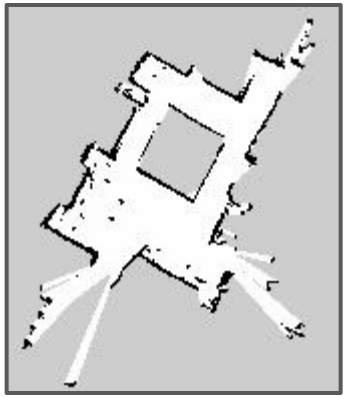
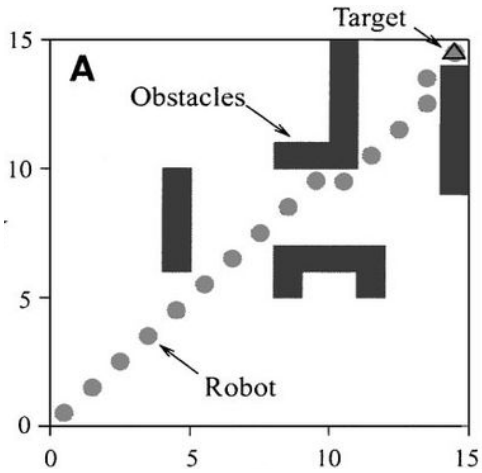


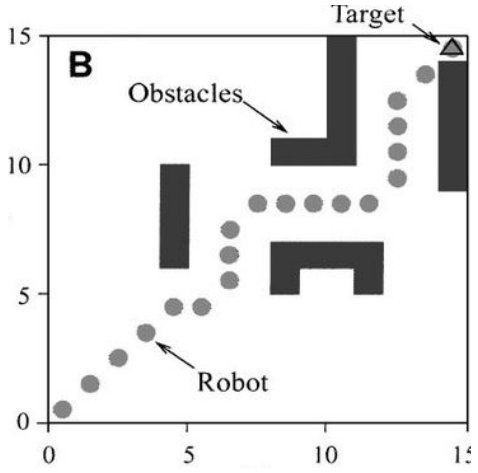
Autonomous Navigation

Perception, Path-Planning and Mapping

Definitions:

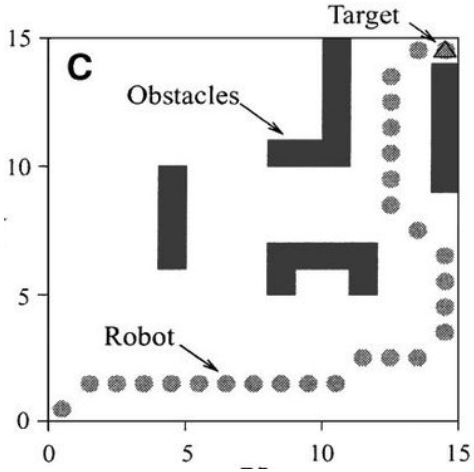
- **Navigation:**
  The process of:
  1. Calculating one's position
  2. Planning a route to a destination
  3. Following the route correctly





- **Autonomous:**

  Having the freedom to take one's own decisions without orders from someone else



How do Humans Navigate?

Consider the the following task:
*Walking from your house to your friend's house.**

Decision to take: Which direction to move in next (at each time instance)

Question:

1. What all do we need to know to take this decision?
2. How do we get this knowledge?

*Don't go to your friend's house. Stay at home

Required Information

1. **The route** - Which street, turns, etc.
2. **Surroundings** - Obstacles around you, walls, ditches, steps, other people
3. **Path** - Your way around the obstacles, planning your *exact* path
4. **Current position** - Where exactly are you right now (WRT goal, obstacles, etc.)

1. Map of the area *(streets, houses, landmarks)*
2. How to use this map?
3. What if you don't have a map?

1. Look around
2. *Understand* the objects around you

1. Keep away from obstacles
2. Stay on the route
3. Stay aware of moving obstacles
4. Plan/replan your immediate path

1. Landmarks *(have you seen them before?)*
2. How fast are you walking? How long have you walked?
3. Keep your eyes open

Navigation in Robots

1. **Global Path Planning** *(route)*
2. **Obstacle detection** *(surroundings)*
3. **Local Path Planning** *(path)*
4. **Localization** *(position)*

1. Pre-existing map as a graph
   OR
   Realtime mapping + Exploration
2. A graph search algorithm for Global route
   Eg: Dijkstra, Bellman-Ford, A-star

1. Sensors
   Eg: LIDAR, stereo cameras
2. Interpreting data

1. Path-planning algorithms
   Eg: Potential Field, Probabilistic Roadmap, RRT

1. Feedback based *(encoders, IMU)* OR Vision-based *(RoVIO)*
2. Sensor Fusion *(Kallmann Filter, EKF)*

Contents:

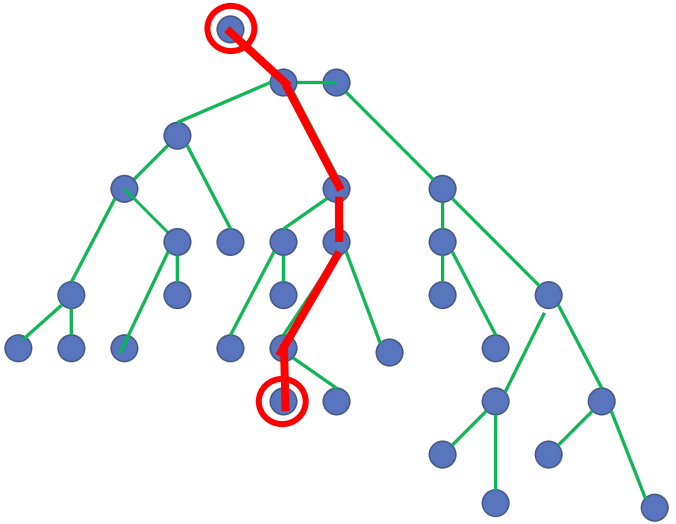
Path Planning

Planning the path that a robot/manipulator/joint will follow to go from the current position to a given goal point

Can be classified based on:

- **Purpose:**
  a. Global
  b. Local

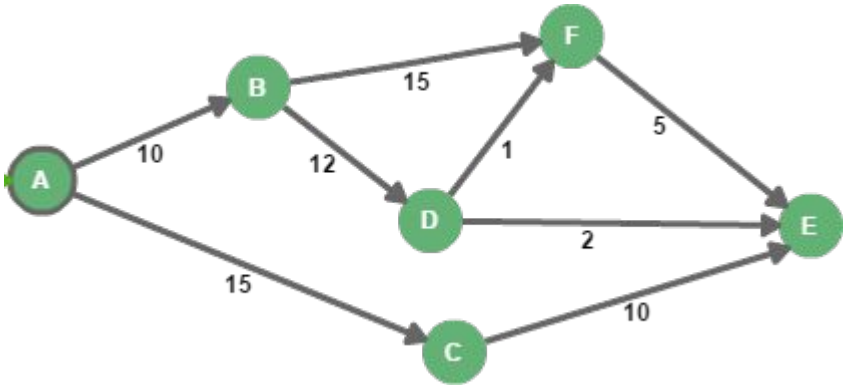
- **Method:**
  a. Graph/Grid based
  b. Probabilistic
  c. Miscellaneous



Graph-based Path Planning

Graphs in Computer Science:

A set of vertices (**nodes**) that are connected by segments (**edges**), where each edge can have a value (**weight**).



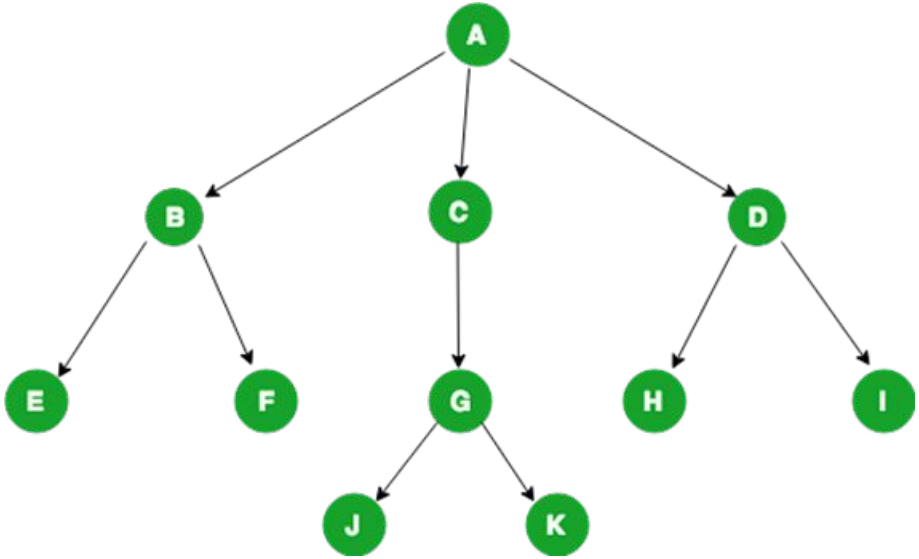
In path planning,
Node → Position
Edge weight → Cost to go from one node to another (eg: Distance, time, power, etc.)

