Autonomous Robotic Vehicles

Hellenic Mediterranean University

Lecture 13

Dr. Alina Eqtami

Lecture Goals - Στόχοι σημερινού μαθήματος!

Μετά το σημερινό μάθημα θα μπορείτε:

- να κατανοείτε τα βασικά στοιχεία της πλοήγησης ρομπότ,
- να διακρίνετε μεταξύ global και local planners,
- να καταλαβαίνετε τι είναι το C-space και γιατί το χρησιμοποιούμε,
- να αντιλαμβάνεστε πώς σχεδιάζουμε μονοπάτια και(στο επόμενο μάθημα) πώς αποφεύγουμε εμπόδια,
- να δείτε παραδείγματα εφαρμογής σε mobile robots και UAVs.

Navigation - Σχεδιασμός Πλοήγησης

- Η πλοήγηση είναι το γνωσιακό επίπεδο ενός mobile robot.
- Συνδυάζει στοιχεία από:
  - kinematics
  - sensing
  - localization
- Στόχος: να φτάσει το ρομπότ στο goal όσο πιο efficiently (αποτελεσματικά) και reliably (αξιόπιστα) γίνεται.

Δυο Βασικές Ικανότητες:

- 1. **Path Planning**
  - Στρατηγική ικανότητα.
  - Υπολογίζει μια trajectory ή path προς τον στόχο.
- 2. **Obstacle Avoidance**
  - Τακτική ικανότητα.
  - Προσαρμόζει την κίνηση σε πραγματικό χρόνο για αποφυγή collisions.

Planning vs Reacting

Σύνδεση με την Πλοήγηση

Η πλοήγηση βασίζεται σε δύο συμπληρωματικές ικανότητες:

- Planning → αντιστοιχεί στο **Path Planning** (στρατηγική επιλογή διαδρομής)
- Reacting → αντιστοιχεί στο **Obstacle Avoidance** (τακτική αντίδραση σε εμπόδια).

Συμπληρωματικότητα

- Το planning παρέχει το γενικό πλάνο κίνησης.
- Το reacting τροποποιεί την κίνηση σε πραγματικό χρόνο βάσει αισθητήρων.

- Χωρίς reacting → το σχέδιο αποτυγχάνει σε αληθινό περιβάλλον.
- Χωρίς planning → οι τοπικές αντιδράσεις δεν οδηγούν σε μακρινό στόχο.

Belief State

Το ρομπότ δεν γνωρίζει την πραγματική του θέση:

Λόγω θορύβου αισθητήρων, αβεβαιότητας στο μοντέλο κίνησης και ατελών χαρτών, το robot δεν γνωρίζει ποτέ με βεβαιότητα το που βρίσκεται.

- Χρησιμοποιεί αισθητήρες (IMU, LiDAR, camera, odometry) και έναν αλγόριθμο localization.
- Το αποτέλεσμα είναι μια **πιθανοτική εκτίμηση θέσης**:

$$b = p(x \mid \text{sensors, map})$$

- Αυτή η εκτίμηση ονομάζεται **belief state**.

Πλοήγηση ως μετάβαση μεταξύ belief states

- Ο στόχος δεν είναι απλώς "φτάνω στο σημείο p".
- Ο στόχος είναι να φτάσω σε **belief state** που αντιστοιχεί στο "βρίσκομαι στο p με μεγάλη βεβαιότητα".

$$b_i \longrightarrow b_g$$

- Το planning σχεδιάζει πορεία που μεταφέρει το robot από το αρχικό belief state b_i στο ζητούμενο belief state b_g.
- Αυτό είναι απαραίτητο επειδή:
  - η εκτίμηση θέσης έχει θόρυβο,
  - ο χάρτης μπορεί να μην είναι τέλειος,
  - το περιβάλλον μπορεί να αλλάζει.

Πρακτικά:

Το navigation είναι μετάβαση μεταξύ πιθανοτικών καταστάσεων, όχι μόνο μεταξύ σημείων στο χώρο.

Completeness - Έννοια Πληρότητας

Ορισμός

Ένα navigation σύστημα είναι **complete** αν:

- όποτε υπάρχει λύση (feasible trajectory),
- το σύστημα θα τη βρει και θα φτάσει στο goal.

- Αν είναι **incomplete**, τότε υπάρχει τουλάχιστον ένα πρόβλημα με λύση που δεν θα βρεθεί.
- Το completeness συχνά θυσιάζεται για υπολογιστική ταχύτητα.

Πρακτική Ερμηνεία:

- **Complete planner**: "Αν υπάρχει δρόμος, θα τον βρω."
- **Incomplete planner**: "Θα βρω γρήγορα λύση, αλλά ίσως χάσω κάποιες."

Navigation Architecture - Αρχιτεκτονική Πλοήγησης

- **1. Global Planner**
  - Υπολογίζει reference path.
  - Αλγόριθμοι: A*, D*, PRM, RRT.
- **2. Local Planner**
  - Μετατρέπει path σε **time-parameterized trajectory**.
  - Κάνει obstacle avoidance.
  - Αλγόριθμοι: DWA, TEB, MPC-based planning.
- **3. Controller**
  - Παρακολουθεί το trajectory.
  - Παράγει roll, pitch, yaw, thrust.

