Ευφυή συστήματα ρομπότ

Εργασία

Κωστινούδης Ευάγγελος
ΑΕΜ: 8708
email: ekostinou@eng.auth.gr

9ο Εξάμηνο

2018-2019

Ο κώδικας πέρα από το αρχείο που ανέβηκε βρίσκεται στη σελίδα. Για κάθε ερώτημα υπάρχει σελίδα στο GitHub για κάθε αλλαγή που έγινε (εκτός αν έγιναν import κάποιες βιβλιοθήκες)

Challenge 1 - Laser-based obstacle avoidance

Στη μέθοδο “produceSpeedsLaser” υπολογίζουμε τις ταχύτητες με τους τύπους που παρουσιάστηκαν στη παρουσίαση 9 σελίδες 99-100. Οι τύποι των ταχυτήτων είναι:

$$u_{obs} = -\sum_{i=1}^{LaserRays} \frac{\cos(\theta_i)}{s_i^2} \quad \text{και} \quad \omega_{obs} = -\sum_{i=1}^{LaserRays} \frac{\sin(\theta_i)}{s_i^2}$$

όπου $\boldsymbol{\theta_i}$ οι γωνία του laser και $\boldsymbol{s_i}$ η απόσταση του laser.

produceSpeedsLaser

Στη μέθοδο “produceSpeeds” προσθέτουμε στην ταχύτητα ***l_laser*** 300 ώστε το ρομπότ να κινείται προς τα μπροστά εκτός αν βρίσκεται κοντά σε εμπόδιο. Διαιρούμαι τη ταχύτητα *a_laser* με 500 ώστε το ρομπότ να στρίβει πιο ομαλά. Τελικά αν κάποια από τις δύο ταχύτητες ξεπερνά το όριο $|u|>0,3$ τότε $u = 0,3\,sign(u)$.

produceSpeeds

Challenge 2 - Path visualization

Σε αυτό το κομμάτι πολλαπλασιάζομαι το σημείο του path με το resolution και προσθέτουμε το origin.

Visualization

Challenge 3 - Path following

Στο κομμάτι αυτό αρχικά υπολογίζουμε τη γωνία του στόχου και τη γωνία μεταξύ στόχου και ρομπότ. Έπειτα, υπολογίζουμε την γωνιακή ταχύτητα από τον τύπο:

$$\omega = \begin{cases} \dfrac{\Delta\theta}{\pi}, & \Delta\theta > -\pi \ \ και \ \ \Delta\theta < \pi \\ \dfrac{\Delta\theta - 2\pi}{\pi}, & \Delta\theta \geq \pi \\ \dfrac{\Delta\theta + 2\pi}{\pi}, & \Delta\theta \leq \pi \end{cases}$$

Ύστερα υπολογίζουμε τη τιμή της γραμμικής ταχύτητας

$$u = 0.3(1 - |\omega|)^{10}$$

και της γωνιακής:

$$\omega = 0.3\, sign(\omega) |\omega|^{1/6}$$

Οι τιμές των εκθετών προκύπτουν μετά από πειράματα ώστε να μην υπάρχει μεγάλη απόκλιση από το στόχο και επίσης το ρομπότ να κινείτε όσο το δυνατόν γρηγορότερα. Τελικά αν κάποια από τις δύο ταχύτητες ξεπερνά το όριο

$$|u|, |\omega| > 0,3 \quad \text{τότε} \quad u = 0,3\, sign(u),\ \omega = 0,3\, sign(\omega)$$

Path following

Challenge 4 - Path following & obstacle avoidance

Υπολογίζουμε τις δύο ταχύτητες από τους τύπους:

$$u = u_{goal} + u_{laser}^{3}\, 3^{-11} \quad \text{και} \quad \omega = \omega_{goal} + \omega_{laser}^{3}\, 3^{-11}$$

Αν κάποια από τις δύο ταχύτητες ξεπερνά το όριο

$$|u|, |\omega| > 0,3 \quad \text{τότε} \quad u = 0,3\, sign(u),\ \omega = 0,3\, sign(\omega)$$

Οι τιμές αυτές προκύπτουν από πειράματα ώστε να ακολουθείται η επιθυμητή διαδρομή και να αποφεύγονται τα εμπόδια.

Path following and obstacle avoidance

Challenge 5 - Smarter subgoal checking

Για να γίνεται καλύτερα ο έλεγχος στόχου του ρομπότ αντί να ελέγχουμε αν έφτασε στο επόμενο στόχο ελέγχουμε όλους τους επόμενους μέχρι τον τελικό.

Subgoal checking

Challenge 6 - Smart target selection

Στο κομμάτι αυτό υλοποιήθηκε η λύση που παρουσιάστηκε στη παρουσίαση 9 σελίδες 68, 99-80. Συγκεκριμένα, για κάθε node υπολογίζουμε τη διαδρομή του. Αν βρέθηκε διαδρομή μέχρι το στόχο υπολογίζουμε το κόστος απόστασης

$$w_{dist} = \sum_{i=1}^{PathSize-1} D_{i,i+1}$$

όπου $D_{i,i+1}$ η απόσταση του υποστόχου ***i*** από τον ***i+1***. Ακόμα υπολογίζουμε το τοπολογικό κόστος

$$w_{topo} = brush(node)$$

όπου *brush* η τιμή του brushfire στο σημείο του στόχου. Υπολογίζουμε το κόστος περιστροφής

$$w_{turn} = \sum_{i=1}^{PathSize} \Theta_i$$

όπου Θ_i η γωνία μεταξύ των διαδοχικών υποστόχων (μαζί με το αρχικό και τελικό σημείο). Τέλος, υπολογίζουμε το κόστος κάλυψης

$$w_{cove} = 1 - \frac{\sum_{i=1}^{PathSize} Coverage(x_i, y_i)}{PathSize \cdot 100}$$

όπου x_i , y_i τα σημεία των διαδοχικών υποστόχων.