Example:
Nodes: {A, B, C, D, E, F}
Edges: { AB, AC, BD, BF, DF, CE, DE, FE }

Tree:
A graph without loops

Terminologies:

1. Root Node
2. Parent Nodes
3. Child Nodes
4. Leaf Nodes
5. Depth/Height

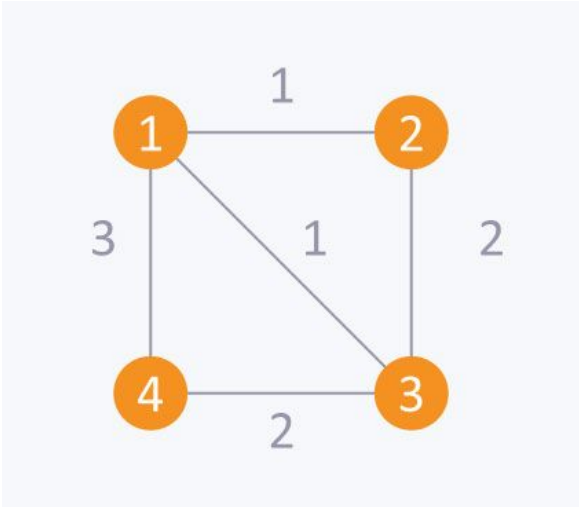


Eg: *Here,*
Root: A Leaves: E, F, J, K, H, I
A is the parent of B
E, F are children of B
Max Depth: 4

Representing a Graph

Method 1: Matrix *(eg: numpy array)*

	1	2	3	4
1	0	1	1	3
2	1	0	2	∞
3	1	2	0	2
4	3	∞	2	0



Method 2: Adjacency list *(eg: Python dictionary)*

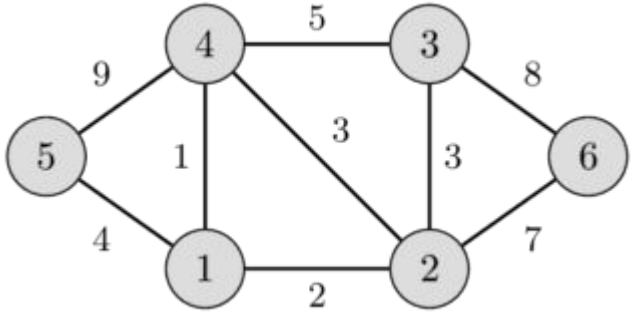
1: {(2,1), (3,1), (4,3)}
2: {(1, 1), (3, 2)}
3: {(1,1), (2,2), (4,2)}
4: {(1,3), (3,2)}

Shortest Path

Path from A to B: A sequence of connected nodes starting from A and ending with B.

Length of a path is the sum of the weights of the edges in it.

Shortest Simple Path: The path from A to B that has the least weight.



Eg: In the given graph, find the shortest path from Node 5 to Node 6.
What is the length of the path?

Dijkstra’s Algorithm

The most widely used **shortest-path** algorithm.

Given a weighted graph and a starting node,
Dijkstra finds the shortest path to *all* nodes.

Idea:

Starting from the source node, we ‘explore’ the graph, by **‘visiting’** adjacent nodes.

Each time we ‘visit’ a node, we update it’s distance from the source

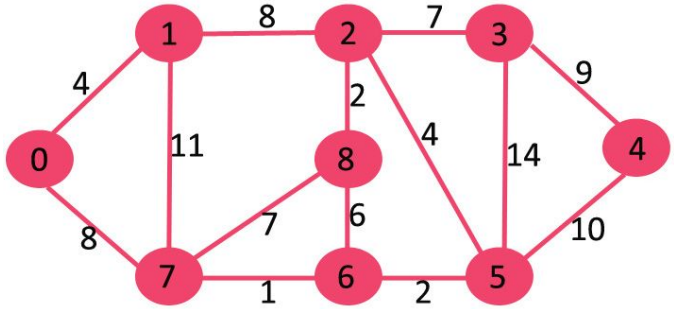
At each iteration, we visit the unvisited node that is nearest to the source.

Dijkstra (continued)

Initialization

Source Node: **0**

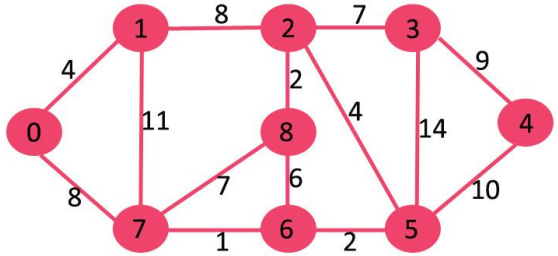
Initialize all distances to infinite (except the source)
Create an empty list of ‘visited nodes.



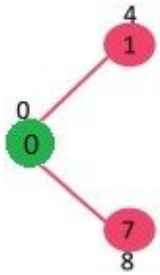
Loop:
At each iteration:

1. Choose the unvisited node with the least distance value. Mark it as visited.
2. Calculate the known distances to its neighbours and update them.