Quadcopter Example – 1: Σχεδιασμός Πλοήγησης

- **Σενάριο**: Το quadcopter ξεκινά από $A = (0, 0, 0.5)$ και πρέπει να φτάσει στο $B = (10, 8, 2)$ μέσα σε αποθήκη με ράφια ύψους $1.5m$.
- **1. Global Planner (π.χ. RRT*)**
  - Αναγνωρίζει ράφι μεταξύ $x = 4$–6 και ύψους $1.5m$.
  - Υπολογίζει path που το αποφεύγει:

$$A \to (3, 0, 1.5) \to (6, 4, 2) \to B$$

  - Το path είναι **γεωμετρικό** και όχι εκτελέσιμο ακόμα.
- **2. Local Planner (π.χ. MPC)**
  - Μετατρέπει το path σε **χρονοπαραμετροποιημένη τροχιά**.
  - Λαμβάνει υπόψη:
    - μέγιστη ταχύτητα $1.2m/s$,
    - μέγιστη κλίση 15°,
    - real-time αισθητήρες.
  - Παράγει trajectory διάρκειας $12s$:

$$p(t) = (x(t), y(t), z(t)),\ 0 \le t \le 12$$

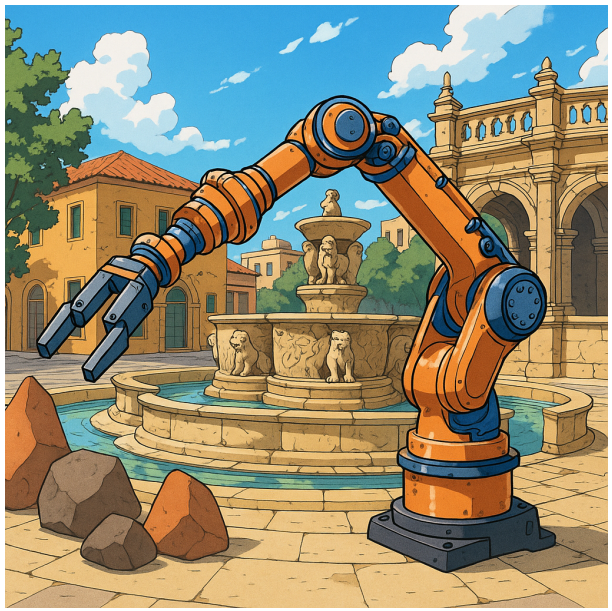
Quadcopter Example – 2: Εκτέλεση Τροχιάς

- **3. Controller (Position & Attitude Control)**
  - Παρακολουθεί την επιθυμητή τροχιά $p(t)$ από τον local planner.
  - Το control γίνεται σε 3 επίπεδα:
    - **Position control**: Υπολογίζει επιθυμητή επιτάχυνση.
    - **Attitude control**: Μετατρέπει επιτάχυνση σε roll, pitch, yaw.
    - **Motor mixer**: Παράγει εντολές στους 4 έλικες.

- **Εκτέλεση σε πραγματικό χρόνο**
  - Το quadcopter διορθώνει αποκλίσεις χρησιμοποιώντας IMU + VIO.
  - Αν εμφανιστεί νέο εμπόδιο, ο local planner αναπροσαρμόζει την τροχιά.
  - Το σύστημα Navigation δουλεύει συνεχώς
    loop: sensing → planning → control.

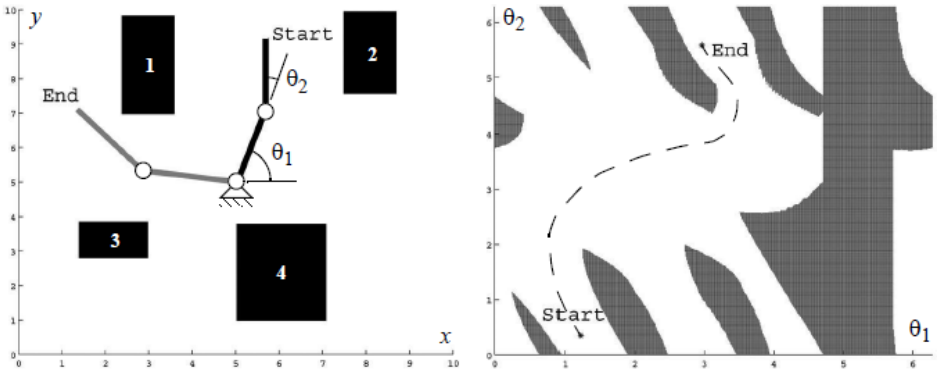
Path Planning Origins - Αφετηρία της Πλοήγησης

- Το path planning μελετήθηκε εντατικά αρχικά στη βιομηχανική ρομποτική (manipulators).
- Ένα manipulator με 6 DOF έχει πολύ πιο δύσκολο πρόβλημα πλοήγησης από ένα mobile robot.
- Ο λόγος: υψηλή διάσταση, πολύπλοκη γεωμετρία, και η δυναμική του συστήματος επηρεάζει το planning.
- Σε σύγκριση, πολλά mobile robots κινούνται αργά $\rightarrow$ η δυναμική αγνοείται.



Configuration Space - Χώρος Διαμόρφωσης (C-space)

- Για ένα manipulator με k DOF, κάθε θέση περιγράφεται από:
$$(q_1, q_2, \ldots, q_k)$$
- Το σύνολο όλων των τέτοιων σημείων είναι ο **χώρος διαμόρφωσης**.
- Τα εμπόδια του φυσικού χώρου μετατρέπονται σε **περιοχές απαγορευμένων διαμορφώσεων**.
- Το path planning γίνεται στο free C-space.



(α) Φυσικός χώρος: Ένας επίπεδος βραχίονας πρέπει να μετακινηθεί από το start στο end, αποφεύγοντας τα εμπόδια 1–4.
(β) Το αντίστοιχο configuration space δείχνει τον ελεύθερο χώρο στις γωνίες θ_1 και θ_2, καθώς και ένα μονοπάτι που επιτυγχάνει τον στόχο.

Πηγή: Siegwart, Nourbakhsh & Scaramuzza — *Introduction to Autonomous Mobile*

Mobile Robot C-space - Απλοποίηση για Κινητά Επίγεια Ρομπότ

- Τα mobile robots έχουν πολύ μικρότερο DOF (συνήθως x, y, θ).
- Στο path planning:
  - θεωρούμε τα περισσότερα robots **holonomic** (παρότι δεν είναι),
  - αγνοούμε το θ όταν δεν είναι κρίσιμο,
  - μοντελοποιούμε το robot ως **σημείο**,
  - **φουσκώνουμε (inflate)** τα εμπόδια κατά την ακτίνα του robot.
- Έτσι το C-space γίνεται ουσιαστικά 2D και ταυτίζεται με τον φυσικό χώρο.

