Εργασία Ευφυών Συστημάτων Ρομπότ
27/09/2020

Υπατία Δάμη (8606)

Μουτζόγλου Δημήτρης (8319)

Πρόκληση 1η:

Στο πρώτο ζητούμενο απαιτούνταν μια reactive αρχιτεκτονική για την αποφυγή εμποδίων κατά την περιπλάνηση του ρομπότ στο χώρο με την χρήση των τιμών που επέστρεφε το LIDAR. Στην produceSpeedsLaser λοιπόν, πρωτίστως αρχικοποιούμε τις τιμές της γραμμικής και της γωνιακής ταχύτητας και των μεταβλητών max_mean,max_index και light_width που θα χρησιμοποιηθούν για την χάραξη της πορείας του ρομπότ. Στη συνέχεια , ελέγχουμε την τιμή της global μεταβλητής self.emergency αν είναι διάφορη του μηδενός, μεταβλητή που έχουμε ορίσει για την περίπτωση που το ρομπότ έχει πλησιάσει επικίνδυνα σε τοίχο και η κανονική διαδικασία ανάθεσης ταχυτήτων είναι ανεπαρκής στο να αποτρέψει την πρόσκρουση. Αν δεν είναι ορίζουμε ως γραμμική ταχύτητα την μέγιστη αρνητική που επιτρέπεται (-0.3) προκειμένου το όχημα να κάνει όπισθεν και πολλαπλασιάζουμε την τιμή της self.emergency με 0.15 με σκοπό την αλλαγή του προσανατολισμού του ρομπότ και την αποτροπή ταλάντωσης του ρομπότ ανάμεσα σε συγκεκριμένες θέσεις. Σε ένθετο if ελέγχουμε την τιμή της self.emergency αν έχει υπερβεί το 0 και αφαιρούμε κατά 1 σε περίπτωση αλήθειας. Τέλος, η self.emergency αυξάνεται κατά 1 και επιστρέφονται γραμμική και γωνιακή ταχύτητα. Η παραπάνω διαδικασία έχει ως εξής καθώς στην κανονική ροή κώδικα ορίζουμε την τιμή της self.emergency -2 προκειμένου η οπισθογωνία που θα διαγράψει το ρομπότ να διαρκέσει για δύο κύκλους ανάθεσης τιμών. Έπειτα, χωρίζουμε τις τιμές του laser_scan σε 22 δέσμες (περιέχουν 30 τιμές απόστασης η κάθε μία) , για τις οποίες και υπολογίζεται το συνολικό άθροισμα των αποστάσεων που μετράει το laser. Αποθηκεύεται στην max_mean το μεγαλύτερο άθροισμα δέσμης και στην max_index η τιμή της μετρούμενης απόστασης το κέντρο της δέσμης αυτής. Η τιμή της γωνιακής ταχύτητας προκύπτει από την διαίρεση με το 2 ,της διαφοράς της θέσης στον πίνακα scan του κέντρου της δέσμης με το μεγαλύτερο άθροισμα και του μήκους του πίνακα (667), και του πολλαπλασιασμού με τον παράγοντα k που ισούται με την μέγιστη δυνατή γωνιακή προς το μήκος του πίνακα scan. Η παραπάνω διαδικασία εξυπηρετεί τον σκοπό το ρομπότ να στρίβει με γωνιακή ταχύτητα ανάλογη της γωνίας ανάμεσα στην κεντρική ακτίνα του πίνακα και του κέντρου της δέσμης με το μεγαλύτερο άθροισμα. Ουσιαστικά , το ρομπότ θα στρίψει με μεγαλύτερη ταχύτητα αν το τόξο των αποστάσεων που περιέχει το μεγαλύτερο άθροισμα , δηλαδή "βλέπουν" σε πιο ανοιχτό πεδίο , έχει μεγάλη γωνία από τον κεντρικό προσανατολισμό του. Η τιμή της γραμμικής ταχύτητας προκύπτει από τον πολλαπλασιασμό της μεσαίας τιμής του πίνακα scan, δηλαδή της τιμής που υποδεικνύει την τέλεια ευθεία απόσταση του ρομπότ από πιθανό εμπόδιο, με την μέγιστη επιτρεπτή ταχύτητα (0.3),τον παράγοντα 2 , και την διαίρεση του προϊόντος με το 10. Το 2 χρησιμοποιείται για την παραγωγή κατά το δυνατόν μεγαλύτερων γραμμικών ταχυτήτων ενώ το 10 είναι η στρογγυλοποιημένη μέγιστη απόσταση που μπορεί να επιστρέψει το LIDAR. Στην περίπτωση που το κελί scan[333] όντως περιέχει την μέγιστη δυνατή μέτρηση , η γραμμική ταχύτητα προκύπτει 0.6 ,εξου και ο έλεγχος για γραμμικές ταχύτητες μεγαλύτερες του 0.3 και η εξίσωση αυτών με την μέγιστη δυνατή. Τέλος ακολουθεί ένας έλεγχος όπου εξετάζεται το περιεχόμενο των ομάδων κελιών (233-333,334-434), κελιών που περιέχουν τις