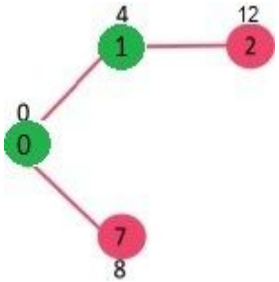
0	1	2	3	4	5	6	7	8
0	∞	∞	∞	∞	∞	∞	∞	∞



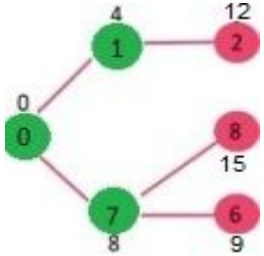
0	1	2	3	4	5	6	7	8
0	∞	∞	∞	∞	∞	∞	∞	∞



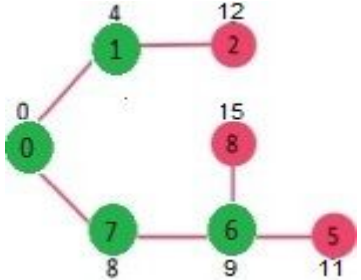
0	1	2	3	4	5	6	7	8
0	4	∞	∞	∞	∞	∞	8	∞



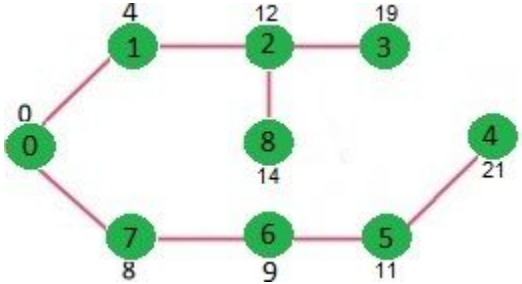
0	1	2	3	4	5	6	7	8
0	4	12	∞	∞	∞	∞	8	∞



0	1	2	3	4	5	6	7	8
0	4	12	∞	∞	∞	9	8	15



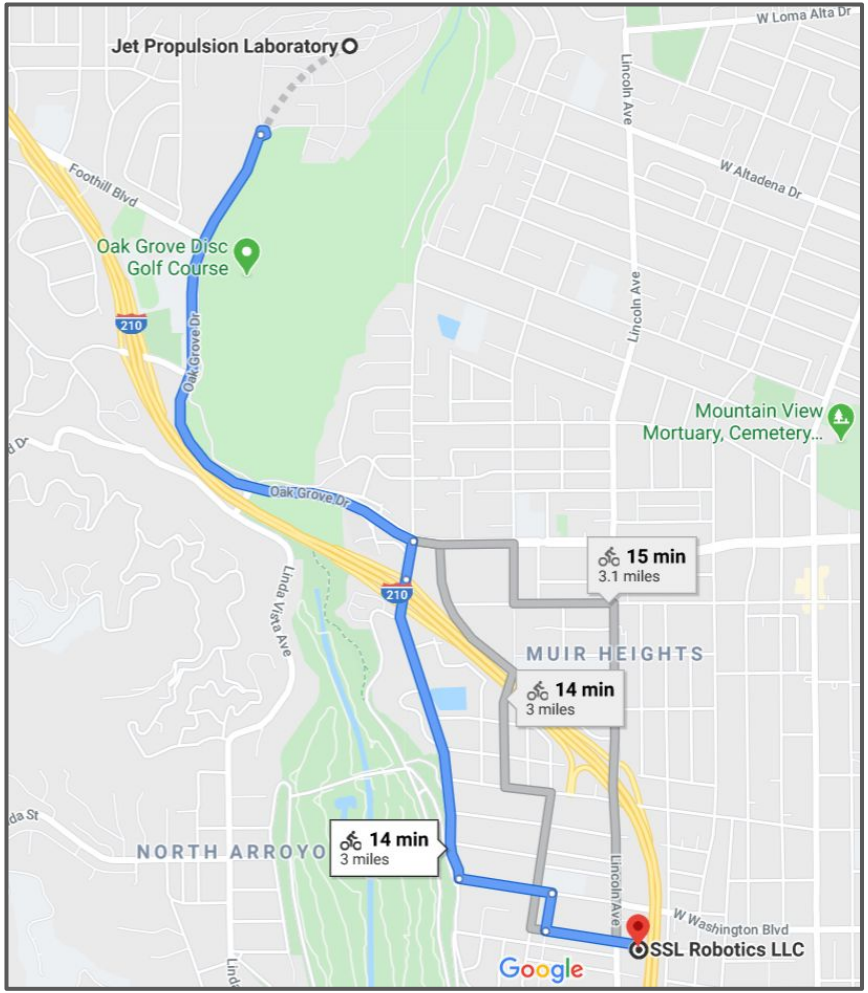
0	1	2	3	4	5	6	7	8
0	4	12	∞	∞	11	9	8	15

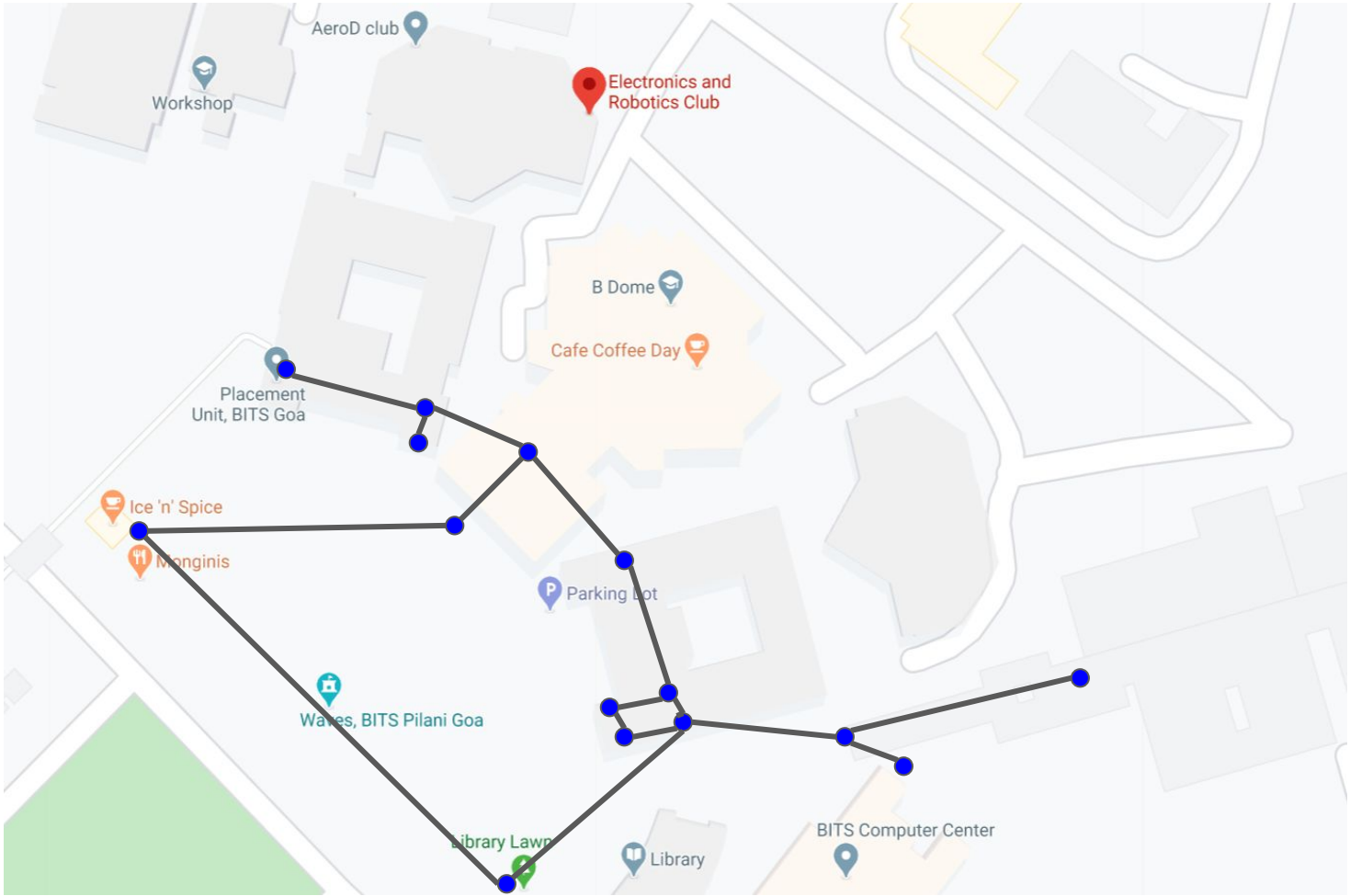


0	1	2	3	4	5	6	7	8
0	4	12	19	21	11	9	8	15

Global Path Planning using Dijkstra

- The map of the surroundings is a graph
- Every intersection is a node
- Every road joining 2 intersections is an edge.
- The edge weight is chosen depending on the purpose.
  Eg: In Google Maps, the edge weight is the estimated time to traverse the edge




AeroD club
Workshop
Electronics and Robotics Club
B Dome
Cafe Coffee Day
Placement Unit, BITS Goa
Ice 'n' Spice
Parking Lot
Library Lawn
Library
BITS Computer Center

A* (A-star) search algorithm

Another graph based path finding algorithm.

Uses a **‘heuristic’** distance from each node to all other nodes.
This is an estimate of the distance between the nodes.

Selects node based on the sum of known distance to an intermediate node and heuristic distance from intermediate node to the goal

Outputs **one of the shortest paths**, and not necessarily the shortest path.

Widely used in **Artificial Intelligence**

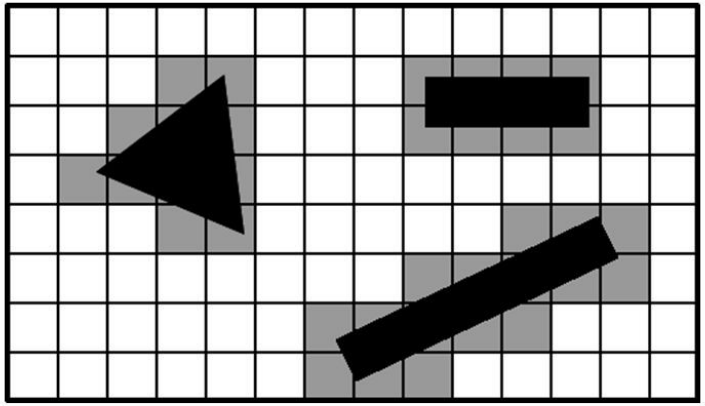
O(n) complexity. *(Dijkstra is* $\boldsymbol{O(n^2)}$ *)*

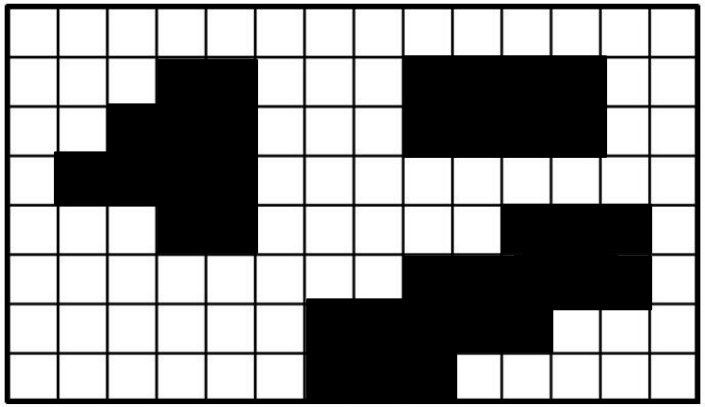
Occupancy grids

Method of representing obstacles -

1. The known area is divided into grid cells
2. Each cell containing an obstacle is marked as occupied (black)
3. Maps in ROS are stored as occupancy grids

