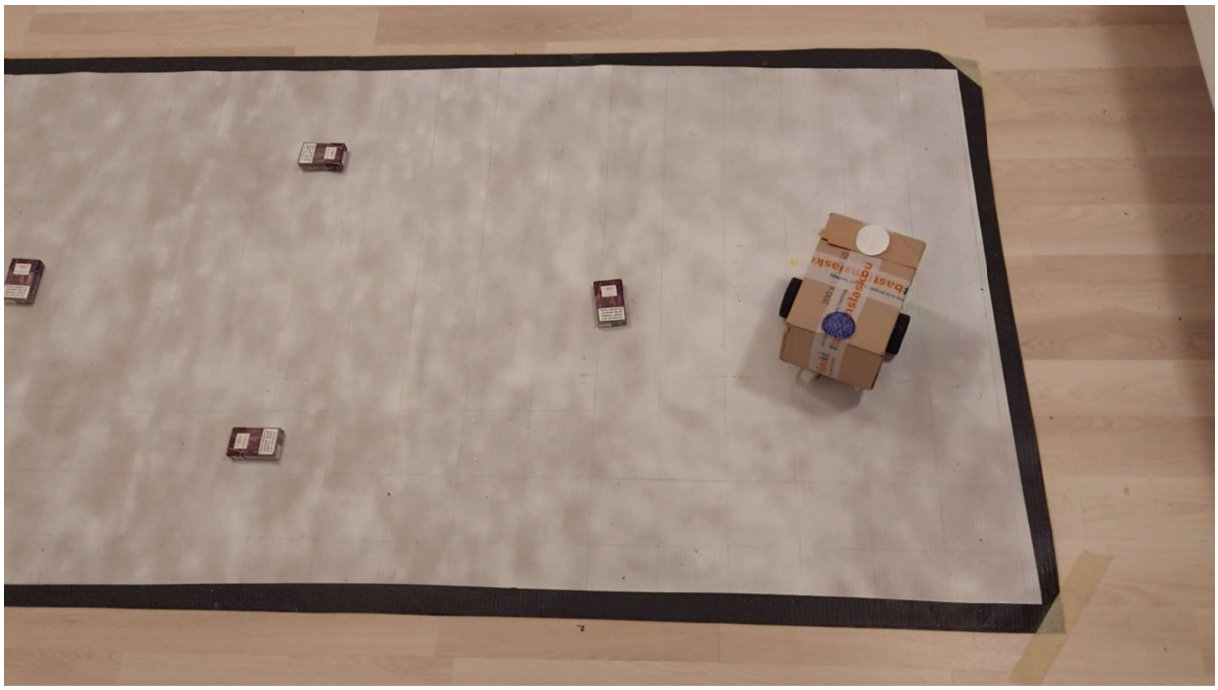
Εικόνα 78: Αλλαγή κατεύθυνσης ρομπότ για σημείο 15, διαδρομή 2



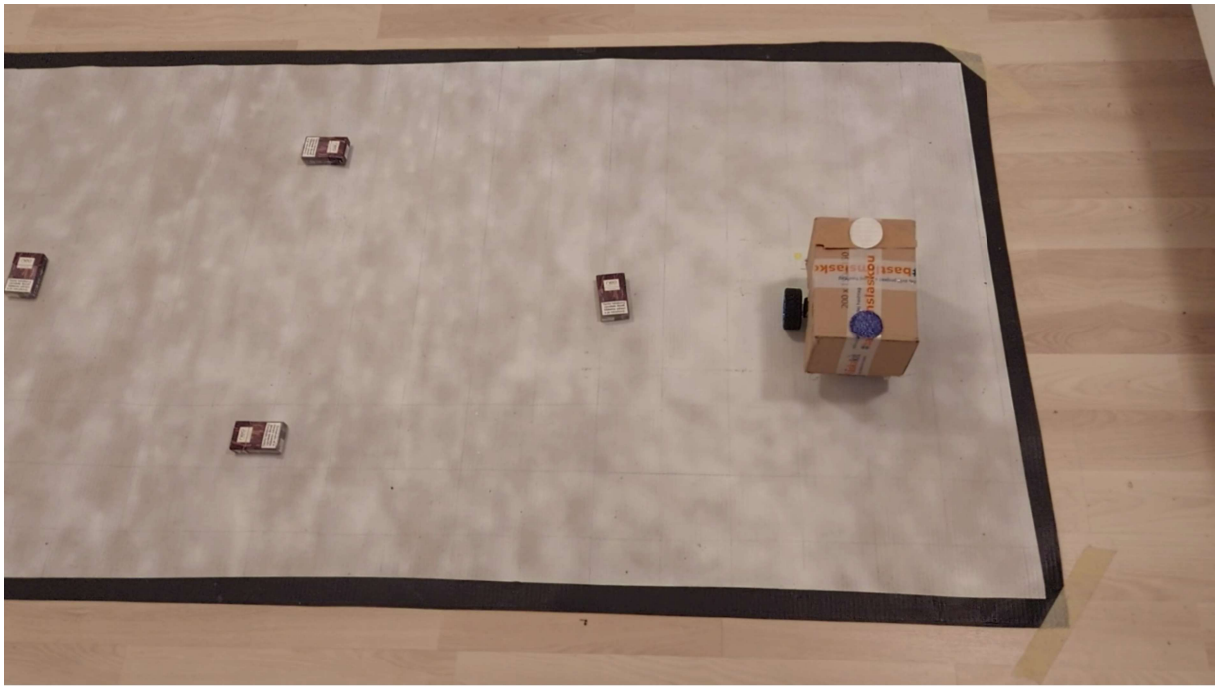
Εικόνα 79: Ρομπότ στο σημείο 15, διαδρομή 2