Unicycle Workspace - Μετάβαση από τον φυσικό χώρο στο C-space – 1

- Ένα unicycle mobile robot περιγράφεται από την κατάσταση:

$$q = (x, y, \theta)$$

- Το θ (προσανατολισμός) δεν επηρεάζει τη δυνατότητα προσέγγισης ενός σημείου, διότι το unicycle μπορεί να περιστραφεί σχεδόν επιτόπου.
- Για το path planning, εφαρμόζουμε προβολή:

$$\pi : \mathbb{R}^3 \to \mathbb{R}^2, \qquad \pi(x, y, \theta) = (x, y)$$

- Άρα ο φυσικός χώρος (x, y, θ) γίνεται επίπεδος **2D** χώρος για σχεδιασμό τροχιάς.

Unicycle C-space - Μετάβαση από τον φυσικό χώρο στο C-space – 2

- Το robot στον φυσικό χώρο έχει ακτίνα r:

$$R = \{(x, y) \mid (x - x_r)^2 + (y - y_r)^2 \leq r^2\}$$

- Στο C-space το robot γίνεται σημείο.
- Τα εμπόδια "διογκώνονται" κατά r (λόγω Minkowski sum):

$$O_{\text{inflated}} = O \oplus B(0, r)$$

όπου

$$B(0, r) = \{(x, y) \mid x^2 + y^2 \leq r^2\}$$

- Έτσι το path planning γίνεται σε:

$$C_{\text{free}} = \mathbb{R}^2 \setminus O_{\text{inflated}}$$

Unicycle Example - Παράδειγμα Μετάβασης στο C-space

- Έστω ότι ένα unicycle έχει ακτίνα $r = 0.3m$.
- Στον φυσικό χώρο υπάρχει κυκλικό εμπόδιο:

$$O = \{(x, y) \mid (x-2)^2 + (y-3)^2 \leq 1^2\}$$

- Στο C-space το εμπόδιο διογκώνεται:

$$O_{\text{inflated}} = \{(x, y) \mid (x-2)^2 + (y-3)^2 \leq (1+0.3)^2\}$$

- Το robot ξεκινά από $A = (0, 0)$ και πρέπει να φτάσει στο $B = (4, 5)$.
- Το path planning βρίσκει τροχιά που παρακάμπτει το O_{inflated} στο επίπεδο (x, y).

UAV Configuration Space - Μαθηματική Περιγραφή

- Ένα UAV μοντελοποιείται κινηματικά με:

$$q = (x, y, z, \phi, \theta, \psi)$$

- Στο global path planning αγνοούμε τον προσανατολισμό (ϕ, θ, ψ) διότι:
  - δεν επηρεάζει την επίτευξη θέσης,
  - ο έλεγχος στάσης (attitude control) το ρυθμίζει ανεξάρτητα.
- Εφαρμόζουμε προβολή:

$$\pi : \mathbb{R}^6 \to \mathbb{R}^3, \qquad \pi(x, y, z, \phi, \theta, \psi) = (x, y, z)$$

- Το UAV έχει ακτίνα ασφαλείας r (ενεργό μέγεθος στο χώρο).
- Τα εμπόδια O στον φυσικό χώρο μετατρέπονται σε «διογκωμένα»:

$$O_{\text{inflated}} = O \oplus B(0, r)$$

όπου $B(0, r)$ είναι η μπάλα ακτίνας r.

- Έτσι ο χώρος ελεύθερης κίνησης είναι:

$$C_{\text{free}} = \mathbb{R}^3 \setminus O_{\text{inflated}}$$

UAV Example - Παράδειγμα Μετάβασης στο C-space

- Έστω UAV με ακτίνα ασφαλείας $r = 0.4m$.
- Στον φυσικό χώρο υπάρχει κυβικό εμπόδιο:

$$O = [2, 4] \times [1, 3] \times [0, 2]$$

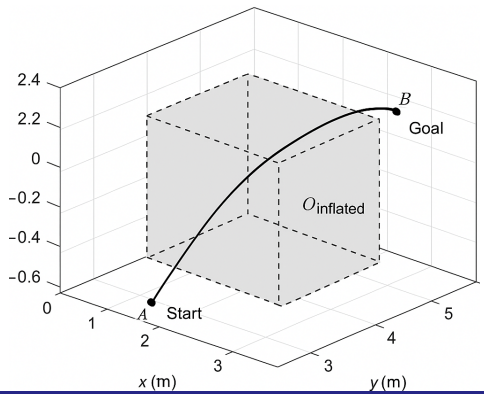
- Στο C-space το εμπόδιο διογκώνεται:

$$O_{\text{inflated}} = [2 - r, 4 + r] \times [1 - r, 3 + r] \times [0 - r, 2 + r]$$

δηλ.

$$O_{\text{inflated}} = [1.6, 4.4] \times [0.6, 3.4] \times [-0.4, 2.4]$$

- Το UAV ξεκινά από $A = (0, 0, 1)$ και στοχεύει στο $B = (6, 5, 2)$.
- Το path planning στο 3D C_{free} επιλέγει μονοπάτι που παρακάμπτει τον διογκωμένο όγκο.



Path Planning Overview

- Το path planning ξεκινά από μια αναπαράσταση του περιβάλλοντος (συνεχής ή διακριτή γεωμετρία, topological map, occupancy grid, κ.λπ.).
- Πρώτο βήμα: μετατροπή του περιβάλλοντος σε **διακριτό χάρτη** κατάλληλο για τον αλγόριθμο που θα χρησιμοποιηθεί.
- Υπάρχουν δύο κύριες στρατηγικές σχεδιασμού:
  - **Graph search**: κατασκευή γράφου στο free space και αναζήτηση βέλτιστης διαδρομής.
  - **Potential field planning**: ορισμός μαθηματικού πεδίου και κίνηση προς το goal με βάση το gradient.
- Στο graph search θα εξετάσουμε:
  - μεθόδους **graph construction** (visibility graph, Voronoi, cell decomposition, lattice),
  - μεθόδους **graph search** (Dijkstra, A*, D*).
- Στόχος: κατανόηση τρόπου δόμησης του χώρου, επιλογής γράφου, και εύρεσης optimal ή feasible διαδρομής.