Αν έχει βρεθεί κάποιο path όλα τα βάρη (για κάθε βάρος ξεχωριστά) τα κοινωνικοποιούμαι στο διάστημα [0,1] με τον τύπο

$$w^k = 1 - \frac{w^k - min(w)}{max(w) - min(w)}$$

Μετά δίνουμε σε κάθε βάρος διαφορετική βαρύτητα δηλαδή

$$w_{final} = 2^3 w_{topo} + 2^2 w_{dist} + 2^1 w_{cove} + 2^0 w_{turn}$$

και υπολογίζουμε το μέγιστο το οποί αντιστοιχεί στο στόχο που επιλέγουμε. Τέλος, ελέγχουμε αν ο στόχος που επιλέχτηκε είναι ίδιος με τον προηγούμενο επιλέγουμε κάποιο άλλο τυχαία, ώστε να μπορεί να ξεκολλήσει αν έχει κολλήσει κάπου.

SmartTargetSelect

selectTarget

Για χρησιμοποιηθεί αυτή η μέθοδος θα πρέπει να ορίσουμε τη τιμή του “target_selector: 'smart'” στο αρχείο “autonomous_expl.yaml”.

tartget_selector

Η τεχνική αυτή αν και αποτελεί καλύτερο τρόπο επιλογής στόχου έχει μεγάλο υπολογιστικό κόστος κυρίως λόγο του υπολογισμού του μονοπατιού για κάθε στόχο.

Extra Challenge 1 : Path optimization / alteration

Για βελτιώσουμε το μονοπάτι χρησιμοποιούμαι την τεχνική που παρουσιάστηκε στη παρουσίαση 8 σελίδα 138. Συγκεκριμένα, στη κλάση **Navigation** και μέθοδο **selectTarget** εφαρμόζουμε

$$y_i = y_i + a(x_i - y_i) + b(y_{i+1} - 2y_i + y_{i-1}),\ 1 \le i \le N-1$$

όπου $\boldsymbol{x}_i$ και $\boldsymbol{y}_i$ τα σημεία της διαδρομής ($\boldsymbol{x}_i$ η αρχική και $\boldsymbol{y}_i$ η τελική). Εφαρμόζουμε τον τύπο μέχρι

$$\sum_{i=1}^{N-1} a(x_i - y_i) + b(y_{i+1} - 2y_i + y_{i-1}) < 10^{-3}$$

Με αυτό τον τρόπο η διαδρομή γίνεται πιο ομαλή.

Path smoothing

Extra Challenge 2 : Algorithmic optimization

Utilities

Για τη βελτίωση του χρόνου επιλογής στόχου βελτιώθηκε ο υπάρχον κώδικας python. Αναλυτικότερα, στο “utilities.py” στη κλάση **Cffi** και μεθόδους **brushfireFromObstacles** , **thinning** και **prune** θέτουμε τις λίστες **brush** και **skeleton** ίση και στις τρεις μεθόδους με τη **y** αντί της ανάθεσης στις διπλές εμφωλεμένες λούπες. Έτσι πετυχαίνουμε 10 φορές πιο γρήγορους χρόνους σε κάθε περίπτωση.

BrushfireFromObstacles

thinning

prune

Στο ίδιο αρχείο και στη κλάση **OgmOperations** στη μέθοδο **blurUnoccupiedOgm** αντί για τετραπλή λούπα χρησιμοποιήθηκαν συνάρτησης της numpy και δισδιάστατη συνέλιξη από ώστε να πετύχουμε καλύτερο αποτέλεσμα. Με αυτό το τρόπο πετύχαμε 100 φορές γρηγορότερους χρόνους.

BlurUnoccupiedOgm

Για τη μέθοδο **findUsefulBoundaries** με τη χρήση συναρτήσεων της numpy αντί για διπλή εμφωλευμένη λούπα πετυχαίνουμε επιτάχυνση 2-10 φορές.

findUsefulBoundaries (min_x)

findUsefulBoundaries (max_x)

findUsefulBoundaries (min_y)

findUsefulBoundaries (max_y)

Topology

Στο αρχείο “topology.py” στη κλάση **Topology** και μεθόδους **skeletonizationCffi** και **skeletonization** με χρήση της numpy πετυχαίνουμε επιτάχυνση 500-1000 φορές.

skeletonizationCffi

skeletonization

Στη μέθοδο **topologicalNodes** αντί για εμφωλευμένες λούπες χρησιμοποιούμε μία πιο σωστή αλγοριθμικά υλοποίηση τους μεθόδους την numpy. Η επιτάχυνση που επιτυγχάνεται είναι τρίτης τάξης.

topologicalNodes

Brushfires

Στο αρχείο “brushfires.py” στη κλάση **Brushfires** και στη μέθοδο **obstaclesBrushfireCffi** με χρήση αντί για εμφωλευμένες λούπες χρησιμοποιήθηκε indexing με αποτέλεσμα επιτάχυνση 10-50 φορές.

ObstaclesBrushfiresCffi

Extra Challenge 3 : Surprize me

Σε αυτό το ερώτημα δε χρησιμοποιήθηκαν μέθοδοι την numpy ούτε υλοποιήθηκε κάτι πέρα από τα ζητούμενα των προηγούμενων ερωτημάτων.