αριστερά και δεξιά εμπρόσθιες αποστάσεις που μετράει το LIDAR για το ρομπότ. Σε περίπτωση που κάποια από αυτές είναι μικρότερη της τιμής 0.3 (εμπειρική τιμή) ορίζουμε την μέγιστη όπισθεν και την μέγιστη γωνιακή με πρόσημο ανάλογο της θέσης του εμποδίου και ορίζουμε την τιμή της self.emergency ίση με -2. Τέλος, επιστρέφουμε ταχύτητες.

```
k=0.3/len(scan)
for i in range(0,22):
  sum=0
  for j in range(1,light_width):
    sum=sum+scan[i*light_width+j]
  if sum/light_width>=max_mean:
    max_mean=sum/light_width
    max_index=i*light_width+15

angular=k*(max_index-len(scan)/2) #the angular velocity is proportional to the value of the angle the robot has to turn
linear=(2*(0.3*scan[333]))/10     #the linear velocity is proportional of the distance between
                                  #the robot and the obstacles directly ahead of it
                                  #10 is by experience the maximum value the scan can read

if linear>0.3:    #restrict the values of speeds under 0.3
  linear=0.3
for i in range(0,100):
  if scan[233+i]<0.3:
    linear=-0.3
    angular=0.3
    self.emergency=2
  elif scan[334+i]<0.3:
    linear=-linear
angular=-0.3
    self.emergency=-2
```

Κατά κανόνα υπολογισμός ταχύτητας και έλεγχος επικίνδυνης εγγύτητας σε εμπόδιο.

Πρόκληση 2η:

Στην 2η πρόκληση ζητείται η οπτικοποίηση του μονοπατιού που πρόκειται να ακολουθήσει το ρομπότ για την επίσκεψη του επόμενου στόχου. Οι διαδοχικές συντεταγμένες του μονοπατιού όπως επιστρέφονται από την συνάρτηση create_path() είναι εκφρασμένες σε pixels, ενώ οι συντεταγμένες που οπτικοποιεί ο Rviz πρέπει να είναι εκφρασμένες σε cm.Επίσης το σημείο αναφοράς τους είναι η αρχική θέση του ρομπότ στον χάρτη, η οποία είναι διαφορετική από το σημείο αναφοράς του χάρτη. Για να γίνει λοιπόν η οπτικοποίηση των συντεταγμένων από τον Rviz ,πολλαπλασιάζουμε τις τιμές του x και y άξονα με το resolution , για να μετατραπούν από pixels σε cm, και έπειτα προστίθεται στο αποτέλεσμα η αρχική θέση του ρομπότ σχετικά με το σημείο αναφοράς του χάρτη, όπως φαίνεται στον παρακάτω κώδικα.

```
ps.pose.position.x =p[0]*self.robot_perception.resolution+self.robot_perception.origin['x']
ps.pose.position.y =p[1]*self.robot_perception.resolution+self.robot_perception.origin['y']
```

Πρόκληση 3η :

Σκοπός της πρόκλησης αυτής είναι ο υπολογισμός των ταχυτήτων του ρομπότ για την προσέγγιση του επόμενου στόχου του. Έχοντας στη διάθεσή μας την τρέχουσα θέση του ρομπότ , την τρέχουσα γωνία του σε σχέση με τους άξονες του χάρτη και τις συντεταγμένες του επόμενου στόχου, υπολογίζεται η γωνία που θα αποκτήσει το ρομπότ όταν θα επισκεφτεί τον στόχο ,με σημείο αναφοράς κοινό με την γωνία της τρέχουσας θέσης. Η διαφορά της γωνίας αυτής με την γωνία της τρέχουσας θέσης του ρομπότ είναι η γωνία που