Visibility Graph - Σύνοψη και Ιδέα

- Ο visibility graph κατασκευάζεται συνδέοντας όλα τα ζεύγη κορυφών των εμποδίων (και τα σημεία start–goal) που **βλέπουν** το ένα το άλλο, δηλαδή χωρίς εμπόδιο ανάμεσά τους.
- Οι ακμές είναι ευθείες γραμμές στο free space και αποτελούν τις **συντομότερες δυνατές** διαδρομές μεταξύ των κορυφών.
- Ο στόχος του planner: εύρεση συντομότερου δρόμου στο visibility graph.

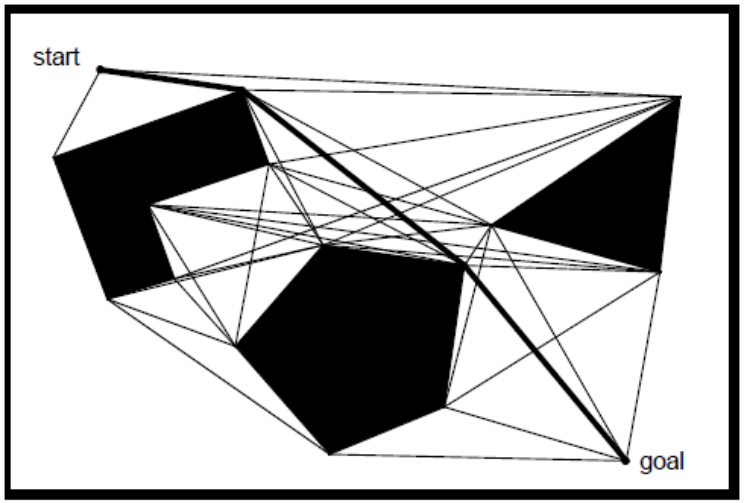


Figure: Visibility graph. Οι κόμβοι είναι το start, goal και οι κορυφές των πολυγωνικών εμποδίων.

Πηγή: Siegwart, Nourbakhsh & Scaramuzza, *Introduction to Autonomous Mobile*

Visibility Graph - Πλεονεκτήματα και Προβλήματα

- **Πλεονεκτήματα**
  - Εύκολη υλοποίηση όταν το περιβάλλον περιγράφεται με πολυγωνικά εμπόδια.
  - Οι λύσεις είναι **βέλτιστες ως προς το μήκος διαδρομής**.
  - Χρήσιμη προσέγγιση για χαμηλής πυκνότητας χάρτες.

- **Μειονεκτήματα**
  - Το μέγεθος του γράφου (Νο κόμβων, ακμών) αυξάνει απότομα με τον αριθμό των πολυγωνικών εμποδίων.
  - Σε πυκνά περιβάλλοντα γίνεται **αργός** και υπολογιστικά ακριβός.
  - Οι βέλτιστες διαδρομές περνούν **πολύ κοντά στα εμπόδια**:

$$\text{optimality in length} \Rightarrow \text{zero safety margin}$$

  - Συνήθης λύση: inflation των εμποδίων ή post-processing του μονοπατιού για αυξημένη απόσταση από τα εμπόδια.

Voronoi Diagram - Σύνοψη και Ιδέα

- Το Voronoi diagram δημιουργεί ένα οδικό δίκτυο στο free space που **μεγιστοποιεί την απόσταση** του ρομπότ από τα εμπόδια.
- Για κάθε σημείο στο χώρο υπολογίζεται η απόσταση από το κοντινότερο εμπόδιο.
- Σημεία που είναι **ισαπέχοντα** από δύο ή περισσότερα εμπόδια σχηματίζουν **κορυφογραμμές** — οι οποίες αποτελούν τις ακμές του Voronoi.
- Αν τα εμπόδια είναι πολυγωνικά, οι ακμές αποτελούνται από:

  ευθύγραμμα τμήματα και παραβολικά τόξα

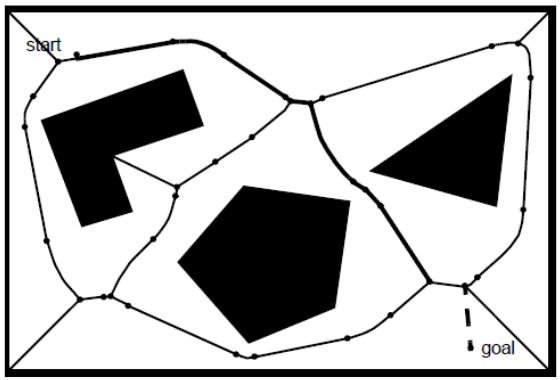


Figure: Voronoi diagram. **Πηγή:** Siegwart, Nourbakhsh & Scaramuzza — *Introduction to Autonomous Mobile Robots*.