Εικόνα 80: Αλλαγή κατεύθυνσης ρομπότ για σημείο 16, διαδρομή 2

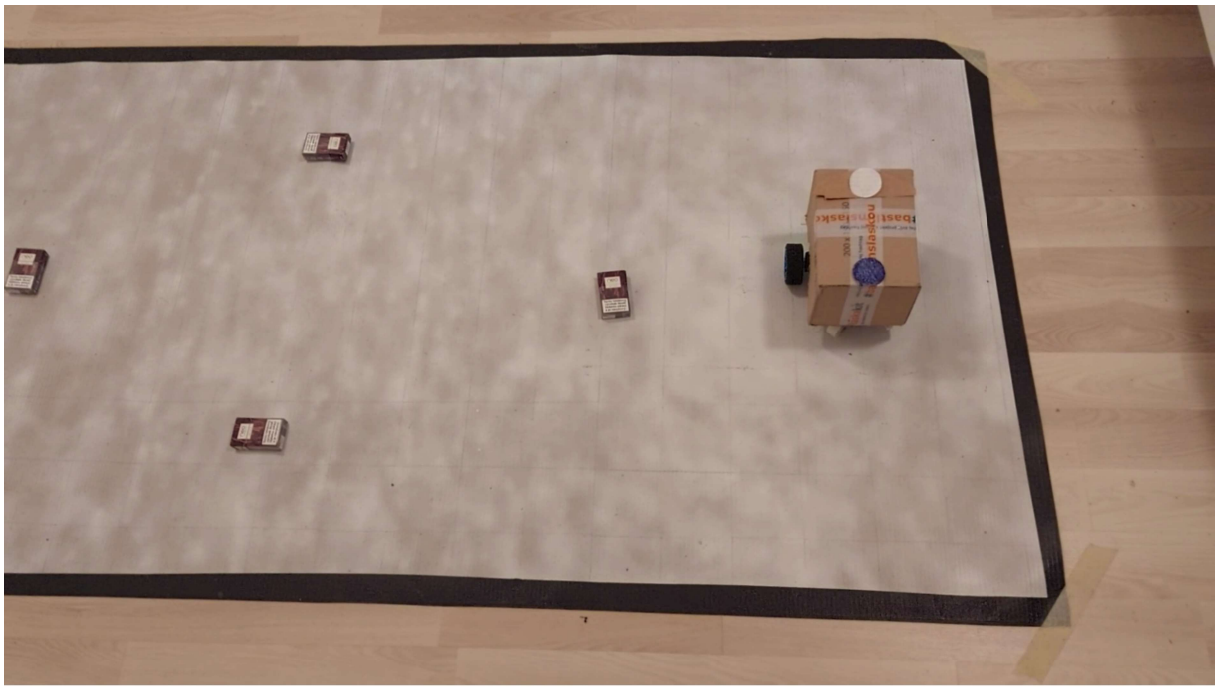


Εικόνα 81: Ρομπότ στο σημείο 16, διαδρομή 2



Εικόνα 82: Αλλαγή κατεύθυνσης ρομπότ για σημείο 17, διαδρομή 2

Η Εικόνα 83 δείχνει το ρομπότ στην τελική του θέση. Όπως φαίνεται από τις εικόνες το ρομπότ κατάφερε να ακολουθήσει την διαδρομή που υπολόγισε η μέθοδος PRM και να σταματήσει σε απόσταση μικρότερη των 10cm από τον τερματισμό.Να σημειωθεί πως και στις δύο διαδρομές χρειάστηκε να γίνει διόρθωση της γωνίας του ρομπότ. Η διαδικασία αυτή δεν απεικονίζεται σε κάποια από τις φωτογραφίες διότι η διαφορά ήταν μικρή και δεν γίνεται εύκολα αντιληπτή με φωτογραφίες.



Εικόνα 83: Ρομπότ στο σημείο 17, διαδρομή 2

Κεφάλαιο 11: Συμπεράσματα

Όπως δείχνουν και τα πειραματικά αποτελέσματα του προηγούμενου κεφαλαίου ο έλεγχος αυτοκινούμενου με χρήση κάμερας και μεθόδου PRM είναι εφικτός. Μεγάλο ρόλο στην διαδικασία του ελέγχου παίζει η κάμερα. Θα πρέπει να γίνει βαθμονόμιση της κάμερας ώστε η διαδρομή που θα υπολογιστεί με την μέθοδο PRM να αντιστοιχεί στον χώρο κίνησης. Διαφορετικά, αν δεν γίνει βαθμονόμιση, η παραμόρφωση που εισάγει η κάμερα στην εικόνα έχει ως αποτέλεσμα οι συντεταγμένες των εμποδίων να μην υπολογίζονται σωστά και η διαδρομή που θα υπολογιστεί δεν θα ανταποκρίνεται στις πραγματικές θέσεις των εμποδίων. Ένας ακόμη παράγοντας που επηρεάζει την ακρίβεια της μεθόδου είναι η σωστή αναγνώριση του χώρου κίνησης. Αν ο χώρος κίνησης καταλαμβάνει μικρό μέρος στην εικόνα τότε γίνεται πιο δύσκολος ο προσδιορισμός των συντεταγμένων του σημείου της αφετηρίας, καθώς αν το αρχικό σημείο που θα τοποθετηθεί το ρομπότ έχει απόκλιση από τις συντεταγμένες της αφετηρίας που έχει υπολογίσει ο αλγόριθμος το ρομπότ, δεν θα ακολουθήσει επιτυχημένη διαδρομή. Κάνοντας περικοπή της εικόνας ώστε ο χώρος κίνησης να αποτελεί το μεγαλύτερο μέρος της γίνεται πολύ εύκολο να αντιστοιχιστούν οι συντεταγμένες της αφετηρίας όπως τις "βλέπει" ο αλγόριθμος στην εικόνα με το πραγματικό σημείο στον χώρο κίνησης.