θα πρέπει να στρίψει ώστε να κατευθυνθεί προς το στόχο του. Έπειτα , υπολογίζεται η γωνιακή ταχύτητα για τέσσερις διαφορετικές περιπτώσεις , αναλόγως το πρόσημο της γωνίας που προέκυψε και την απόλυτη τιμή της , προκειμένου το ρομπότ να πραγματοποιήσει την μικρότερη δυνατή περιστροφή. Π.χ εάν το πρόσημο της γωνίας είναι θετικό , αλλά το μέγεθός της είναι μεγαλύτερο από 180 μοίρες, αντί για θετική γωνιακή ταχύτητα δίνουμε στο ρομπότ αρνητική γωνιακή ταχύτητα για να διαγράψει μικρότερο τόξο. Τέλος , το αποτέλεσμα που προκύπτει πολλαπλασιάζεται με την μέγιστη δυνατή τιμή που μπορεί να πάρει η γωνιακή ταχύτητα. Για να μην ξεπεραστεί αυτή η τιμή, σε έναν έλεγχο if θέτουμε τις τιμές που τυγχάνει να ξεπερνάνε το όριο με την μέγιστη αυτή τιμή.

```
if dtheta>=0 and dtheta<pi:
  angular=dtheta/pi
elif dtheta>0 and dtheta>=pi:
  angular=(dtheta-2*pi)/pi
elif dtheta<=0 and dtheta>-pi:
  angular=dtheta/pi
elif dtheta<0 and dtheta<-pi:
  angular=(dtheta+2*pi)/pi

angular=angular*0.3
```

Όσο για την γραμμική ταχύτητα ,υπολογίζεται ως η διαφορά της μονάδας με την απόλυτη τιμή της γωνιακής ταχύτητας, υψωμένη στη δεκάτη έκτη και πολλαπλασιασμένη με τη μέγιστη επιτρεπτή τιμή της . Επιλέγουμε μεγάλη δύναμη για να αποφύγουμε φαινόμενα overshooting και να καταστήσουμε την γωνιακή ταχύτητα πρωτεύουσας σημασίας στην χάραξη της πορείας προς τον στόχο.

```
linear=0.3*(pow(1-abs(angular),16))
```

Ένας παράγοντας που μας επιτρέπει τη χρήση σχετικά μεγάλων γραμμικών ταχυτήτων είναι το ότι το μονοπάτι που ακολουθεί το ρομπότ δημιουργείται συνήθως πάνω στο διάγραμμα Voronoi του χάρτη , του οποίου τα σημεία ισαπέχουν από τα εμπόδια και έτσι οι περιπτώσεις πρόσκρουσης σε εμπόδια λόγω γραμμικής ταχύτητας είναι ελάχιστες.

Πρόκληση 4η :

Στην πρόκληση αυτή καλούμαστε να δώσουμε συνδυαστικές ταχύτητες στο ρομπότ ,έτσι ώστε να κατευθύνεται προς τον στόχο του αλλά ταυτόχρονα να είναι ικανό να αποφεύγει και τυχόν εμπόδια. Για να πετύχουμε αυτό το στόχο εφαρμόζουμε την τεχνική motor schema συνδυάζοντας τις ταχύτητες που υπολογίστηκαν στην 3η πρόκληση για την προσέγγιση στόχου, και των ταχυτήτων που υπολογίστηκαν για την αποφυγή εμποδίων. Για τον υπολογισμό της γραμμικής ταχύτητας αποφυγής εμποδίων λαμβάνονται υπόψη οι τιμές του lidar που αντιστοιχούν στις μπροστινές 60 μοίρες του οπτικού πεδίου του ρομπότ, καθώς αυτό είναι το κρίσιμο πεδίο όσον αφορά την πρόσκρουση σε κάποιο εμπόδιο λόγω γραμμικής ταχύτητας. Για το εύρος αυτών των τιμών του lidar υπολογίζεται ένα άθροισμα γραμμικών ταχυτήτων,(μια για κάθε τιμή) που προκύπτουν ως συνημιτονοειδής συνάρτηση της γωνίας από την κεντρική τιμή του lidar διαιρεμένη με την απόσταση από το εμπόδιο που εντοπίζεται από το lidar στη συγκεκριμένη θέση. Η ταχύτητα μετά τον