Voronoi Diagram - Πλεονεκτήματα, Αδυναμίες και Εκτελεσιμότητα

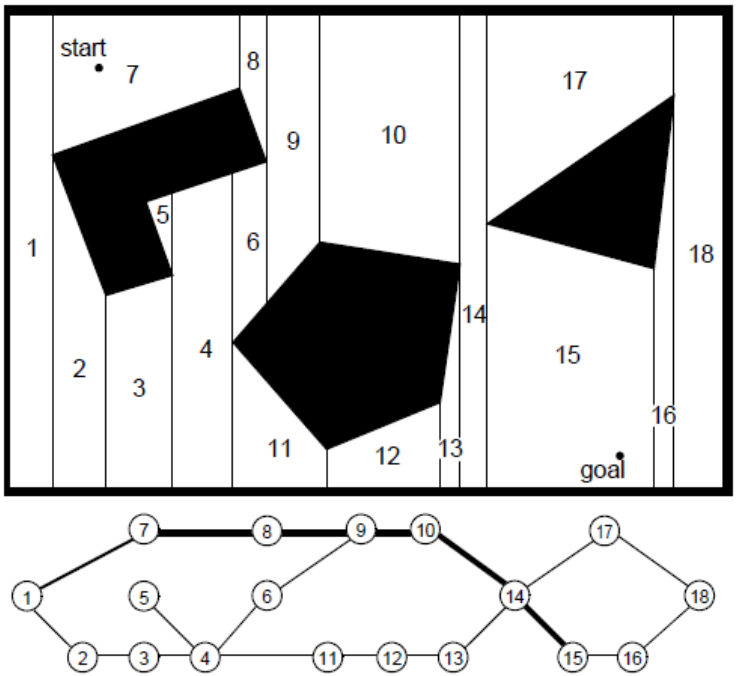
- **Πλεονεκτήματα**
  - **Μεγιστοποιεί την απόσταση** από τα εμπόδια — ασφαλής πλοήγηση.
  - **Complete** μέθοδος: αν υπάρχει μονοπάτι στο free space, υπάρχει και στο Voronoi.
  - Σαφής γεωμετρική ερμηνεία (κορυφογραμμές ισαπέχοντων σημείων).
- **Αδυναμίες**
  - Οι διαδρομές είναι συνήθως **μη βέλτιστες** ως προς το μήκος.
  - Αν το ρομπότ χρησιμοποιεί **short-range localization sensors**, η μεγάλη απόσταση από τα εμπόδια μπορεί να δημιουργήσει κακή ποιότητα localization.
- **Εκτελεσιμότητα στην πράξη**
  - Ρομπότ με range sensors μπορούν να ακολουθήσουν Voronoi edges φυσικά, μεγιστοποιώντας τις μετρήσεις από δύο αντικείμενα.
  - Αυτό επιτρέπει ακόμη και **αυτόματη εξερεύνηση**: το ρομπότ «ανιχνεύει» και ακολουθεί άγνωστες Voronoi ακμές.

Exact Cell Decomposition - Έννοια και Ιδιότητες

- Η μέθοδος χωρίζει το περιβάλλον σε **κυψέλες** (cells) των οποίων τα σύνορα ορίζονται από **γεωμετρικά κρίσιμα σημεία**.
- Κάθε cell είναι:

  είτε πλήρως ελεύθερο είτε πλήρως κατειλημμένο

- Η αναπαράσταση είναι **lossless** — δεν χάνεται καμία πληροφορία του free space.
- Η ακριβής γεωμετρία των εμποδίων διατηρείται πλήρως.



Exact Cell Decomposition - Πλεονεκτήματα και Μειονεκτήματα

- **Πλεονεκτήματα**
  - Απόλυτη ακρίβεια (no approximation).
  - Πλήρης (complete) αναζήτηση διαδρομής.
  - Εξαιρετικά αποδοτική σε **αραιά** περιβάλλοντα (λίγα εμπόδια $\rightarrow$ λίγα cells).
- **Μειονεκτήματα**
  - Ο αριθμός των cells εξαρτάται από την **πολυπλοκότητα** του χώρου.
  - Πολύ μεγάλος υπολογιστικός φόρτος σε περίπλοκα περιβάλλοντα.
  - Σπάνια χρησιμοποιείται στην πράξη λόγω δυσκολίας υλοποίησης.

Approximate Cell Decomposition - Βασική Ιδέα

- Το περιβάλλον διαιρείται σε cells **σταθερού** ή **μεταβαλλόμενου** μεγέθους.
- **Fixed-size decomposition**
  - Σταθερό grid (π.χ. occupancy grid).
  - Πολύ δημοφιλές — απλό, γρήγορο, εύκολο στην υλοποίηση.
  - Μπορεί να «χάσει» στενά περάσματα (λόγω resolution).
- **Variable-size decomposition**
  - Προσαρμόζεται στην πολυπλοκότητα του χώρου.
  - Λιγότερη μνήμη σε αραιά περιβάλλοντα.
- **Μειονέκτημα**:

  Όχι πάντα complete (χάνεται λεπτομέρεια).

- **Πλεονέκτημα**:

  Πολύ χαμηλή υπολογιστική πολυπλοκότητα.

Exact vs Approximate - Συζήτηση

- **Exact cell decomposition**
  - Lossless — πλήρης γεωμετρική πληροφορία.
  - Complete σε όλες τις περιπτώσεις.
  - Αργό και δύσκολο σε περίπλοκα περιβάλλοντα.
- **Approximate cell decomposition**
  - Γρήγορο, απλό, πολύ πρακτικό.
  - Χαμηλή μνήμη (ειδικά το variable-size).
  - Μπορεί να χάσει λεπτομέρειες $\rightarrow$ όχι πάντα complete.
- **Trade-off:**

$$\text{Accuracy (Exact)} \leftrightarrow \text{Efficiency (Approximate)}$$

Graph Search Overview

- Αφού κατασκευάσουμε τον γράφο με τις **graph construction methods** (visibility graph, Voronoi, exact/approximate cell decomposition), περνάμε στο επόμενο κρίσιμο στάδιο:

  graph search algorithms

- Στόχος:

  εύρεση βέλτιστης ή εφικτής διαδρομής (path)

  από start σε goal στον connectivity graph.

- Όλοι οι αλγόριθμοι αναζήτησης χρησιμοποιούν κοινές έννοιες κόστους, αλλά διαφοροποιούνται στο πώς τα συνδυάζουν.

Graph Search Cost Functions - Βασικά Κόστη και Αλγόριθμοι

- Εισάγουμε τα βασικά κόστη για οποιονδήποτε αλγόριθμο αναζήτησης:
  - $g(n)$ — path cost: συσσωρευμένο κόστος από το start μέχρι το n.
  - $c(n, n')$ — edge cost: κόστος μετάβασης από n στο n'.
  - $h(n)$ — heuristic cost: εκτίμηση έως το goal.
  - $f(n)$ — **συνολικό αναμενόμενο κόστος**:

$$f(n) = g(n) + \alpha\, h(n)$$

- Με διαφορετικές επιλογές $h(n)$ και α προκύπτουν:
  - **BFS/DFS**: ίσα edge costs.
  - **Dijkstra**: $f(n) = g(n)$.
  - **A***: $f(n) = g(n) + h(n)$ (admissible/consistent h).
  - **Greedy A***: $f(n) = h(n)$ (γρήγορο, μη βέλτιστο).