Για την κατασκευή του ρομπότ είναι σημαντικό να προσδιοριστεί ο αριθμός παλμών μιας πλήρους περιστροφής του κάθε μοτέρ. Αν τα χαρακτηριστικά των μοτέρ είναι διαφορετικά και υπάρχει μεγάλη απόκλιση του αριθμού των παλμών μεταξύ των δύο τότε δυσκολεύεται ο έλεγχος των μοτέρ και δεν είναι δυνατόν να προβλεφθεί η συμπεριφορά του. Επιπλέον πρέπει να γίνει έλεγχος της ταχύτητας των μοτέρ για την ομαλή επιτάχυνσή τους. Αν τα μοτέρ ξεκινήσουν με μεγάλη αρχική ταχύτητα υπάρχει ο κίνδυνος κάποιος από τους τροχούς να ολισθήσει με αποτέλεσμα το ρομπότ να βγει εκτός της πορείας που έχει υπολογίσει η μέθοδος PRM. Η επιλογή της τροφοδοσίας μπορεί επίσης να επηρεάσει το σύστημα. Η μπαταρία που θα χρησιμοποιηθεί θα πρέπει να έχει αρκετή ισχύ ώστε το ρομπότ να μπορέσει να ολοκληρώσει την διαδρομή.

Κεφάλαιο 12: Μελλοντικές βελτιώσεις

Η πρώτη βελτίωση που μπορεί να γίνει στο σύστημα είναι η επιτήρηση της τάσης της μπαταρίας του ρομπότ. Κατά την διεξαγωγή των πειραμάτων παρατηρήθηκε πως αν η τάση της μπαταρίας πέσει αρκετά γίνεται δύσκολος ο έλεγχος των μοτέρ με αποτέλεσμα να μην μπορεί να ακολουθήσει την διαδρομή που υπολόγισε η μέθοδος. Αυτό μπορεί να αποφευχθεί μετρώντας την τάση της μπαταρίας κι αν διαπιστωθεί πως είναι κάτω από κάποιο όριο το ρομπότ να μην ξεκινάει την διαδρομή και να βγάζει κάποιο μήνυμα σφάλματος. Μια δεύτερη βελτίωση είναι στην ασύρματη επικοινωνία. Όπως υλοποιήθηκε το σύστημα δεν γίνεται έλεγχος ότι τα δεδομένα έχουν αποσταλεί και ληφθεί σωστά. Για αυτό προτείνεται να δημιουργηθεί πρωτόκολλο επικοινωνίας που θα ελέγχει αν έχει γίνει σωστή ανταλλαγή δεδομένων και αν εντοπιστεί κάποιο λάθος να γίνεται επανάληψη της τελευταίας επικοινωνίας. Επιπλέον μπορεί να βελτιωθεί ο αλγόριθμος εντοπισμού του χώρου κίνησης. Η υπάρχουσα υλοποίηση βασίζεται στον εντοπισμό των ακμών του χαρτονιού που χρησιμοποιήθηκε ως χώρος κίνησης, έχοντας ως δεδομένο ότι γύρο από το χαρτόνι δεν υπάρχουν άλλα αντικείμενα. Αν υπάρχουν αντικείμενα γύρο από το χαρτόνι τότε ο αλγόριθμος δεν μπορεί να εντοπίσει τον χώρο κίνησης σωστά. Για να αυξηθεί η αξιοπιστία του συστήματος προτείνεται η ανάπτυξη αλγορίθμου για την διόρθωση θέσης του ρομπότ. Αυτό σε συνδυασμό με τον αλγόριθμο για την διόρθωση γωνίας που έχει υλοποιηθεί θα αυξήσουν την ακρίβεια του συστήματος και η πιθανότητα σύγκρουσης του ρομπότ με κάποιο εμπόδιο θα μειωθεί. Τέλος στο hardware του ρομπότ μπορεί να προστεθεί ένας αισθητήρας υπερήχων. Ο αισθητήρας αυτός μπορεί να χρησιμοποιηθεί για την επίλυση δύο προβλημάτων. Το πρώτο είναι ο εντοπισμός εμποδίων τα οποία προστέθηκαν μετά τον υπολογισμό της διαδρομής και δεν είναι γνωστά στο σύστημα. Με την χρήση του αισθητήρα υπερήχων το ρομπότ μπορεί να εντοπίσει αυτά τα εμπόδια αν πρόκειται να προσκρούσει πάνω τους και να ειδοποιήσει το σύστημα να υπολογίσει νέα διαδρομή. Το δεύτερο πρόβλημα αφορά την χρήση του συστήματος σε μεγαλύτερη κλίμακα. Αν το σύστημα χρησιμοποιείται στην βιομηχανία είναι σημαντικό να μην προκαλέσει τραυματισμούς στους εργαζόμενους του χώρου. Με την χρήση του αισθητήρα υπερήχων το ρομπότ θα μπορεί να εντοπίσει τυχόν εργαζόμενους που διασχίζουν τον χώρο και να προσαρμόσει την ταχύτητά του ώστε να μην προκληθούν τραυματισμοί.