υπολογισμό της αποκτά αντίθετο πρόσημο, καθώς εφόσον πρόκειται για αποφυγή εμποδίων θα πρέπει να οδηγήσει το ρομπότ στην αντίθετη κατεύθυνση. Επομένως, με αυτή τη φόρμουλα η γραμμική ταχύτητα μειώνεται από όρους που αφορούν σημεία κοντά στο κέντρο του lidar, όπου η συνημιτονοειδής συνάρτηση αυξάνεται , και από όρους με μικρή απόσταση από εμπόδια. Επισημαίνεται πως λόγω της ύπαρξης δεκαδικών αποστάσεων στο χάρτη της συγκεκριμένης προσομοίωσης, μιας τάξης κάτω του 0, πολλαπλασιάζεται με το 10 η απόσταση από τα εμπόδια ,διότι σε περίπτωση ύπαρξης δεκαδικής τιμής στον διαιρέτη , υψωμένη στο τετράγωνο, θα προκαλούσε πολύ μεγαλύτερη μείωση της ταχύτητας του ρομπότ, γεγονός ανεπιθύμητο. Η ίδια διαδικασία ακολουθείται για τον υπολογισμό της γωνιακής ταχύτητας μόνο που τώρα λαμβάνουμε υπόψη τις τιμές του lidar που αντιστοιχούν σε 60 μοίρες στις δύο πλευρές του ρομπότ, εφόσον αυτό το πεδίο που αφορά την πρόσκρουση σε εμπόδια λόγω γωνιακής ταχύτητας. Τελικά οι τιμές που προέκυψαν για τις δύο ταχύτητες πολλαπλασιάζονται με παράγοντες που προέκυψαν από διαδοχικά πειράματα, έτσι ώστε να παράγονται ικανοποιητικά μεγάλες τιμές ταχυτήτων , αλλά και να επιτυγχάνεται αποφυγή εμποδίων.

```
for i in range(260,410):
  theta=(30*(333-i))/72
  u_obs+=(math.cos(math.radians(theta)))/pow(10*scan[i],2)
u_obs=-u_obs

for i in range(0,149):
  theta=60+i*0.4047
  w_obs+=(math.sin(math.radians(theta)))/pow(10*scan[i+407],2)  #left scan  w_obs
for i in range(0,149):
  theta=-120+i*0.4047              # 0.4047 is the proportion between cells and degrees
  w_obs+=(math.sin(math.radians(theta)))/pow(10*scan[i],2)  #right scan  w_obs
w_obs=-w_obs

linear=0.1*u_obs      #multiply speeds with scaling factor derived by trials
angular=0.045*w_obs
```

Ακολουθούν if έλεγχοι που καθορίζουν την επίδραση αυτών των ταχυτήτων στην συνολική ταχύτητα του ρομπότ, έχοντας ως στόχο την επίτευξη της μέγιστης δυνατής ταχύτητας του. Μετά από πειράματα προέκυψαν τα όρια για τις τιμές των ταχυτήτων όπως φαίνονται παρακάτω:

```
if linear>0.07 :
  linear=0.07
if linear<-0.07:
  linear=-0.07
if angular>0.3:
  angular=0.3
if angular<-0.3:
  angular=-0.3
```

Η γραμμική ταχύτητα εμποδίων παίρνει μικρό όριο , επομένως επηρεάζει λίγο την τελική ταχύτητα. Αντίθετα, η γωνιακή έχει πολύ μεγαλύτερο όριο , διότι όπως παρατηρήθηκε το ρομπότ κινδυνεύει κυρίως από εμπόδια που απαιτείται μεγάλη γωνιακή ταχύτητα προς την αντίθετη κατεύθυνση για να αποφευχθούν. Τελικά οι ταχύτητες αυτές προστίθενται στις ταχύτητες που αφορούν το στόχο του ρομπότ και προκύπτουν οι τελικές ταχύτητες του. Εάν λοιπόν το ρομπότ κατευθύνεται προς ένα στόχο και αναπτύσσει γωνιακή ταχύτητα προς αυτόν , αλλά ταυτόχρονα και προς ένα εμπόδιο , η αντίθετη γωνιακή ταχύτητα που θα

προκύψει από τις ταχύτητες αποφυγής εμποδίων θα αναγκάσει το ρομπότ να κινηθεί όσο χρειάζεται προς την αντίθετη κατεύθυνση , αποφεύγοντας το εμπόδιο επιτυχώς.