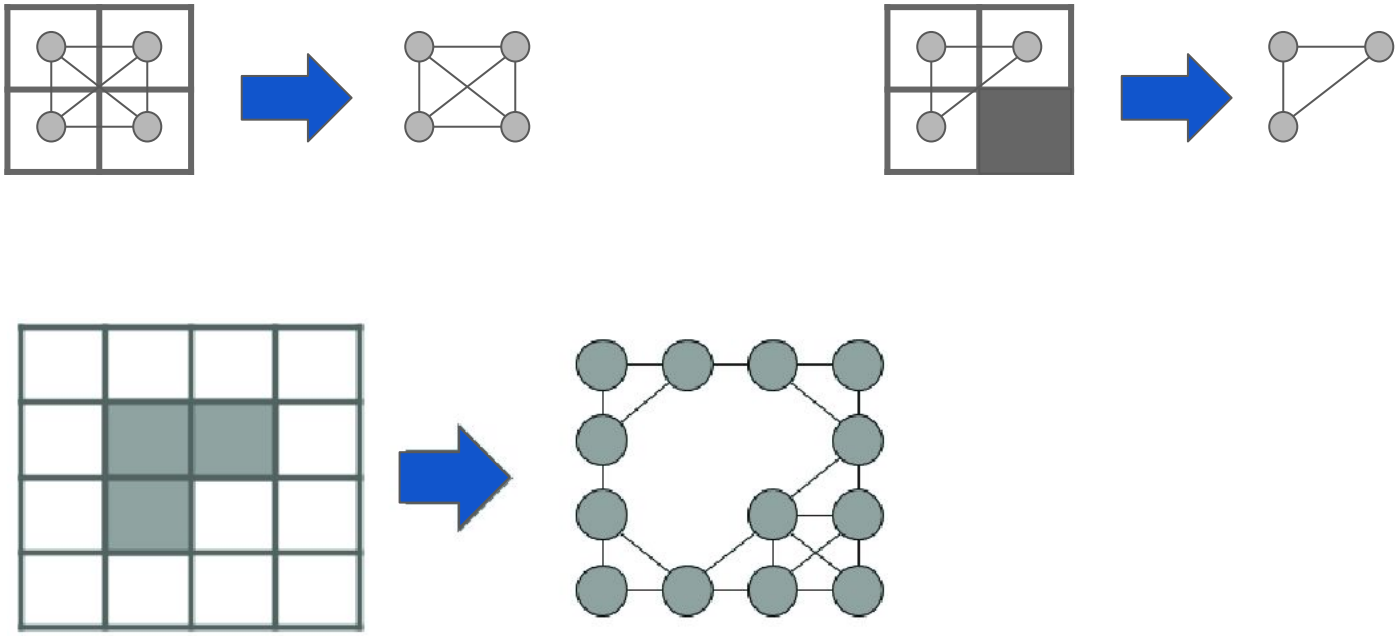
How do we use occupancy grids in path-planning?





Grid-based Path Planning

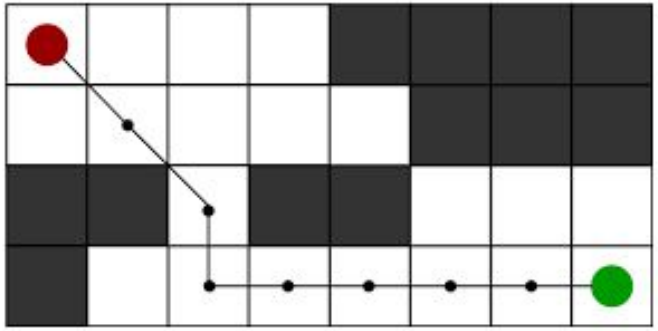
Occupancy grids can be converted to graphs

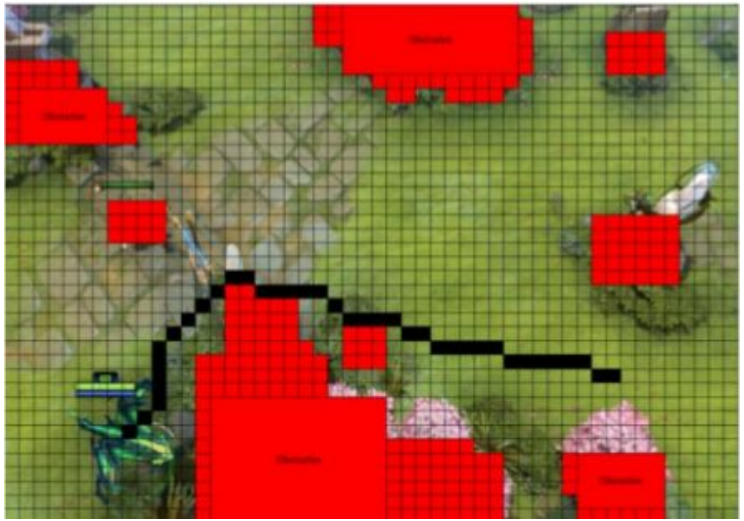


A* for path planning

The occupancy grid is converted to a graph where,

1. Adjacent free nodes are connected by an edge of weight **L.**
2. Cells sharing a corner are connected by an **L√2** edge.





A used by characters in DOTA 2, LoL, AoE, etc.*

Other graph/grid based algorithms

- Bellman Ford
- Grassfire
- A
- Trapezoidal decomposition
- Prim’s MST/ Kruskal’s MST

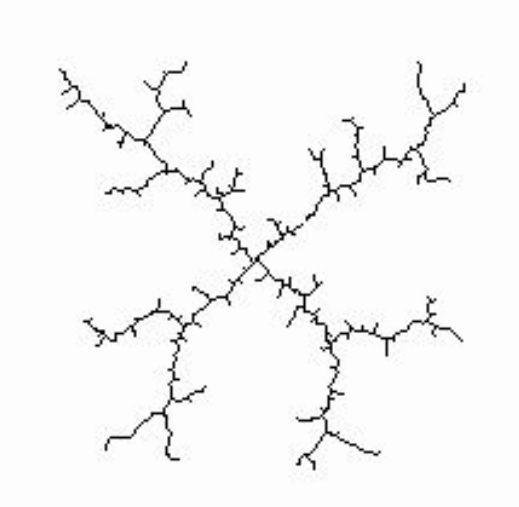
Reference for learning:

Issues with Graph/Grid based methods

- Robot space is modeled as discrete, when it is actually continuous
- Maps (graphs or occupancy grids) may not always be available
- Best-paths are not always needed. ‘Goog’ paths can suffice.
- Good for Global planning, but not for local planning.

Probabilistic Path Planning Algorithms

- Sampling based
- No guarantee of getting a path at any point of time (Always a **probability**)
- Give a possible path that may not be optimal
- Robot space is **continuous** (Not discrete)



Probabilistic Roadmap

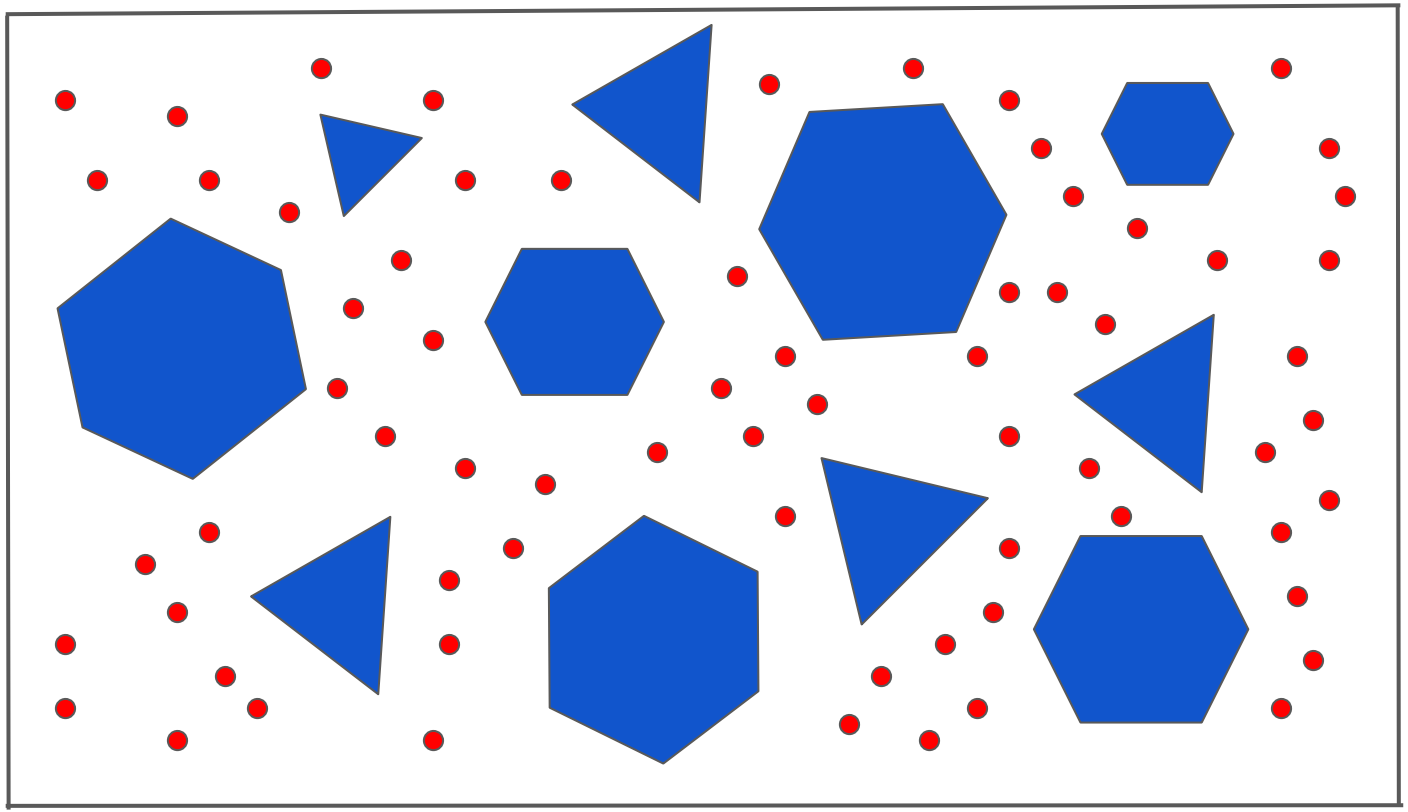
- **Step 1:**
  Sampling random points in the robot space