Βιβλιογραφία

1: Wikipedia contributors, Deterministic system,
2: Wikipedia contributors, Stochastic optimization,
3: Lydia E. Kavraki, Petr Svestka, Jean-Claud Latombe, Mark H. Overnars, Probabilistic Roadmaps for path planning in high-dimensional configuration sapces,
4: Nancy M. Amato, O. Burchan Bayazit, Lucia K. Dale, Christopher Jones, Daniel Vallejo, Choosing Good Distance Metrics and Local Planners for Probabilistic RoadmapMethods, 1998
5: Σπυρίδων Καζαρλής, Εργαστηριακές σημειώσεις μαθήματος "Αυτόνομα Ρομποτικά Συστήματα",
6: Roland Geraerts Mark H. Overmars, A Comparative Study of ProbabilisticRoadmap Planners,

Παράρτημα Α: Κώδικας Arduino Uno

```
#include <util/atomic.h>
#include <SoftwareSerial.h>

// Motor A pins
#define ENC1_A 2 //Εξοδος κωδικοποιητή 1
#define ENC2_A 9 // Έξοδος κωδικοποιητή 2
#define PWM_A 10 //Έλεγχος ταχύτητας
#define IN_A 12  //Ελεγχος κατεύθυνσης
//Motor B pins
#define ENC1_B 3 // Έξοδος κωδικοποιητή 1
#define ENC2_B 8 // Έξοδος κωδικοποιητή 2
#define PWM_B 11 //Ελεγχος ταχύτητας
#define IN_B 13  //Ελεγχος κατεύθυνσης

#define PULSES_ONE_REVOLUTION 600 //Για τον υπολογισμό των RPM
//Κλάση PID
class MyPID
{
 private:
   float kp, ki, kd, signalMax;
   float errorPrev, errorInteg;

 public:
   //Δομητής
   MyPID(): kp(1), ki(0), kd(0), signalMax(255), errorInteg(0.0) {}

   //Μέθοδος για την ρύθμιση παραμέτρων
   void setParams(float kpIn, float kiIn, float kdIn, float signalMaxIn)
   {
     kp = kpIn;
     ki = kiIn;
```

```
  kd = kdIn;
  signalMax = signalMaxIn;
}

//Μέθοδος για τον μηδενισμό των σφαλμάτων που έχουν υπολογιστεί
void clearErrors(void)
{
  errorInteg = 0;
  errorPrev = 0;
}

//Μέθοδος για έλεγχο ταχύτητας μοτέρ
void evalu( int value, int target, float deltaT, int &power)
{
  //Σφάλμα
  int error = target - value;

  //Σφάλμα ολοκλήρωσης
  errorInteg = errorInteg + error * deltaT;

  //Σφάλμα διαφόρισης
  float errorDer = (error - errorPrev) / deltaT;

  //Σήμα ελέγχου
  float Signal = kp * error + ki * errorInteg + kd * errorDer;

  //Ταχύτητα μοτέρ
  power = (int) fabs(Signal);
  if ( power > signalMax)
  {
    power = signalMax;
  }
```

```
  }
};

//Μεταβλητές υπολογισμού ταχύτητας
long prevTA = 0, prevTB = 0;
int prevTempA =0, prevTempB = 0;

volatile int senseA = 0, senseB = 0; //Μεταβλητές μέτρησης παλμών

//Μεταβλητές φιλτραρίσματος ταχύτητας
float velFiltA = 0, velFiltB = 0;
float velPrevA = 0, velPrevB = 0;

int tempA = 0, tempB = 0; //Προσωρινές μεταβλητές που αντιγράφουν τα SenseA και SenseB
                          //για υπολογισμό από το κυρίως πρόγραμμα
static int target = 0;    //Μεταβλητή αποθήκευσης απαραίτητων παλμών
static int velTarget = 5; //Επιθυμητή ταχύτητα, χρησιμοποιείται για έλεγχο ταχύτητας

//Σήματα Rx/Tx Pins πλακέτας Bluetooth
SoftwareSerial blue(A4, A5); //Rx, Tx
MyPID pid_A;
MyPID pid_B;

void setup() {
  Serial.begin(9600);
  blue.begin(9600);

  //Αρχικοποίηση motorA
  pinMode(ENC1_A, INPUT);
  pinMode(ENC2_A, INPUT);
  pinMode(PWM_A, OUTPUT);
  pinMode(IN_A, OUTPUT);
```

```
  //Αρχικοποίηση motorB
  pinMode(ENC1_B, INPUT);
  pinMode(ENC2_B, INPUT);
  pinMode(PWM_B, OUTPUT);
  pinMode(IN_B, OUTPUT);

  //Αρχικοποίηση ρουτινών εξωτερικής διακοπής
  attachInterrupt(digitalPinToInterrupt(ENC1_A), SensorA, RISING);
  attachInterrupt(digitalPinToInterrupt(ENC1_B), SensorB, RISING);
  //Δεξιόστροφη περιστροφή μοτέρ
  digitalWrite(IN_A, HIGH);
  digitalWrite(IN_B, HIGH);

  //Αρχικοποίηση παραμέτρων PID
  pid_A.setParams(30, 12, 0, 255);
  pid_B.setParams(25, 14, 0.08, 255);
}

void loop() {

  char command; //Μεταβλητή αποθήκευσης δεδομένων bluetooth
  //Serial.println("Waiting command");
  //blue.println("Waiting command");
  blue.println(1, DEC);      //Αποστολή '1'
  while (!blue.available()); //Αναμονή για απόκριση με δεδομένα κατεύθυνσης

  blue.readBytes(&command, 1); //Αποθήκευση δεδομένων κατεύθυνσης στη μεταβλητή
                               //command
  //Serial.print("Command:");
  //Serial.print('\t');
  //Serial.println(command);

  switch (command)//Επιλογή κατεύθυνσης