Πρόκληση 5η :

Στόχος της πρόκλησης αυτής είναι η εξυπνότερη επιλογή επόμενου υπό-στόχου, για την μείωση του χρόνου επίσκεψης του τελικού στόχου, αλλά και για την εξομάλυνση των κινήσεών του προς αυτόν. Για την διεξαγωγή της, στην ρουτίνα υπολογισμού του επόμενου υπό-στόχου ελέγχεται για όλους τους εναπομείναντες υπό-στόχους του ρομπότ, εάν η απόστασή τους από αυτό είναι μικρότερη από ένα κατώφλι ,που προκύπτει πειραματικά. Εάν ισχύει αυτό για κάποιον από αυτούς, τότε τίθεται σαν επόμενος υπό-στόχος, άσχετα αν το ρομπότ δεν έχει προσεγγίσει τον προηγούμενο υπό-στόχο του.

```
for i in range(self.next_subtarget+1,len(self.subtargets)):
  dist= math.hypot(\
    rx - self.subtargets[i][0], \
    ry - self.subtargets[i][1])
  if dist<15 :
    self.next_subtarget=i
```

Λούπα υπολογισμού επόμενου υπό-στόχου

Μία ενδιαφέρουσα παρατήρηση είναι πως καθώς μεγαλώνει η απόλυτη τιμή του κατωφλίου αυτού , εξομαλύνεται η πορεία του ρομπότ στο μονοπάτι ,καθώς κάνει λιγότερες απότομες κινήσεις και η διαδρομή γίνεται πιο ομαλή και ευθεία .

Πρόκληση 6η:

Το ζητούμενο της έκτης πρόκλησης ήταν η δημιουργία ενός έξυπνου τρόπου επιλογής στόχων για το ρομπότ ώστε να καταφέρει να εξερευνήσει το εμβαδό του χώρου με όσο το δυνατόν λιγότερή κίνηση. Υλοποιώντας στην selectTarget αρχικά ελέγχουμε αν το μήκος του πίνακα των κόμβων είναι μικρότερο ή ίσο του 2,δηλαδη το ρομπότ πλησιάζει στην πλήρη εξερεύνηση του χώρου. Σε περίπτωση αλήθειας ο στόχος καθορίζεται από την selectRandomTarget που εξασφαλίζει πως το ρομπότ θα κινηθεί προς τα εναπομείναντα ανεξερεύνητα κομμάτια του χάρτη (συμπεριλαμβανομένου και των περισσευούμενων κόμβων). Έπεται η λήψη των τιμών του Sonar που μας ενδιαφέρουν ,η αποθήκευση τους στις μεταβλητές sonar_left/right/front , το άθροισμα τους και ο έλεγχος αν το άθροισμα υπερβαίνει το 1.2 (εμπειρική τιμή).Ο έλεγχος αυτός συμβαίνει για να διαπιστωθεί αν το ρομπότ έχει βρεθεί σε τοπικά ελάχιστα, σε σημεία στον χάρτη περικυκλωμένα από εμπόδια. Σε περίπτωση αλήθειας τον στόχο θα καθορίσει η sonar_avoidance συνάρτηση που θα εξηγηθεί παρακάτω. Ακολουθεί έλεγχος για την περίπτωση time out ή αποτυχίας του planner να διαγράψει μονοπάτι για το ρομπότ. Σε περίπτωση αλήθειας , ως στόχος ορίζεται ο κόμβος που συγκέντρωσε το δεύτερο καλύτερο σκορ στον υπολογισμό του προηγούμενου κύκλου. Ένα ένθετο if ελέγχει την μεταβλητή timeout_happened και αν είναι 1 ο στόχος θα καθοριστεί εντέλει από την sonar_avoidance και η μεταβλητή timeout_happened θα μηδενιστεί. Εκτός του ένθετου if η timeout_happened παίρνει τιμή 1