Cost Functions in Practice - Τι σημαίνουν πρακτικά τα κόστη

- Graph search σημαίνει ότι κάθε κόμβος έχει ένα **κόστος** που λέει πόσο «ακριβό» είναι το να φτάσουμε εκεί.
- $c(n, n')$ — **Edge Cost**
  - Το κόστος για να πάω **από το** n **στο** n'.
  - Π.χ. μήκος ευθύγραμμου τμήματος, ενέργεια, χρόνος, κίνδυνος.
- $g(n)$ — **Path Cost (Accumulated Cost)**
  - Το κόστος του **μονοπατιού μέχρι τον κόμβο** n.
  - Υπολογίζεται αθροιστικά:

$$g(n) = \sum c(i, j)$$

  - Δηλαδή πόσο μας έχει «κοστίσει» να φτάσουμε μέχρι εδώ.
- $h(n)$ — **Heuristic Cost-to-Go**
  - Εκτίμηση του κόστους για να φτάσουμε από το n στο **goal**.
  - Δεν είναι ακριβές — όμως πρέπει να είναι «αισιόδοξο» (να μην υπερεκτιμά).

Heuristic Function $h(n)$ - Πρακτικός Υπολογισμός (2D/3D)

Στόχος του $h(n)$: Να εκτιμήσει «πόσο μακριά» είμαστε από το goal **χωρίς να λάβει υπόψη εμπόδια**, ώστε να παραμένει **admissible** (δεν υπερεκτιμά) και **consistent**.

- 1. 2D Mobile Robots (Grid-Based Planning)
  - **Manhattan distance** (4-γειτονικές κινήσεις):
$$h(n) = |x_n - x_g| + |y_n - y_g|$$
  - **Diagonal distance** (8-γειτονικές κινήσεις):
$$h(n) = \max(|x_n - x_g|,\ |y_n - y_g|)$$
    Λειτουργεί καλύτερα όταν επιτρέπονται διαγώνιες κινήσεις.
  - **Euclidean distance** (συνεχής κίνηση, πραγματική γεωμετρική απόσταση):
$$h(n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2}$$
- 2. 3D Robots / UAVs
$$h(n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2 + (z_n - z_g)^2}$$

Heuristic Function $h(n)$ - Πρακτικός Υπολογισμός (Unicycle/Manipulators)

Για πιο σύνθετα ρομποτικά μοντέλα, το heuristic πρέπει να είναι απλό, γρήγορο και να μην παραβιάζει την προϋπόθεση της υποεκτίμησης.

- **1. Unicycle / Car-like Robots**

$$h(n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2}$$

  - Αγνοούμε το θ γιατί θα οδηγούσε σε ανακριβή (μη-admissible) εκτίμηση.
  - Το orientation λαμβάνεται υπόψη από το $g(n)$ κατά τη διάρκεια της αναζήτησης.

- **2. Manipulators (Configuration Space Planning)**

$$h(n) = \|q_n - q_g\|$$

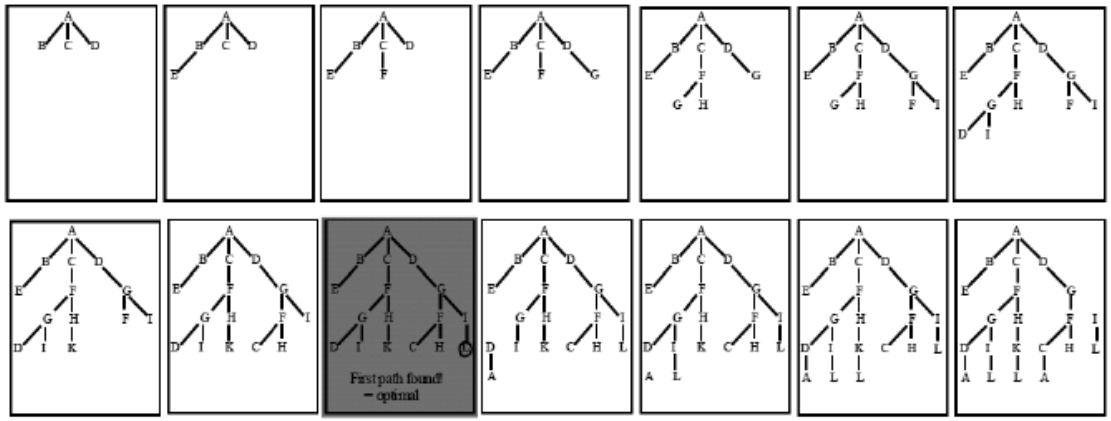
όπου q είναι το διάνυσμα γωνιών άρθρωσης.

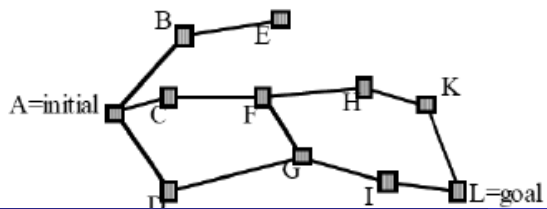
Breadth-First Search (BFS) - Βασική Ιδέα

- Το BFS ξεκινά από το **start node** και εξερευνά όλους τους **γείτονες** του.
- Έπειτα εξερευνά τους γείτονες των γειτόνων κ.ο.κ.
- Η διαδικασία αυτή λέγεται **node expansion**.
- Η αναζήτηση γίνεται **επίπεδο–επίπεδο**:

  πρώτα όλα τα nodes σε απόσταση 1, μετά σε απόσταση 2, ...

- Τερματίζει μόλις φτάσει στο **goal**.