```

```
{
 //Μπροστά
 case 'f':
   {
     velTarget = 4;
     start(command);
     break;
   }
 //Πίσω
 case 'b':
   {
     velTarget = 4;
     start(command);
     break;
   }
 //Δεξιά
 case 'r':
   {
     velTarget = 2;
     start(command);
     break;
   }
 //Αριστερά
 case 'l':
   {
     velTarget = 2;
     start(command);
     break;
   }
 //Λάθος εντολή
 default:
   {
```

```
      //Serial.println("Wrong Input");
      blue.println("Wrong Input");
      break;
    }
  }
}

void start(char dir)
{
 blue.println(2, DEC);//Αποστολή '2'
 while (!blue.available());// Αναμονή για απόκριση με δεδομένα αριθμός παλμών

 target = blue.parseInt();/Αποθήκευση αριθμού παλμών στην μεταβλητή target
 setDirection(dir);//Ρουτίνα ρύθμισης κατεύθυνσης μέσω L298P

 //Αρχικοποίηση μεταβλητών ως 0
 ATOMIC_BLOCK(ATOMIC_RESTORESTATE)//Ιδιο με noInterupts()
 {
  senseA = 0;
  senseB = 0;
 }
 tempA = 0;
 tempB = 0;
 velFiltA = 0;
 velPrevA = 0;
 velFiltB = 0;
 velPrevB = 0;
 prevTA = micros();
 pid_A.clearErrors();
 pid_B.clearErrors();
 do {
  int pwrA = 0, pwrB = 0;//Μεταβλητές για έλεγχο ταχύτητας μοτέρ
```

```
float velocityA = 0, velocityB = 0;//Μεταβλητές για υπολογισμό ταχύτητας
ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
{
  //Αντιγραφή δεδομένων παλμών από RAM
  tempA = senseA;
  tempB = senseB;
}

long currT = micros();//Τρέχων χρόνος σε μs
float deltaT = ((float) (currT - prevTA)) / 1.0e6;//Διαφορά χρόνου σε s
velocityA = (tempA - prevTempA) / deltaT;//Ταχύτητα MotorA
velocityB = (tempB - prevTempB) / deltaT;//Ταχύτητα MotorB
prevTA = currT;//Αποθήκευση χρόνου για τον επόμενο υπολογισμό ταχύτητας
prevTempA = tempA;// Αποθήκευση παλμών motorA για επόμενο υπολογισμό ταχύτητας
prevTempB = tempB;// Αποθήκευση παλμών motorB για επόμενο υπολογισμό ταχύτητας

// Μετασχηματισμός ταχύτητας σε RPM
float velA = PULSES_ONE_REVOLUTION / (velocityA * deltaT);
float velB = PULSES_ONE_REVOLUTION / (velocityB * deltaT);

// Χαμηλοπερατό φίλτρο  (25 Hz cutoff)
velFiltA = 0.854 * velFiltA + 0.0728 * velA + 0.0728 * velPrevA;
velPrevA = velA;

velFiltB = 0.854 * velFiltB + 0.0728 * velB + 0.0728 * velPrevB;
velPrevB = velB;

//Υπολογισμός PID για έλεγχο ταχύτητας motorA
pid_A.evalu(velFiltA, velTarget, deltaT, pwrA);

//Υπολογισμός PID για έλεγχο ταχύτητας motorB
pid_B.evalu(velFiltB, velTarget, deltaT, pwrB);
```

```
  analogWrite(PWM_A, pwrA);//Ρύθμιση ταχύτητας motorA
  analogWrite(PWM_B, pwrB);// Ρύθμιση ταχύτητας motorB

//Αποσφαλμάτωση
Serial.print(velTarget);
Serial.print(" ");
Serial.print(velFiltA);
//Serial.print(velA);
Serial.print(" ");
Serial.print(velFiltB);
//Serial.print(velB);
Serial.print(" ");
Serial.println();

}
while (tempB < target && tempA < target);//Αν ένα από τα μοτέρ έχει φτάσει το στόχο
                                          //παλμών τερματισμός βρόχου επανάληψης

analogWrite(PWM_A, 0);//Σταμάτημα motorA
analogWrite(PWM_B, 0);// Σταμάτημα motorB

//Μηδενισμός μεταβλητών
ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
{
  senseA = 0;
  senseB = 0;
}
tempA = 0;
tempB = 0;
prevTempA = 0;
prevTempB = 0;
velFiltA = 0;
```

```
 velPrevA = 0;
  velFiltB = 0;
  velPrevB = 0;
}

void setDirection(char dir )////Ρουτίνα ρύθμισης κατεύθυνσης μέσω L298P
{
  switch (dir)
  {
    case 'f'://Μπροστά
      {
        digitalWrite(IN_A, HIGH);
        digitalWrite(IN_B, HIGH);
        break;
      }
    case 'b'://Πίσω
      {
        digitalWrite(IN_A, LOW);
        digitalWrite(IN_B, LOW);
        break;
      }
    case 'l'://Αριστερά
      {
        digitalWrite(IN_A, HIGH);
        digitalWrite(IN_B, LOW);
        break;
      }
    case 'r'://Δεξιά
      {
        digitalWrite(IN_A, LOW);
        digitalWrite(IN_B, HIGH);
        break;
```

```
      }
    default://Λάθος εντολή
      {
        blue.println("Wrong Input");
        break;
      }
  }
}

//Ρουτίνα διακοπής μέτρηση παλμών motorA
void SensorA()
{
  senseA = senseA + 1;
}
// Ρουτίνα διακοπής μέτρηση παλμών motorB
void SensorB()
{
  senseB = senseB + 1;
}
```