- **Step 2:**
  Joining the points with their neighbours
  (Typically K-nearest neighbours algorithm)

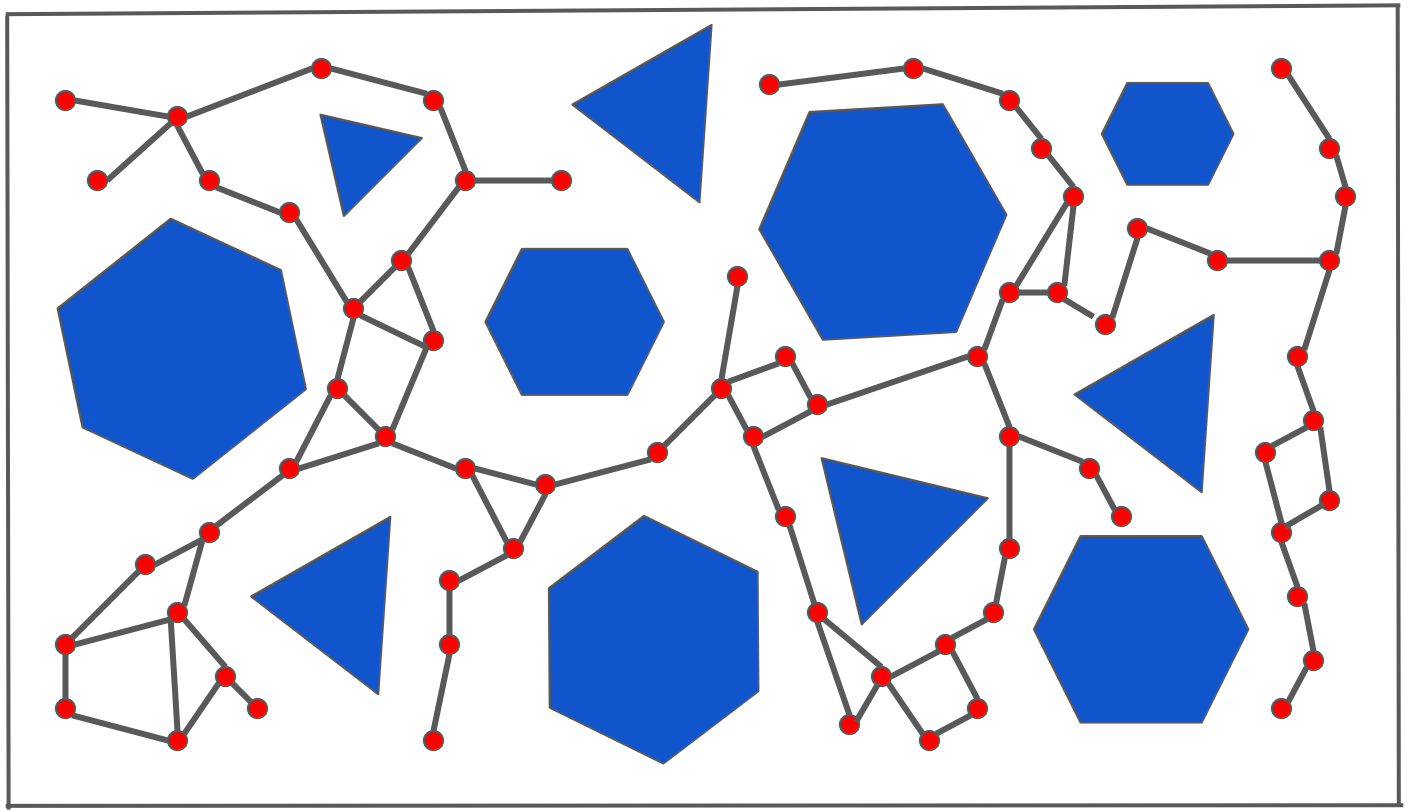
- **Step 3:**
  Query

- **Step 3:**
  Graph based planner in the graph obtained
  (Typically Dijkstra)

