# ΕΥΦΥΗ ΣΥΣΤΗΜΑΤΑ ΡΟΜΠΟΤ

Κιλάρογλου Ελευθέριος
ΑΕΜ: 8501

Αναφορά εργασίας

(Τα extra challenge δεν έγιναν λόγω έλλειψης χρόνου)

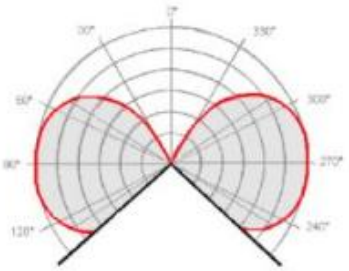
# Challenge 1: Laser-based obstacle avoidance

Θέλουμε το robot να κινείται στο χάρτη χωρίς να συγκρούεται σε εμπόδια. Έτσι εφόσον έχουμε lidar για μέτρηση αποστάσεων χρησιμοποιήσαμε τους παρακάτω τύπους:

*Obstacle avoidance – Rotational correction*

- Use the side rays

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

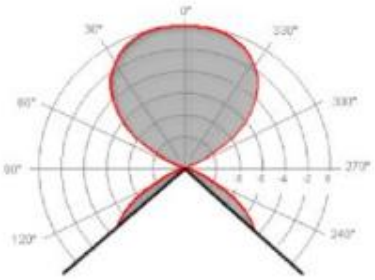




*Obstacle avoidance – Linear correction*

- What rays to use?
- Frontal mostly

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



Το αποτέλεσμα ήταν το robot να πηγαίνει μπρος-πίσω συνέχεια χωρίς ουσιαστικά να προχωράει. Το objective αποφυγής εμποδίων το πετύχαμε. Θέλουμε όμως ένα πλάνο ώστε το ρομπότ να κινείται.

$$u = u_{Path} + c_u \cdot u_{Obs} = u_{max} \cdot (1 - |\omega|)^2 - c_u \cdot \sum_{i=1}^{LaserRays} \frac{\cos(\theta_i)}{s_i^2}$$

$$\omega = \omega_{Path} + c_\omega \cdot \omega_{Obs} = \omega_{max} \cdot \omega - c_\omega \cdot \sum_{i=1}^{LaserRays} \frac{\sin(\theta_i)}{s_i^2}$$

Όταν έχουμε κάποιο πλάνο μια καλή υλοποίηση συνδυασμού ταχυτήτων είναι το παραπάνω motor schema. Εμείς επειδή θέλουμε απλό wandering επιλέξαμε $u_{path} = 1$ (απλώς να προχωράει το ρομπότ με μια σταθερή ταχύτητα) και παίξαμε με το $c_u$.

Κάνοντας κατάλληλα print και πειράματα ο τύπος $u = 1 + \frac{u_{obs}}{200}$ αποδείχθηκε αρκετά αποτελεσματικός. Το φυσικά δε χρειάστηκε να το πειράξουμε μας ενδιαφέρει το robot να περιστρέφεται μόνο με σκοπό την αποφυγή στόχων(και όχι ακολουθώντας κάποιο plan). Τέλος επειδή θέλουμε $u_{max} = 0.3\ m/s$ και $\omega_{max} = 0.3\ rad/s$ περιορίσαμε τις ταχύτητες στο διάστημα $[-0.3\ , 0.3]$ (δηλαδή αν π.χ. είχαμε $u > 0.3$ θέταμε $u = 0.3$.

# Challenge 2: Path visualization

Εδώ θέλουμε να εμφανίσουμε το path που δημιουργήσαμε (με βάση τις συντεταγμένες του robot) στο χάρτη. Για το βήμα αυτό το μόνο που χρειαζόμαστε είναι οι συντεταγμένες του path στο σύστημα του χάρτη, έστω A(ο υπόλοιπος κώδικας δίνεται έτοιμος). Οπότε έχουμε:

self.robot_perception.resolution = the resolution of the occupancy grid map

self.robot_perception.origin = the translation between the (0,0) of the robot pose and the (0,0) of the map