Παράρτημα Β: Κώδικας Matlab

```
load('cameraParams.mat');%Φόρτωσε χαρακτηριστικά κάμερας
%Μηχανικά χαρακτηριστικά ρομπότ
wheelCircumference = pi * 0.065; %μέτρα
robotWidth = 0.105; %μέτρα
pulsesPerRotation = 600;
robotPose = 0;

% %Αρχικοποίηση Bluetooth
hc=MyBt;
hc.deviceName = "HC-06";
%Αν δεν υπάρχει σύνδεση, σύνδεση με  Uno
if exist ('dev', 'var') == 0
   dev = hc.connect;
   dev.BytesAvailableFcnMode = 'terminator';
   dev.Terminator = 'LF';
end

%Αρχικοποίηση κάμερας
if exist ('cam', 'var') == 0
        cam =webcam;
end
cam.resolution = '1280x720';  %1280x720, 640x480, 640x360

map = snapshot(cam);%Λήψη φωτογραφίας

map = undistortImage(map,cameraParams,'OutputView','valid');%Αφαίρεση παραμόρφωσης
κάμερας
mapgray=rgb2gray(map);%Μετασχηματισμός εικόνας σε γκρι
mapblur = imgaussfilt(mapgray, 4.6);%Εφαρρμογή φίλτρου Gauss
edgemap = edge(mapblur, 'sobel');%Εντοπισμός ακμών

%Μεταβλητές για περικοπή εικόνας
whiteCol ={};
whiteRow={};
colPrev = -1;
rowPrev = -1;

%Σάρωση εικόνας και εντοπισμός ακμών
for col = 1:600
        for row = 250:1400 %150:1100
        if(edgemap(col,row) > 0)
        if (col~=colPrev)
                whiteCol=[whiteCol, col];
            colPrev = col;
```

```
            end
        if (row~=rowPrev)
                whiteRow=[whiteRow, row];
                rowPrev = row;
            end
        end
        end
end

%Περικοπή εικόνας βάση των ακμών που εντοπίστηκαν
cropedMap = imcrop(map, [ min(cell2mat(whiteRow)) min(cell2mat(whiteCol))
(max(cell2mat(whiteRow)) - min(cell2mat(whiteRow))) (max(cell2mat(whiteCol)) -
min(cell2mat(whiteCol)))]);

map=rgb2gray(cropedMap);%Μετασχηματισμός κομμένης εικόνας σε γκρι
map=imbinarize(map);%Μετασχηματισμός κομμένης εικόνας ασπρόμαυρη
map=imcomplement(map);%Αντιστροφή χρωμάτων
map = robotics.BinaryOccupancyGrid(map, 400);%Δημιουργία binary occupancy grid
figure('Name', 'Binary Occupancy Grid');show(map)%Εμφάνηση binary occupancy grid

robotRadius = 0.15;% Ακτίνα ρομπότ για διόγκωση
mapInflated = copy(map);%  Αντιγραφή Occupancy Grid
inflate(mapInflated,robotRadius);%  Διόγκωση νέου Occupancy Grid
prm = robotics.PRM ;%  Δημιουργία PRM
prm.Map = mapInflated;%  Ορισμός Occupancy Grid που θα χρησιμοποιηθεί
prm.NumNodes = 150;% Ρύθμιση αριθμών κόμβων
prm.ConnectionDistance = 0.2;% Ρύθμιση μέγιστης απόστασης σύνδεσης κόμβων

startLocation = [0.3 0.6];% Θέση εκκίνησης
endLocation = [2 0.6];% Θέση τερματισμού
path = findpath(prm, startLocation, endLocation);%  Εκτέλεση PRM
figure('Name', 'PRM');show(prm);% Εμφάνιση λύσης


distance = calcDistance(path);%Συνάρτηση υπολογισμού απόστασης μεταξύ κόμβων
degrees = calcDegrees(path);%Συνάρτηση υπολογισμού γωνίας μεταξύ κόμβων
input('press Enter to continue');
count = 0;%Χρησιμοποιείται από συνάρτηση διόρθωση γωνίας για αποφυγή ατέρμονα
βρόχου

for i=1 : length(distance)%Επανάληψη για κάθε κόμβο
   while count < 10%count = 10 εντοπισμός απέτυχε
      estimate = snapshot(cam);%Λήψη φωτογραφίας
      [estimateCenter, estimateDeg] = EstimateRobotPosition(estimate, cameraParams,
whiteRow, whiteCol);%Εντοπισμός γωνίας ρομπότ
      count = count + 1;
```

```
        if estimateCenter > 0%Επιτυχής εντοπισμός
            disp('Position')
            robotPose = estimateDeg;%Ανανέωση γωνίας ρομπότ
            robotCenter = estimateCenter;%Ανανέωση θέσης ρομπότ
            count = 0;
            break%Εξοδος από βρόχο
        end
    end
    if(count >= 9)%Εντοπισμός απέτυχε
        disp("Detection failed");
        robotPose = degrees(i);%Ορισμός γωνίας ρομπότ με την τιμή που υπολογίστηκε
        count = 0;
    end

    %Συνάρτηση υπολογισμός κατεύθυνσης και μοίρες στροφής