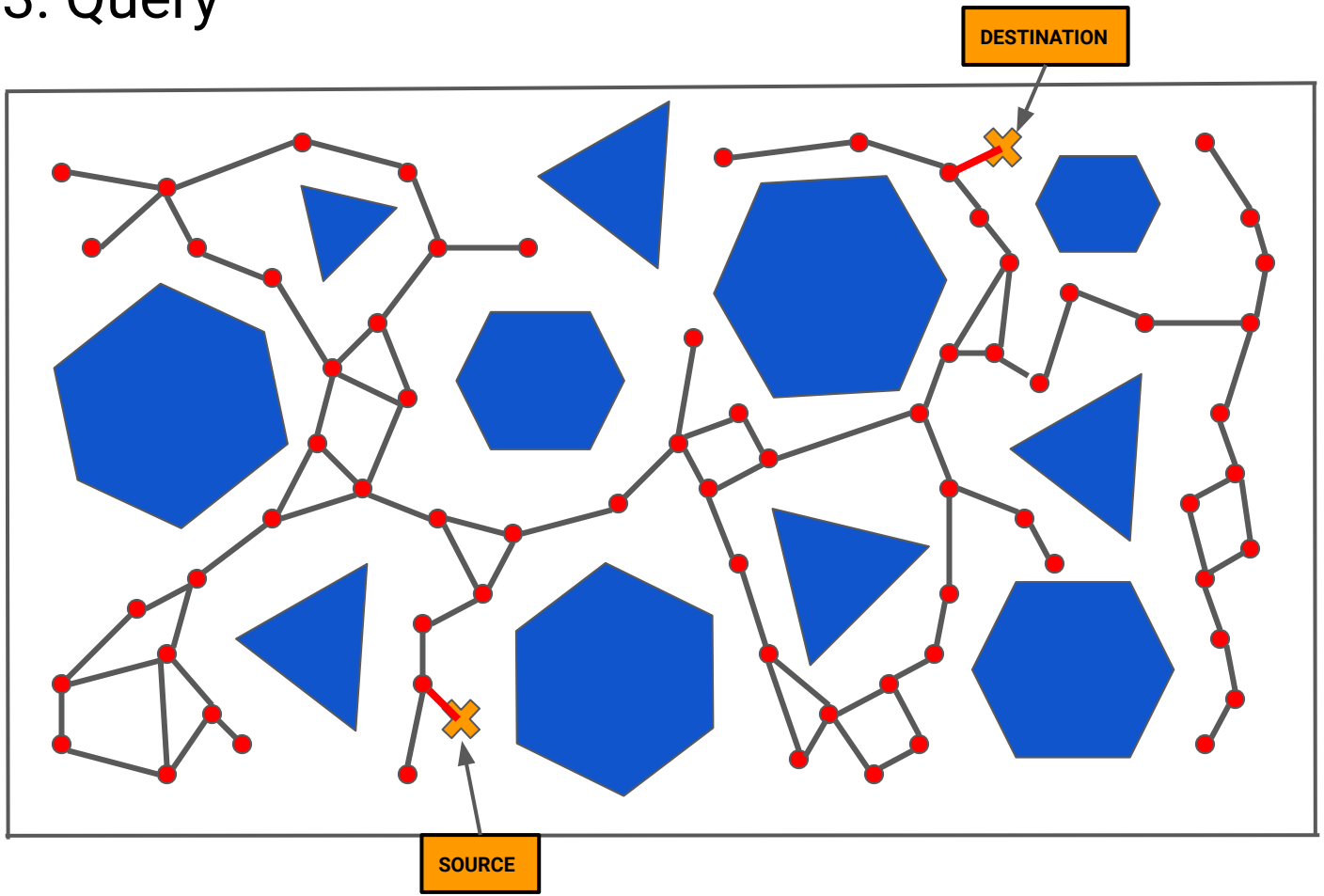
Step 1: Sampling points



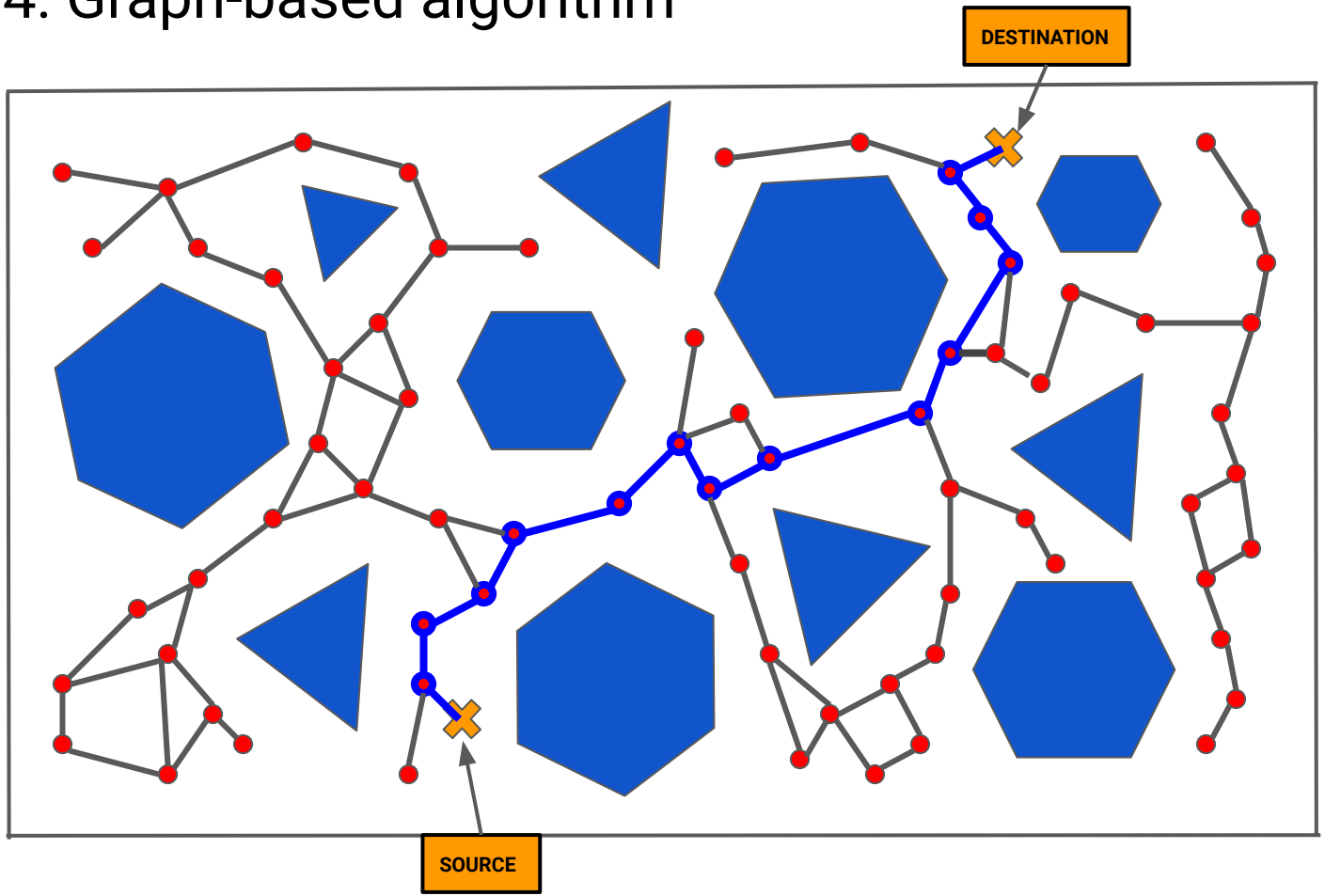
Step 2: Connecting Neighbours



Step 3: Query



Step 4: Graph-based algorithm



Rapidly Exploring Random Tree (RRT)

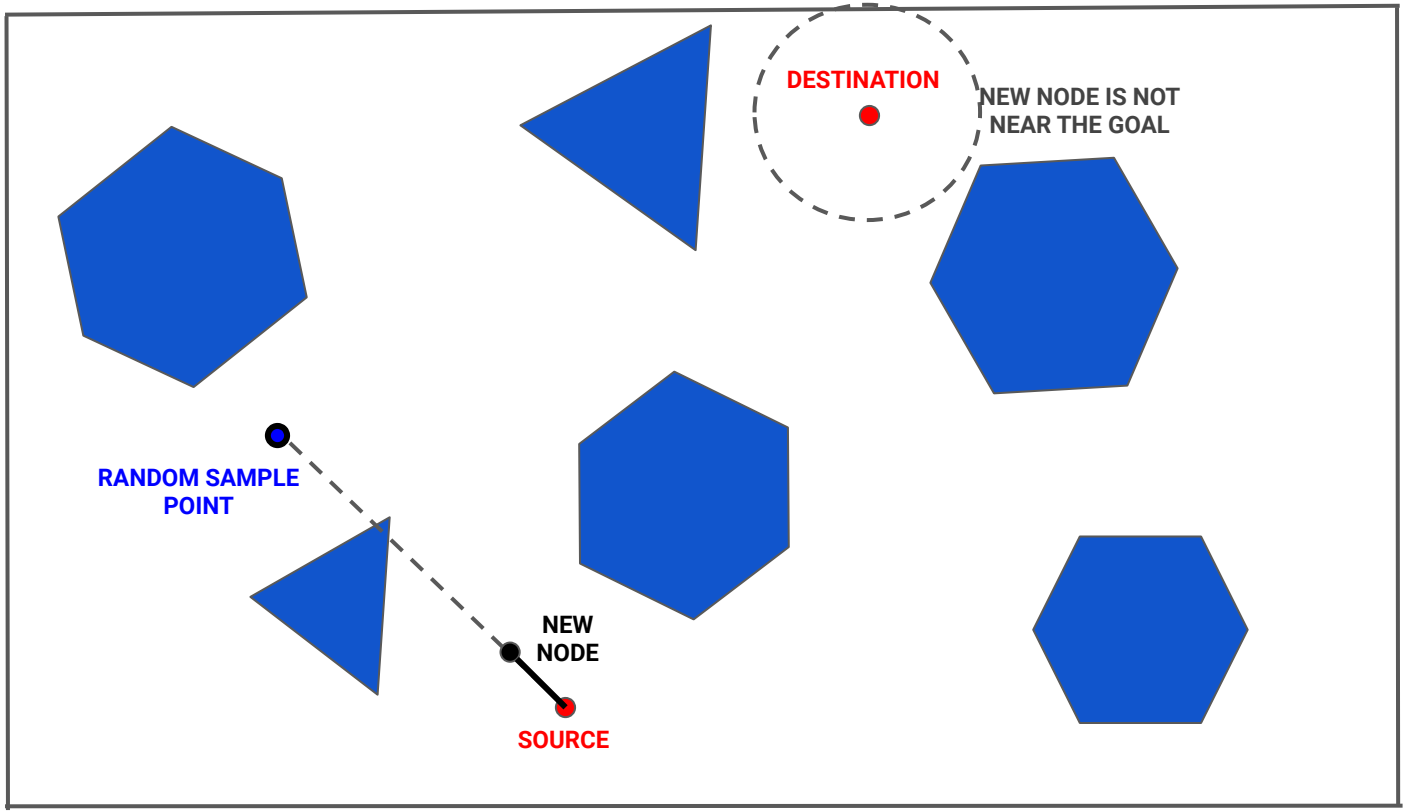
Initially, the tree contains only the source

- **Step 1:**
  Sample a random point that is not on an obstacle
- **Step 2:**
  Add a neighbour to the nearest existing node, in the direction of the new point
- **Step 3:**
  Check if the new node is close to the destination
- **Step 4:**
  Backtrack

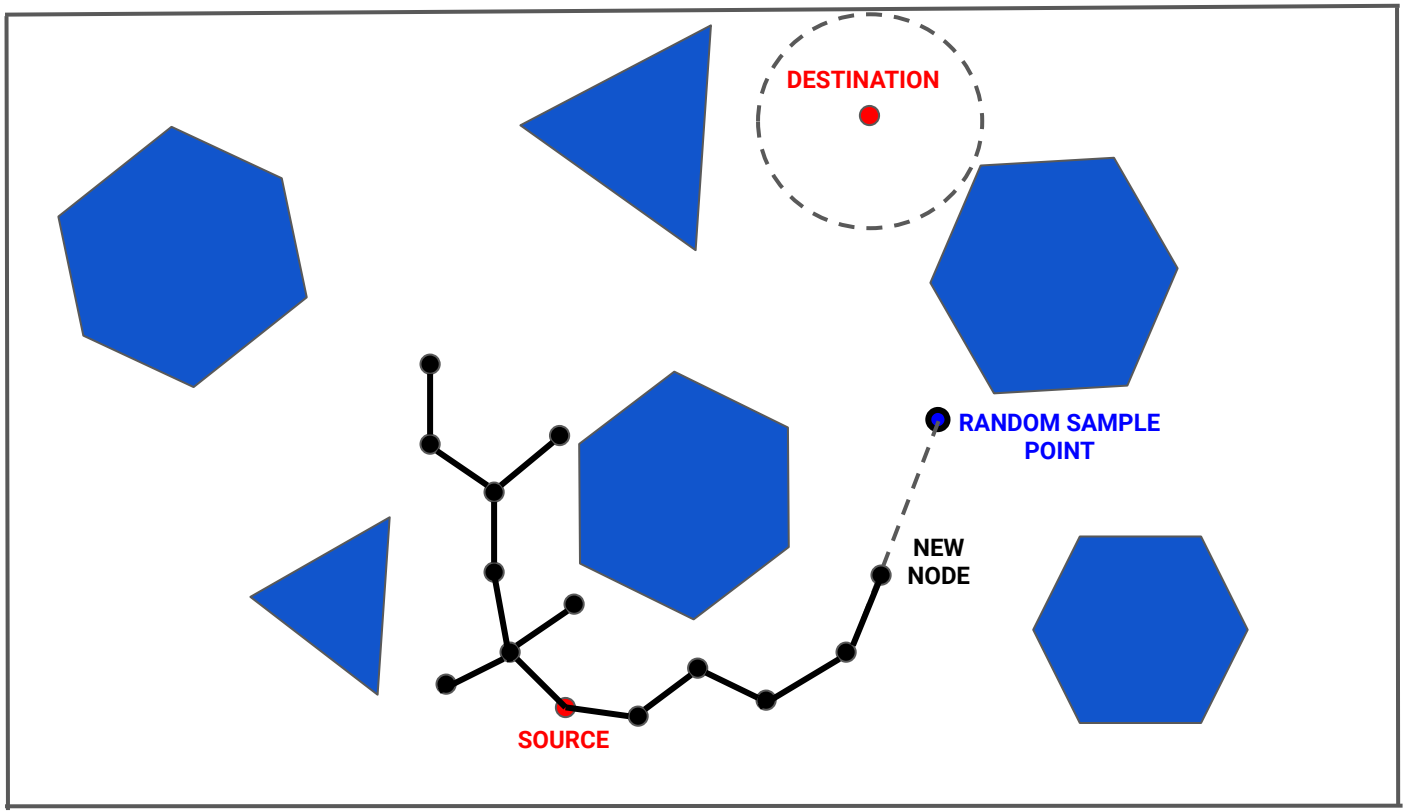
Loop: (Steps 1 to 3)