και επιστρέφεται ο στόχος. Έπειτα από τους ελέγχους που απαιτούν ειδικό χειρισμό, ακολουθεί ο υπολογισμός του επόμενου στόχου με cost-based προσέγγιση. Θεωρούμε σημαντικότερα για το πρόβλημά μας τα κόστη που αφορούν topology, coverage και distance. Για τον υπολογισμό του τοπολογικού κόστους κάθε διαθέσιμου κόμβου χρησιμοποιείται ο πίνακας brushfire και ακολουθείται μία επαναληπτική διαδικασία όπου ελέγχεται η απόσταση του ρομπότ από εμπόδια ή ανεξερεύνητα σημεία στον κάθετο άξονα(μπρος και πίσω του ρομπότ) και στον οριζόντιο (δεξιά και αριστερά) .Σε ένα άθροισμα που αρχικοποιείται με τιμή μηδέν προσθέτουμε μία μονάδα για κάθε συνεχόμενο κελί με τιμή διάφορη του μηδενός(το εμπόδιο έχει τιμή 0 στον πίνακα brushfire) στην ίδια στήλη και στην ίδια γραμμή με τον εξεταζόμενο κόμβο δηλαδή μετράμε την απόσταση του κόμβου από εμπόδιο στην κατεύθυνση των τεσσάρων σημείων του ορίζοντα. Για τον υπολογισμό του κόστους κάλυψης του χώρου κάθε κόμβου χρησιμοποιείται ο πίνακας coverage και μία επαναληπτική διαδικασία που υπολογίζει το συνολικό άθροισμα των αποστάσεων των κόμβων από τα ακάλυπτα σημεία στον κάθετο και οριζόντιο άξονα όπως προηγουμένως. Αρχικά , ελέγχεται αν η θέση του κόμβου στο χώρο αντιστοιχεί σε κελί του πίνακα coverage, αν όχι αποδίδεται μεγάλη τιμή (5000) στο άθροισμα του κόμβου λόγω έλλειψης επαρκούς πληροφορίας και προχωράμε στον υπολογισμό για τον επόμενο κόμβο. Στον κατά κανόνα υπολογισμό προσθέτουμε μία μονάδα στο άθροισμα του κόμβου για κάθε συνεχόμενο κελί του πίνακα στην ίδια στήλη ή την ίδια γραμμή με τον κόμβο για τιμή διάφορη του εκατό. Ο πίνακας coverage περιέχει κελιά με τιμή 0 για τα σημεία του χώρου που δεν έχουμε επισκεφτεί ακόμα και 100 για τα χαρτογραφημένα. Στην προκειμένη περίπτωση αποδίδουμε μεγάλα αθροίσματα για κόμβους που είναι μακριά από χαρτογραφημένη περιοχή καθώς στόχευση μας είναι η πλήρης κάλυψη δωματίων που έχουν εν μέρει χαρτογραφηθεί . Στη συνέχεια , θα βρούμε τη μέγιστη τιμή (maxi) που βρίσκεται στον πίνακα κοστών κάλυψης και θα αποδώσουμε την τιμή 1.2*maxi σε κάθε κελί έχει τιμή ίση με 5000 δηλαδή σε κάθε κόμβο που οι "συντεταγμένες" του βρίσκονταν εκτός του πίνακα κάλυψης. Το κόστος απόστασης για κάθε κόμβο υπολογίζεται με βάση την ευκλείδεια απόσταση του από την παρούσα θέση του ρομπότ.

```
#using the brushfire array in order to calculate topology cost for each node
for n in nodes:
  sum_temp=0
  index=n[1]+1
  while brush[n[0],index]!=0:
    sum_temp+=1
    index+=1
    if index==numrows-1:  #numrows
      break
  index=n[1]-1
  while brush[n[0],index]!=0 :
    sum_temp+=1
    index-=1
    if index==0:
      break
  index=n[0]+1
  while brush[index,n[1]]!=0:
    sum_temp+=1
    index+=1
    if index==numcols-1:  #numcols
      break
  index=n[0]-1
  while brush[index,n[1]]!=0:
    sum_temp+=1
    index-=1
    if index==0:
      break

  topo_cost.append(sum_temp/4)
```

Λούπα υπολογισμού τοπολογικού κόστους.

Παρακάτω, δημιουργούμε λίστες όπου και θα αποθηκεύσουμε τα κανονικοποιημένα κόστη και το τελικό κόστος κάθε κόμβου που προέρχεται από την επιλογή αυθαίρετων βαρών που θα πολλαπλασιαστούν με τα πρώτα κόστη. Τα επιλεγόμενα βάρη που προέκυψαν από την διαδικασία του πειράματος, είναι 0,4 για το τοπολογικό και το κόστος κάλυψης και 0,2 για το κόστος απόστασης (άθροισμα βαρών ισούται με μονάδα). Στο συγκεκριμένο μοντέλο επιλέγουμε να δώσουμε βάση στο τοπολογικό κόστος και το κόστος κάλυψης καθώς ο υπολογισμός της ευκλείδειας απόστασης δεν λαμβάνει υπόψη πιθανά εμπόδια που παρεμβάλλονται. Υπογραμμίζεται πως στην περίπτωση που ο πίνακας coverage δεν έχει προλάβει να διαμορφωθεί , βαρύτητα δίνεται μονάχα σε κόστος απόστασης (0,4) και τοπολογίας(0,6). Τέλος, αναζητούμε τον δείκτη του κόμβου με το μικρότερο κόστος και αποθηκεύουμε αυτόν ως στόχο και τον αμέσως επόμενο ως self.node2_index_x/y. Απομένει ένας τελευταίος έλεγχος όπου εξετάζεται αν ο στόχος είναι ίδιος με αυτόν που υπολογίστηκε τον προηγούμενο κύκλο (αυτό συμβαίνει όταν το ρομπότ δεν μπορεί να προσεγγίσει τον στόχο στο χρονικό διάστημα που του δίνεται λόγω της άπωσης που του επιβάλλουμε για την αποφυγή συγκρούσεων). Σε περίπτωση αλήθειας καλείται να ορίσει τον στόχο η sonar_avoidance.