    [direction, turn] = calcTurn(degrees(i), robotPose);

    if abs(turn) > 180%Βρές την κατεύθυνσή με τις λιγότερες μοίρες για την στροφή
        turn = abs(360 - abs(turn));
        if direction == 'l'
            direction = 'r';
        elseif direction == 'r'
            direction = 'l';
        end
    end

    arcFromDegrees = degreesToArc(turn, robotWidth);%Υπολογισμός τόξου στροφής
    arcFromDegrees = distanceToPulses(arcFromDegrees, pulsesPerRotation,
wheelCircumference);%Υπολογισμός παλμών στροφής
    arcFromDegrees = round(arcFromDegrees);%Στρογγυλοποίηση προς τα πάνω
    dist = distanceToPulses(distance(i), pulsesPerRotation, wheelCircumference);
%Υπολογισμός παλμών για ευθεία κίνηση
    dist = round(dist);%Στρογγυλοποίηση προς τα πάνω

    fprintf(dev, '%c', direction);%Αποστολή εντολής κατεύθυνση στροφής
    data = fscanf(dev, '%i');%Διάβασε απάντηση
    while(data~=2)%Αναμονή μέχρι να ληφθεί η τιμή '2'
        data = fscanf(dev, '%i');
    end
    fprintf(dev, '%i', abs(arcFromDegrees));%Αποστολή παλμών στροφής
    data = fscanf(dev, '%i');%Διάβασε απάντηση
    while(data~=1)% Αναμονή μέχρι να ληφθεί η τιμή '1'
        data = fscanf(dev, '%i');
    end
    pause(0.5);%Χρειάζεται για εντοπισμό γωνίας
    while count < 10%Έλεγχος αν η γωνία είναι εντός επιθυμητού εύρους
```

```
        estimate = snapshot(cam);%Λήψη φωτογραφίας
        [estimateCenter, estimateDeg] = EstimateRobotPosition(estimate, cameraParams,
whiteRow, whiteCol);%Υπολογισμός γωνίας ρομπότ
        count = count + 1;
        if estimateCenter > 0
            robotPose = estimateDeg;
            robotCenter = estimateCenter;
            count = 0;
            disp("Correction turn");
            while ( abs(robotPose - degrees(i)) >= 3.5)%Σφάλμα > κατώγφλι
                correctPose( robotPose, degrees(i), robotWidth, pulsesPerRotation,
wheelCircumference, dev );%Διόρθωση γωνίας
                estimate = snapshot(cam);%Λήψη φωτογραφίας
                [estimateCenter, estimateDeg] = EstimateRobotPosition(estimate, cameraParams,
whiteRow, whiteCol);%%Υπολογισμός γωνίας ρομπότ
                robotPose = estimateDeg;%Ανανέωση μεταβλητής και έλεγχος αν σφάλμα <
κατώφλι
            end
            break
        end
    end

    if(count >= 9)%Εντοπισμός απέτυχε
        disp("Detection failed");
        count = 0;
    end

    fprintf(dev, '%c', 'f');%Αποστολή εντολής για κίνηση προς τα μπροστά
    data = fscanf(dev, '%i');%Διάβασε απάντηση
    while(data~=2)%Αναμονή μέχρι να ληφθεί η τιμή '2'
        data = fscanf(dev, '%i');
    end
    fprintf(dev, '%i', dist);%Αποστολή παλμών
    data = fscanf(dev, '%i');%Διάβασε απάντηση
    while(data~=1)% Αναμονή μέχρι να ληφθεί η τιμή '1'
        data = fscanf(dev, '%i');
    end
    pause(0.5);%Χρειάζεται για εντοπισμό γωνίας
end

    %Συνάρτηση για μετατροπή απόστασης σε παλμούς
    function pulses = distanceToPulses(dist, numberOfPulses, circumference)
    pulses = ((numberOfPulses*dist) / circumference);
    end

    %Συνάρτηση για μετατροπή γωνίας σε τόξο
```

```
function arc = degreesToArc(deg, width)
arc = (2*pi*width) * (abs(deg) / 360);
end

%Συνάρτηση για υπολογισμό κατεύθυνσης στροφής
function [dir, turn] = calcTurn(deg, pose)

turn = deg - pose;
if turn > 0
   dir = 'l';
elseif turn < 0
   dir = 'r';
elseif turn == 0
   dir = 'f';
end
end

%Συνάρτηση για διόρθωση γωνίας ρομπότ
function correctPose( robotPose, degrees, robotWidth, pulsesPerRotation,
wheelCircumference, dev )
if(robotPose ~= 'e')%Έλεγχος αν ο εντοπισμός ήτα επιτυχής
   if(robotPose < 0)%Μετασχηματισμός γωνίας από [-180, 180] σε [0, 360]