(Until **step 3** returns **True**)

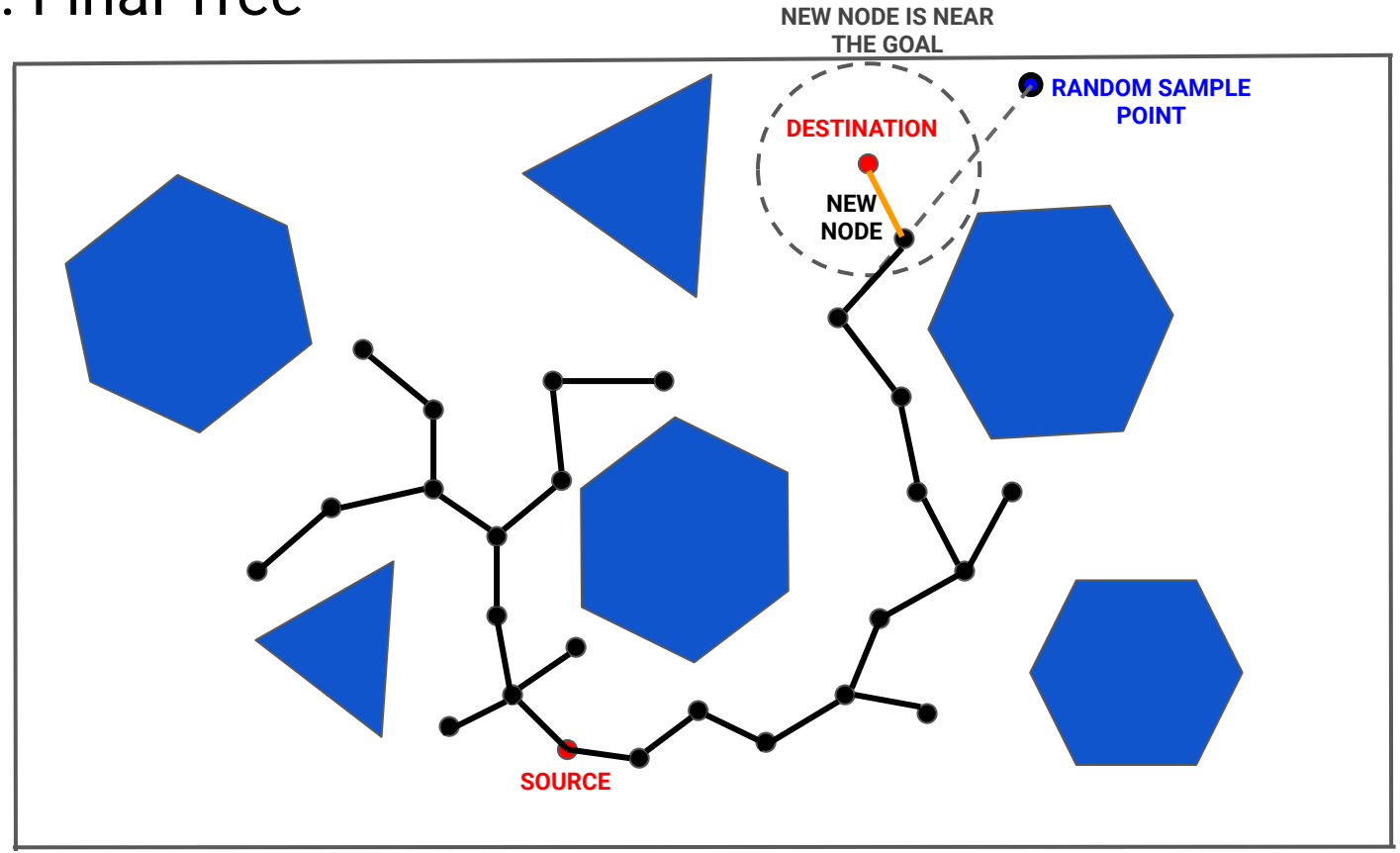
RRT: Initial configuration



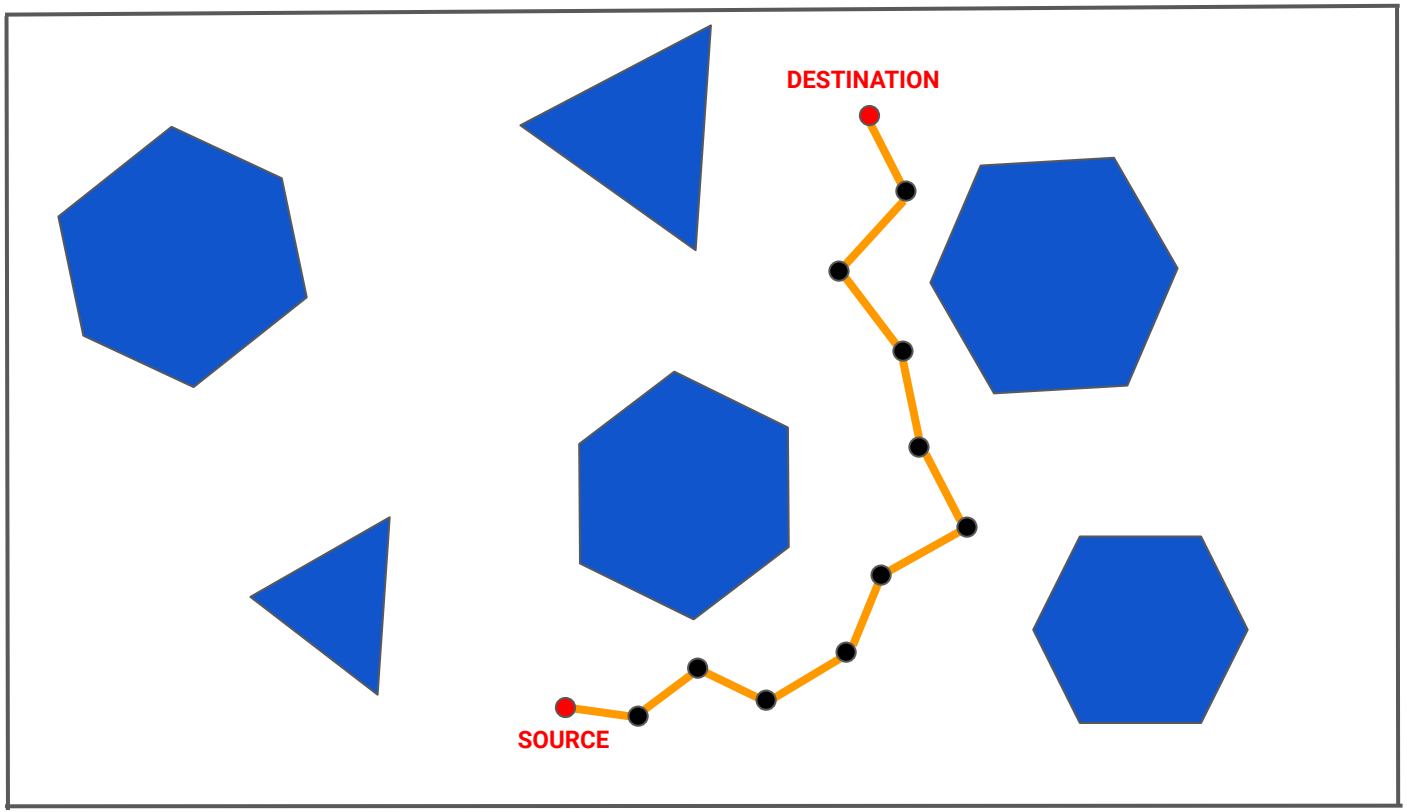
RRT: Intermediate configuration



RRT: Final Tree



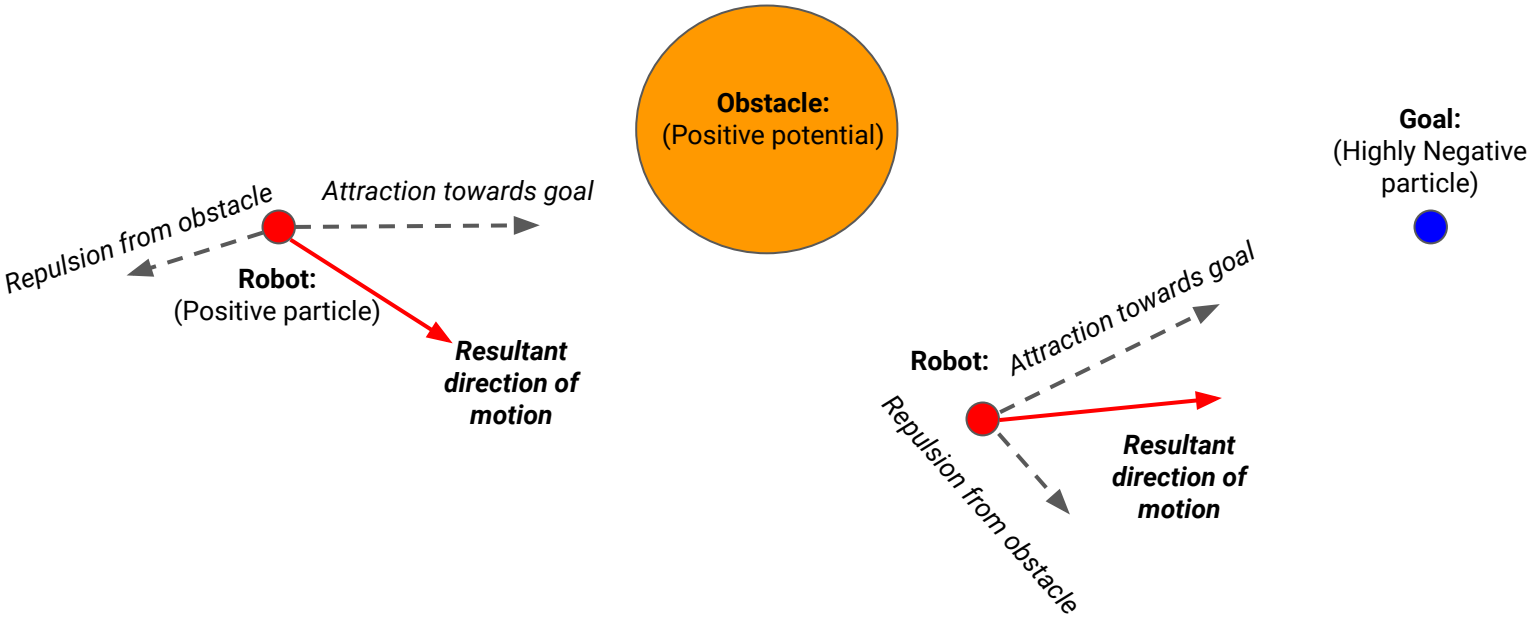
RRT: Backtracking



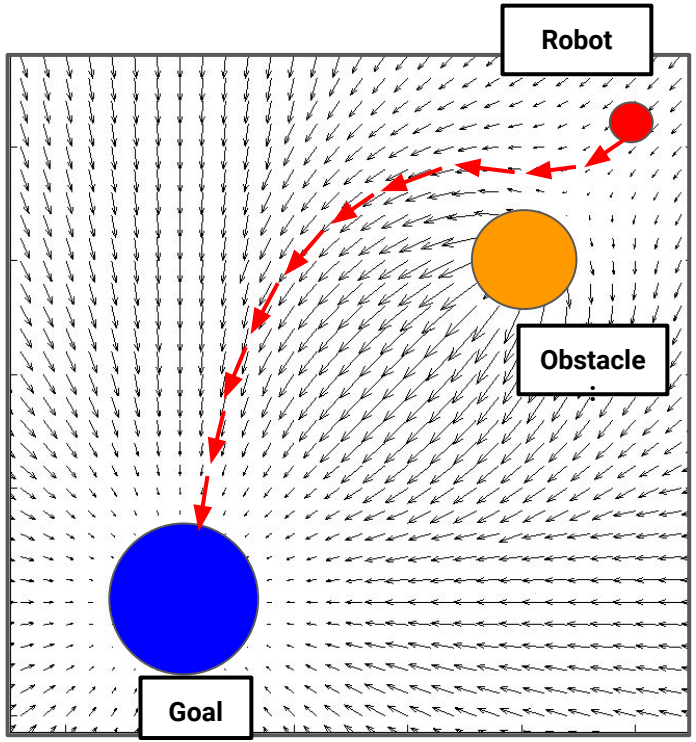
Potential Field Motion Planner

- The system is modeled as particles with potential fields
- The **robot** is assumed to be a particle of high **positive potential**
- The **Goal** is assumed to be a fixed point of high **negative potential**
  (The robot faces an attraction vector towards the goal)
- **Obstacles** are assumed to have a high **positive potential**
  (The robot experiences repulsion from the obstacles)