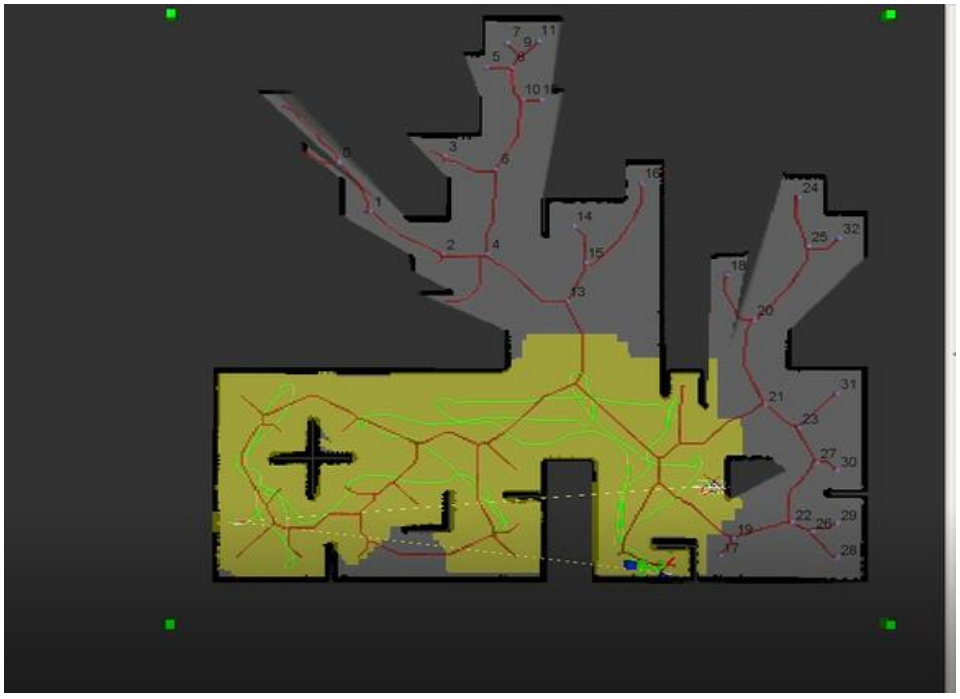
Sonar_avoidance: ονομάζουμε μια συνάρτηση που καλείται για τις περιπτώσεις που το ρομπότ περιβάλλεται από τοίχους/εμπόδια στις 3/4 κατευθύνσεις ή δυσκολεύεται να προσεγγίσει τον επιθυμητό κόμβο. Με την χρήση του πίνακα ogm και την θέση του ρομπότ στο χάρτη υπολογίζουμε ένα άθροισμα για 4 πιθανές κατευθύνσεις (βορρά, ανατολή, νότο ,δύση). Το άθροισμα αυτό αντανακλά την απόσταση που θα κάλυπτε το ρομπότ αν κινούνταν πάνω στην κατεύθυνση προτού έρθει σε σύγκρουση με εμπόδιο ή τοίχο. Επιλέγεται η κατεύθυνση (ανατολή- δύση διατηρούν το y,βορράς - νότος διατηρούν το x) με το μεγαλύτερο άθροισμα και ορίζεται ως στόχος το σημείο με συντεταγμένη το μέσον της απόστασης. Παραδείγματος χάριν αν το ρομπότ περιβάλλονταν από Βορρά - Ανατολή - Δύση από εμπόδια το μεγαλύτερο άθροισμα θα προέκυπτε ομολογουμένως από το Νότο , επομένως ως στόχος θα επιλέγονταν το σημείο που θα διατηρούσε την τετμημένη και θα όριζε ως τεταγμένη το ήμισυ του αθροίσματος (ακέραια τιμή). Η συνάρτηση αυτή επινοήθηκε προκειμένου το ρομπότ είτε να βελτιώνει την θέση του για να προσεγγίσει κάποιο στόχο που απέτυχε να προσεγγίσει στον προηγούμενο κύκλο ή να αποφεύγεται η σύγκρουση με τοίχους εξαιτίας των στενών περιθωρίων κίνησης σε σημεία του χάρτη.

```
#sonar_avoidance is a function that calculates the direction the robot should head in order to escape a tight spot
def sonar_avoidance (self,pose_global_x,pose_global_y,ogm,numcols,numrows):

        tally=[0,0,0,0] #a sum is calculated for north,east,south and west direction

        index=pose_global_y+1
        while ogm[pose_global_x,index]!=100:
            tally[0]=tally[0]+1
            index+=1
            if index==numrows-1:  #numrows
                break
        index=pose_global_y-1
        while ogm[pose_global_x,index]!=100 :
            tally[1]=tally[1]+1
            index-=1
            if index==0:
                break
        index=pose_global_x+1
        while ogm[index,pose_global_y]!=100:
            tally[2]=tally[2]+1
            index+=1
            if index==numcols-1:  #numcols
                break
        index=pose_global_x-1
        while ogm[index,pose_global_y]!=100:
            tally[3]=tally[3]+1
            index-=1
            if index==0:
                break
        index=tally.index(max(tally))
        if index==0:
            target=[pose_global_x,pose_global_y+(int)(tally[0]/2)]
            return target
        elif index==1:
            target=[pose_global_x,pose_global_y-(int)(tally[1]/2)]
            return target
        elif index==2:
            target=[pose_global_x+(int)(tally[2]/2),pose_global_y]
            return target
        elif index==3:
            target=[pose_global_x-(int)(tally[3]/2),pose_global_y]
            return target
```