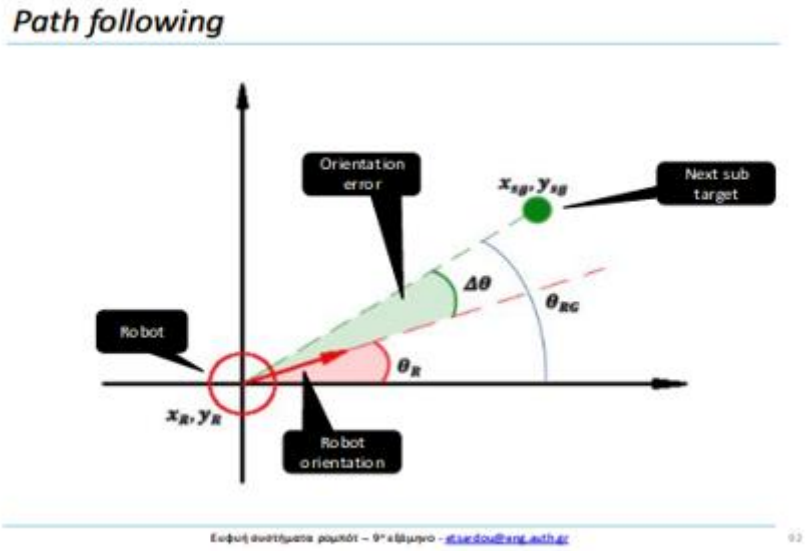
έχουμε επίσης τις συντεταγμένες p[0] (“x”) και p[1] (“y”) του δημιουργούμενου path με βάση το σύστημα του robot, έστω B. Ενώ στο σύστημα B έχουμε βήμα 1 στο σύστημα A το βήμα είναι self.robot_perception.resolution. Οπότε αν έχουμε προχωρήσει κατά x στο B, θα έχουμε προχωρήσει κατά x * self.robot_perception.resolution στο A. Επίσης το (0,0) στο σύστημα του robot μπορεί να μη συμπίπτει με το (0,0) του χάρτη. Οπότε χρειαζόμαστε μεταφορά του συστήματος συντεταγμένων.

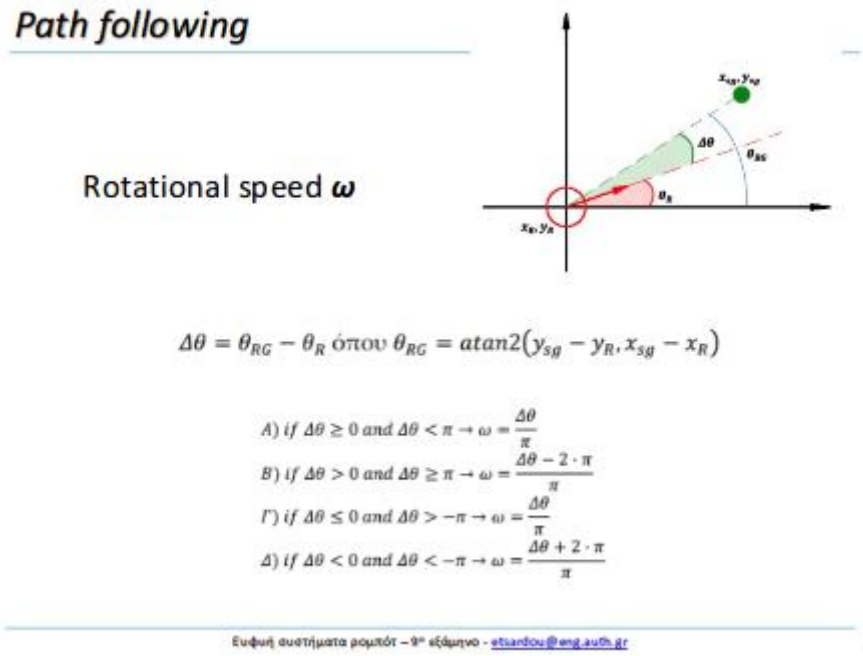
Έτσι τελικά θα έχουμε:

$$
\begin{aligned}
&\tau\varepsilon\tau\mu\eta\mu\acute{\varepsilon}\nu\eta\ A \\
&\quad = (\alpha\rho\chi\iota\kappa\acute{\eta}\ \theta\acute{\varepsilon}\sigma\eta\ x_0\tau o\upsilon\ robot) + (\beta\acute{\eta}\mu\alpha\ \kappa\alpha\tau\acute{\alpha}\ x) \\
&\quad = self.robot\_perception.origin['x'] \\
&\quad + self.robot\_perception.resolution \\
&\quad * (\tau\varepsilon\tau\mu\eta\mu\acute{\varepsilon}\nu\eta\ B)
\end{aligned}
$$

Τα ίδια ακριβώς ισχύουν και για την τεταγμένη y. Έτσι κάνοντας αυτή τη διαδικασία για κάθε σημείο (x,y) του path του συστήματος B, καταλήγουμε να έχουμε το path στο σύστημα A και τελικά να το προβάλουμε στο Rviz tool.