      robotPose = robotPose + 360;
   end
   [direction, turn] = calcTurn(degrees, robotPose);
   if abs(turn) > 180
      turn = abs(360 - abs(turn));
      if direction == 'l'
         direction = 'r';
      elseif direction == 'r'
         direction = 'l';
      end
   end
   arcFromDegrees = degreesToArc(turn, robotWidth);
   arcFromDegrees = distanceToPulses(arcFromDegrees, pulsesPerRotation,
wheelCircumference);
   arcFromDegrees = round(arcFromDegrees);
   if(arcFromDegrees == 0)
      arcFromDegrees = 1;
   end
   fprintf(dev, '%c', direction);
   data = fscanf(dev, '%i');
   while(data~=2)
      data = fscanf(dev, '%i');
   end
   fprintf(dev, '%i', abs(arcFromDegrees));
```

```
        data = fscanf(dev, '%i');
        while(data~=1)
            data = fscanf(dev, '%i');
        end
        pause(0.5);
    end
    end

%Συνάρτηση για υπολογισμό γωνίας ρομπότ μέσω κάμερας
function [robotCenter, robotDeg] = EstimateRobotPosition(A, cameraParams, whiteRow,
whiteCol)
     %Αφαίρεση παραμόρφωση εικόνας
    estimate = undistortImage(A,cameraParams,'OutputView','valid');
    %Μετατροπή εικόνας σε γκρι
    estimate=rgb2gray(estimate);
    %Περικοπή εικόνας
    estimate = imcrop(estimate, [ min(cell2mat(whiteRow)) min(cell2mat(whiteCol))
(max(cell2mat(whiteRow)) - min(cell2mat(whiteRow))) (max(cell2mat(whiteCol)) -
min(cell2mat(whiteCol)))]);
     %Εντοπισμός σκούρων κύκλου
    [centersDark,radiiDark] = imfindcircles(estimate,[10 14],'ObjectPolarity','dark',
'Sensitivity',0.94);
    %Εντοπισμός φωτινού κύκλου
    [centersBright,radiiBright] = imfindcircles(estimate,[10 14],'ObjectPolarity','bright',
'Sensitivity',0.95);
    centers = [centersDark ; centersBright];%Αποθήκευση κέντρων κύκλων
     if size(centers) == 2 %Έλεγχος ότι μόνο 2 κύκλοι εντοπίστηκαν
      line = true;
    else
      line = false;
    end
     if line == true%Αν μόνο 2 κύκλοι εντοπίστηκαν
    x_line = centers(:,1);
    y_line = centers(:,2);

    dist=calcDistance(centers);%Υπολογισμός απόστασης κέντρων
    deg=calcDegrees(centers);%Υπολογισμός γωνίας κέντρων
    robotCenter = [];%Μεταβλητή για τον υπολογισμό θέσης του ρομπότ, δεν χρησιμοποιείται
στο κυρίως πρόγραμμα
    robotCenter = [robotCenter, (x_line(2) - (dist/2)*cos(deg2rad(deg)))];%Μετασχηματισμός
πολικών συντεταγμένων σε καρτεσιανές
    robotCenter = [robotCenter, (y_line(2) - (dist/2)*sin(deg2rad(deg)))];%Μετασχηματισμός
πολικών συντεταγμένων σε καρτεσιανές
    %viscircles(robotCenter, 2, 'Color', 'blue');
    for i=1: length(centers)%Μετασχηματισμός θέσης κέντρων από pixels σε X,Y
        centers(i,1) = (centers(i,1)/400);% + 0.09;
```

```
        centers(i,2) = 1.066 - (centers(i,2)/400);
    end
    robotCenter(1,1) = (robotCenter(1,1)/400);% + 0.09;
    robotCenter(1,2) = 1.066 - (robotCenter(1,2)/400);
    robotDeg=calcDegrees(centers);%Υπολογισμός γωνίας ρομπότ
    else
        robotCenter = -1;
        robotDeg = 'e';%Σηματοδοτεί σφάλμα
    end
end

%Συνάρτηση για τον υπολογισμό απόστασης μεταξύ κόμβων
function DIST = calcDistance(A)
x=[];
y=[];
DIST=[];
for i=1 : size(A)
    x = [x, A(i,1)];
    y = [y, A(i,2)];
end

for i=1 : (length(x) - 1)
    tempX = (x(i+1) - x(i))^2;
    tempY = (y(i+1) - y(i))^2;
    temp = sqrt(tempY + tempX);
    DIST = [DIST, temp];
end
end

%Συνάρτηση για τον υπολογισμό γωνίας μεταξύ κόμβων
function DEG = calcDegrees(A)
x=[];
y=[];
DEG=[];
for i=1 : size(A)
    x = [x, A(i,1)];
    y = [y, A(i,2)];
end

for i=1 : (length(x) - 1)
    temp = atan2d( (y(i+1) - y(i) ), ( x(i+1) - x(i) ) );
    if(temp < 0)
        temp = temp + 360;
    end
    DEG = [DEG, temp];
end
```

```
end
```