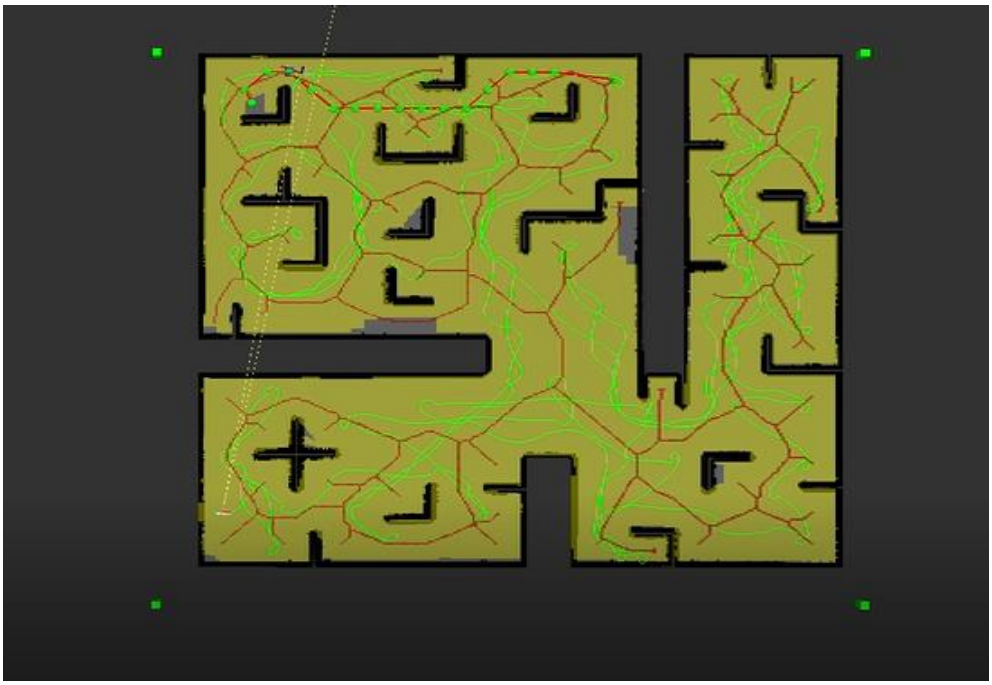
Σώμα sonar avoidance



Sonar avoidance

Παραδοχές:

- Ο υπολογισμός των ταχυτήτων στις προσκλήσεις 3,4 στηρίχθηκε σε τύπους από τις διαφάνειες του μαθήματος και συνεχείς δοκιμές από τις οποίες προέκυψαν πολλαπλασιαστικοί παράγοντες και κατώφλια για τον έλεγχο της ταχύτητας.



Full coverage and exploration of space

Στο παρακάτω λινκ υπαρχει το βιντεο με την πληρη καλυψη του χώρου

https://www.dropbox.com/s/m4q602y1klxb2yk/weeeeeee%20arreeee%20the%20chaampionsssssssss%20no%20time%20for%20loosers.mp4?dl=0