# Challenge 3 : Path following




Αρχικά εξετάσαμε την περίπτωση:

$$\omega_{path} = \omega * \omega_{max}$$

$$u_{path} = u_{max} * (1 - |\omega|)^n$$

(αυτά ξέρουμε αυτά εμπιστευόμαστε)

Προβλήματα που παρουσιάστηκαν:

i) Για μικρό Δθ το ω ήταν πολύ μικρό με αποτέλεσμα να μην προλαβαίνει να στρίψει το robot.

Αντιμετώπιση:

Μιας που δεν υπήρχε σοβαρός λόγος ώστε το robot να γυρίζει "αργά" επιλέξαμε την πιο απλή λύση στο πρόβλημα:

$$\omega_{path} = sgn(\Delta\theta) * \omega_{max}$$

ii) Για μικρά n η ταχύτητα ήταν πολύ μεγάλη
( $(1 - |\omega|) < 1$ άρα $(1 - |\omega|)^n$ φθίνουσα )
Οπότε και πάλι το robot δεν προλάβαινε να στρίψει κατάλληλα. Οπότε αυξήσαμε το n και κάνοντας πειράματα επιλέχθηκε η τιμή n = 10 (δεν υπήρχε κάποια ουσιαστική διαφορά για n = 9 ή n = 11.

Έτσι το robot είχε κατά μέσο όρο μικρότερη ταχύτητα και έτσι προλάβαινε να πάρει την επιθυμητή κλίση για το επόμενο subtarget με μικρό τίμημα φυσικά τον χρόνο εξερεύνησης. (μικρή ταχύτητα → μεγάλος χρόνος εξερεύνησης)

# Challenge 4 : Path following & obstacle avoidance

Στο σημείο αυτό επιλέξαμε μια υβριδική στρατηγική:

Όταν μας ενδιέφερε απλώς το wandering καταλήξαμε στον τύπο $u = 1 + \frac{u_{obs}}{200}$ . Έτσι όταν $u > 0$ το robot προχωρούσε μπροστά ενώ όταν $u < 0$ το robot πήγαινε πίσω με σκοπό να αποφύγει κάποιο εμπόδιο.

Οπότε όταν $u = 1 + \frac{u_{obs}}{200} > 0 : u = u_{path}$ και $\omega = \omega_{path}$

Έτσι το robot κινείται προς το στόχο χωρίς να το "νοιάζουν" τα εμπόδια. Ενώ όταν $u = 1 + \frac{u_{obs}}{200} < 0$ είμαστε κοντά στη σύγκρουση και έτσι αν δε συνδυάσουμε την ταχύτητα για obstacle avoidance το robot θα συγκρουστεί.

Εδώ χρησιμοποιήσαμε τους τύπους:

$$u = u_{Path} + c_u \cdot u_{Obs} = u_{max} \cdot (1 - |\omega|)^2 - c_u \cdot \sum_{i=1}^{LaserRays} \frac{\cos(\theta_i)}{s_i^2}$$

$$\omega = \omega_{Path} + c_\omega \cdot \omega_{Obs} = \omega_{max} \cdot \omega - c_\omega \cdot \sum_{i=1}^{LaserRays} \frac{\sin(\theta_i)}{s_i^2}$$

Με print των τιμών των $u_{obs}$ και $\omega_{obs}$ και πειράματα καταλήξαμε σε $c_u = c_\omega = 10^{-4}$. Τέλος περιορίσαμε και πάλι τις ταχύτητες στο διάστημα $[-0.3\ , 0.3]$.

# Challenge 5 : Smarter subgoal checking

Ως στιγμής κάθε φορά τσεκάραμε εάν το robot πλησίασε το επόμενο subtarget (next_subtarget) . Πλέον ελέγχουμε εάν το robot έχει πλησιάσει το next_subtarget και όλους τους επόμενους subtargets. Έτσι αν το robot πάει “κατά λάθος” στο μεθεπόμενο subtarget δε θα χρειαστεί να πάει σε αυτόν που δεν πήγε και θα συνεχίσει από εκεί την πορεία του.