Breadth-First Search (BFS) - Ιδιότητες και Βέλτιστη Λύση

- Επιστρέφει πάντα το **μονοπάτι με τα λιγότερα** edges.
- Άρα, αν **όλα τα** edges **έχουν ίδιο κόστος**:

BFS είναι optimal

- Αν όμως τα edges έχουν **διαφορετικά κόστη**, τότε:

BFS δεν εγγυάται minimum-cost path

- Το BFS είναι **πολύ γρήγορο**: κάθε κόμβος επισκέπτεται το πολύ μία φορά.
- Πολυπλοκότητα:

$$O(\text{number of nodes})$$

Time Complexity - Πρακτική Ερμηνεία του $O(\cdot)$

Τι είναι το $O(\cdot)$;

- Είναι ένας τρόπος να περιγράψουμε **πόσο χειρότερη γίνεται η απόδοση** ενός αλγορίθμου όταν αυξάνεται το μέγεθος των δεδομένων.
- Δεν μετράει «πόσα δευτερόλεπτα» κάνει ένας αλγόριθμος.
- Μετράει **πώς αλλάζει ο χρόνος όταν το πρόβλημα μεγαλώνει**.

Πρακτικά παραδείγματα:

- $O(N)$ Κάνεις μια εργασία που απαιτεί να «κοιτάξεις» κάθε στοιχείο μία φορά. Αν τα δεδομένα διπλασιαστούν $\rightarrow$ ο χρόνος διπλασιάζεται.
- $O(N \log N)$ Κάθε φορά που διπλασιάζονται τα δεδομένα, ο χρόνος αυξάνεται λίγο πιο γρήγορα από γραμμικά — π.χ. όπως όταν ταξινομείς πράγματα με έξυπνο τρόπο.
- $O(N^2)$ Για κάθε στοιχείο πρέπει να συγκρίνεις όλα τα υπόλοιπα. Αν τα δεδομένα διπλασιαστούν $\rightarrow$ ο χρόνος γίνεται περίπου **4 φορές** μεγαλύτερος.

Wavefront Expansion (NF1) - BFS σε Grid

- Το wavefront expansion είναι υλοποίηση του BFS πάνω σε occupancy grids.
- Η αναζήτηση ξεκινά από το **goal** και εξαπλώνεται σαν «κύμα» προς όλες τις κατευθύνσεις.
- Σε κάθε κελί ανατίθεται η **Manhattan απόσταση** από το goal:

$$d = |x - x_g| + |y - y_g|$$

- Η Manhattan απόσταση μετρά πόσα **4-directional βήματα** (πάνω/κάτω/αριστερά/δεξιά) χρειάζεται το ρομπότ για να φτάσει στο goal — όπως στο Μανχάταν με δρόμους σε «τετράγωνα».
- Το μονοπάτι προκύπτει ακολουθώντας κελιά που έχουν **μειούμενη** τιμή απόστασης.
- Το σύστημα είναι εξαιρετικά αποδοτικό:

$$O(\text{number of cells})$$

Depth-First Search (DFS) - Βασική Ιδέα

- Το DFS επεκτείνει κάθε node όσο πιο βαθιά γίνεται, μέχρι να φτάσει σε κόμβο χωρίς successors.
- Όταν μια διαδρομή τερματίσει, η αναζήτηση **backtracks** και συνεχίζει με τον επόμενο διαθέσιμο γείτονα.
- Μπορεί να επισκεφτεί ξανά nodes ή να εισέλθει σε «άχρηστα» μονοπάτια, αλλά αυτά αποφεύγονται εύκολα με κατάλληλη υλοποίηση.
- **Πλεονέκτημα**: Πολύ χαμηλή κατανάλωση μνήμης.
- Το DFS χρειάζεται να αποθηκεύσει μόνο:
  - το τρέχον μονοπάτι από start $\rightarrow$ current node,
  - και τους μη εξερευνημένους γείτονες για κάθε node στο path.
- Μόλις ένα node εξερευνηθεί πλήρως, μπορεί να διαγραφεί από τη μνήμη.

Dijkstra's Algorithm - Βασική Ιδέα και Μηχανισμός

- Ο αλγόριθμος Dijkstra βρίσκει το μονοπάτι **ελάχιστου κόστους** μεταξύ start και goal.
- Μοιάζει με BFS, αλλά επιτρέπει **οποιοδήποτε θετικό** edge cost.
- Εγγυάται **optimal λύση** ακόμη και σε μη-ομοιόμορφους γράφους.
- Χρησιμοποιεί μια δομή δεδομένων **heap (priority queue)**:
  - κάθε node στο heap έχει key το $f(n) = g(n)$,
  - αναδιατάσσεται ώστε στην κορυφή να είναι πάντα το node με το μικρότερο κόστος.
- Διαδικασία:
  1. Ξεκινά από το start node.
  2. Επεκτείνει τον φθηνότερο node στο heap.
  3. Βάζει τους γείτονες στο heap με ενημερωμένο κόστος.
  4. Συνεχίζει μέχρι να επεκταθεί το goal.

Dijkstra's Algorithm - Ιδιότητες, Πολυπλοκότητα και Χρήση στη Ρομποτική

- **Βέλτιστος**:

$$f(n) = g(n),\ h(n) = 0 \Rightarrow \text{βρίσκει το minimum-cost path}$$

- **Πολυπλοκότητα**: $O(m + n \log n)$ όπου:
  - n = αριθμός κόμβων και m = αριθμός ακμών.
- **Γιατί χρησιμοποιείται στη ρομποτική**:
  - Συνήθως εκτελείται από το goal **προς όλες τις κατευθύνσεις**.
  - Έτσι προκύπτει ένας **χάρτης κόστους** (cost-to-go) για κάθε πιθανή θέση.
  - Το robot μπορεί να ξεκινήσει από οπουδήποτε και να ξέρει:

    ποια είναι η βέλτιστη κατεύθυνση προς το goal

  - Το path μπορεί να ανανεώνεται κατά την κίνηση, χωρίς πλήρες replanning.
- Εξαιρετικά χρήσιμο σε παρουσία **θορύβου αισθητήρων και πράξης**, αφού το robot μπορεί να «ξαναβρίσκει» την πορεία του.

A* Algorithm - Βασική Ιδέα και Μηχανισμός

- Ο αλγόριθμος A* συνδυάζει:

$$f(n) = g(n) + h(n)$$

όπου:

  - $g(n)$ = κόστος διαδρομής μέχρι το node n
  - $h(n)$ = εκτίμηση κόστους από n μέχρι το goal (heuristic)

- Για να εγγυηθεί **optimal λύση**, το heuristic πρέπει να είναι:
  - **admissible**: δεν υπερεκτιμά ποτέ το πραγματικό κόστος,
  - **consistent**: $h(n) \leq c(n, n') + h(n')$.
- Ο A* επεκτείνει nodes με το μικρότερο $f(n) \rightarrow$ **καθοδηγεί την αναζήτηση προς το goal**.
- Σε grids χρησιμοποιούνται συχνά:
  - Ευκλείδεια απόσταση
  - Manhattan απόσταση

A* Algorithm - Απόδοση και Επεκτάσεις

- Ο A* απαιτεί πολύ λιγότερα node expansions από τον Dijkstra, όταν το heuristic είναι καλό.
- **Χρονική πολυπλοκότητα**: εξαρτάται από την ποιότητα του $h(n)$. Συνήθως πολύ καλύτερη από Dijkstra.
- **Suboptimal A* (Weighted A*)**:

$$f(n) = g(n) + \epsilon\, h(n)$$

  - Γρηγορότερη αναζήτηση με εγγυημένη sub-optimality.
  - Το path κοστίζει το πολύ ϵ φορές το βέλτιστο.
- **Anytime A***:
  - Ξεκινά με υψηλό $\epsilon \rightarrow$ πολύ γρήγορη λύση.
  - Μειώνει προοδευτικά το $\epsilon \rightarrow$ βελτιώνει τη λύση.
  - Ιδανικό για real-time ρομποτικά συστήματα.

D* Algorithm -Incremental Replanning

- Ο D* είναι μια **incremental replanning** εκδοχή του A*.
- **Ιδέα**: όταν αλλάξει το περιβάλλον (π.χ. εμφανιστεί νέο εμπόδιο), δεν χρειάζεται να γίνει νέο A* από την αρχή.
- Αντίθετα, ο D* επαναυπολογίζει μόνο τα **επηρεασμένα** states του γράφου.
- Στη ρομποτική, οι αλλαγές είναι συνήθως **τοπικές**, άρα:

  πολύ μικρότερο υπολογιστικό κόστος από A*

- Για αποδοτικότητα, η αναζήτηση γίνεται αντίστροφα:

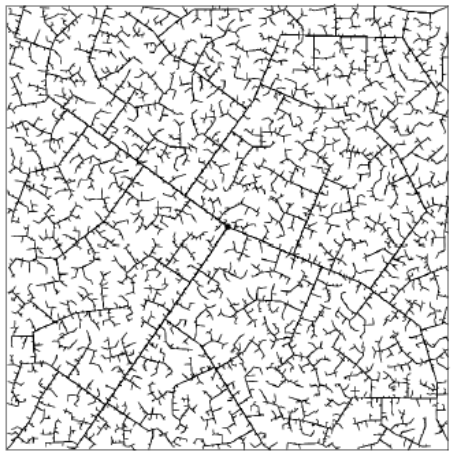
  ξεκινάει από το goal και «διαχέει» το κόστος προς τα έξω

- Με αυτό τον τρόπο, μεγάλα τμήματα της προηγούμενης λύσης **παραμένουν έγκυρα**.
- Μείωση χρόνου αναζήτησης: 1–2 **τάξεις μεγέθους ταχύτερος** από τον A*.
- Υπάρχει και η έκδοση **Anytime D***, όπως το Anytime A*, που βελτιώνει σταδιακά την ποιότητα του μονοπατιού.

Randomized Graph Search - Τυχαιοποιημένη Αναζήτηση

- Σε υψηλής διάστασης προβλήματα (manipulators, molecule folding κ.λπ.) η εξαντλητική αναζήτηση είναι **ανέφικτη**.
- Κατάλληλο heuristic συχνά **δεν υπάρχει** και η μείωση διάστασης **παραβιάζει κινηματικούς/δυναμικούς περιορισμούς**.
- Τυχαιοποιημένες μέθοδοι:
  - **Δεν εγγυώνται optimal ύ**
  - Αλλά δίνουν **πολύ γρήγορες** λύσεις
  - Ιδανικές για **πολύπλοκους, μη-κυρτούς** χώρους
- Η πιο δημοφιλής μέθοδος: **Rapidly-exploring Random Trees (RRTs)**

45 iterations



2345 iterations

Rapidly-exploring Random Trees (RRTs) - Μηχανισμός

- Το RRT χτίζει ένα **δέντρο** στον free space **online** κατά την αναζήτηση.
- Σε κάθε βήμα:
  1. Επιλέγεται μια **τυχαία διαμόρφωση** $q_{\text{ρανδ}}$.
  2. Βρίσκουμε το κοντινότερο node του δέντρου: $q_{\text{νεαρ}}$.
  3. Επεκτείνουμε το δέντρο από $q_{\text{νεαρ}}$ προς $q_{\text{ρανδ}}$ με **σταθερό μήκος** και με βάση το motion model.
  4. Αν η ακμή δεν έχει σύγκρουση, προσθέτουμε το νέο node.
- **Επεκτάσεις RRT**:
  - **Bidirectional RRT**: δέντρα από start & goal $\rightarrow$ ταχύτερη σύγκλιση.
  - **Goal biasing**: επιλέγουμε το goal ως random δείγμα με κάποια πιθανότητα.
- **Ιδιότητες**:
  - Όχι optimal, ούτε deterministically complete.
  - **Probabilistically complete**:

$$\text{αν υπάρχει λύση, θα βρεθεί καθώς τα samples} \rightarrow \infty$$