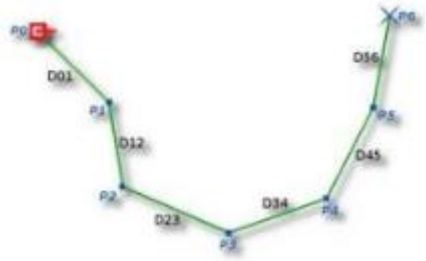
# Challenge 6 : Smart target selection

Εδώ χρησιμοποιήσαμε την cost based selection method. Έτσι για κάθε topological node μετρήσαμε τα ακόλουθα βάρη:

Topological cost:

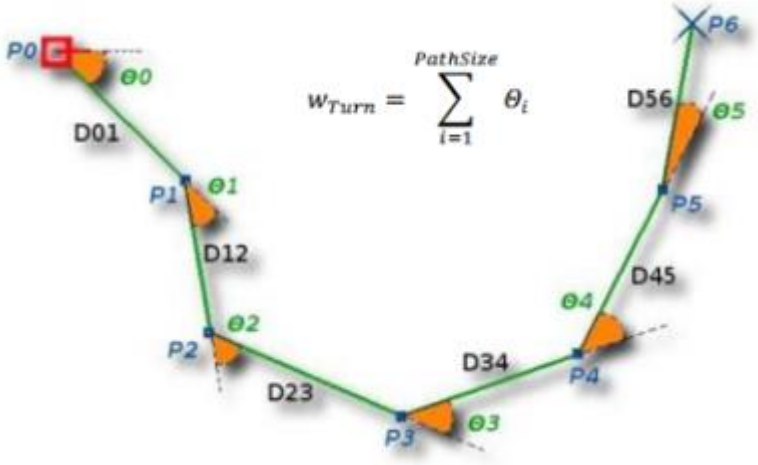
$$w_{Topo} = brush(node)$$

Distance cost:




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

Rotation cost:




$$w_{Turn} = \sum_{i=1}^{PathSize} \theta_i$$

Coverage cost :

$$w_{Cove} = \frac{\sum_{i=1}^{PathSize} Coverage[x_{P_i}, y_{P_i}]}{PathSize \cdot 255}$$

Και στη συνέχεια κάναμε normalization:

$$w^k_{Dist_n} = 1 - \frac{w^k_{Dist} - Min(w_{Dist})}{Max(w_{Dist}) - Min(w_{Dist})}$$

$$w^k_{Topo_n} = 1 - \frac{w^k_{Topo} - Min(w_{Topo})}{Max(w_{Topo}) - Min(w_{Topo})}$$

$$w^k_{Turn_n} = 1 - \frac{w^k_{Turn} - Min(w_{Turn})}{Max(w_{Turn}) - Min(w_{Turn})}$$

$$w^k_{Cove_n} = 1 - \frac{w^k_{Cove} - Min(w_{Cove})}{Max(w_{Cove}) - Min(w_{Cove})}$$

Τέλος υπολογίσαμε το τελικό βάρους του κάθε topological node με βάση τον τύπο:

$$w(k) = w_{Coeff}(k) \cdot P_{Pre}(k) = \frac{2^3 \cdot w^k_{1_n} + 2^2 \cdot w^k_{2_n} + 2^1 \cdot w^k_{3_n} + 2^0 \cdot w^k_{4_n}}{2^4 - 1} \cdot \left(2^3 \cdot \left\lfloor \frac{w^k_{1_n}}{w_{Th_1}} \right\rfloor + 2^2 \cdot \left\lfloor \frac{w^k_{2_n}}{w_{Th_2}} \right\rfloor + 2^1 \cdot \left\lfloor \frac{w^k_{3_n}}{w_{Th_3}} \right\rfloor + 2^0 \cdot \left\lfloor \frac{w^k_{4_n}}{w_{Th_4}} \right\rfloor\right)$$

(κάνοντας δηλαδή prioritization)

Έτσι επιλέγεται το topological node που έχει το μεγαλύτερο βάρος κάθε φορά. Εάν για οποιοδήποτε λόγο αποτύχει το target selection μέσω της cost based method, επιλέγουμε τον επόμενο στόχο τυχαία.