- Every point in space has a certain **direction of field**. This direction decides the direction of motion of the robot.

Potential Field Example



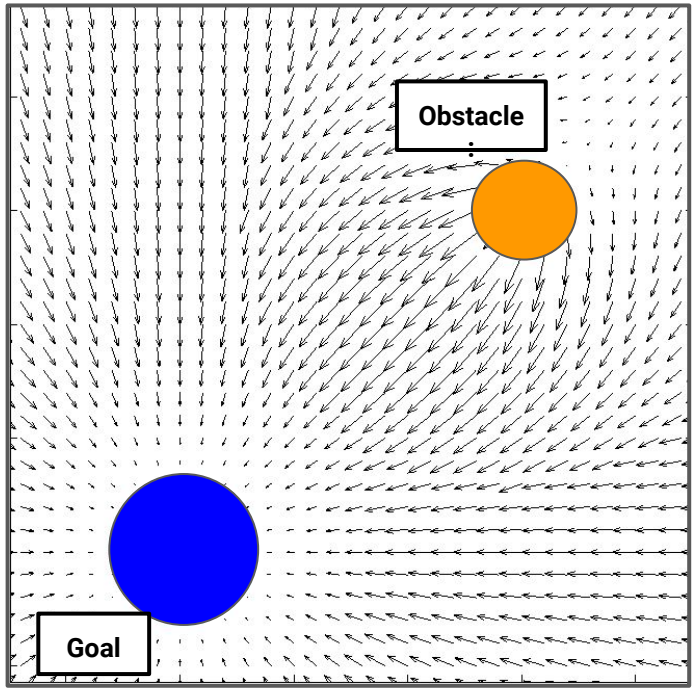
Calculating the Direction of motion



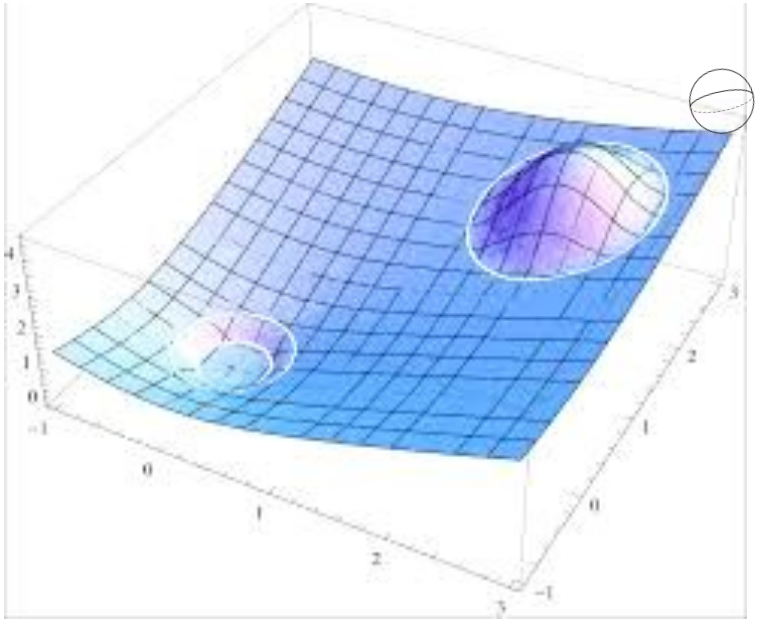
- The field at any point in space is the resultant vector of the fields of all obstacles and the goal
- The direction of the field at any point gives the direction of motion of the robot at that point.

Image:
Arrows represent the resultant field at each point

Alternative Intuition



Inclined plane sloping towards the goal, where obstacles are peaks



Motion of the robot: Similar to the motion of a ball rolling down this surface

Advantages of Potential Field

- Gives a **smooth path** (unlike RRT and Roadmap algorithms)
- Useful for **moving obstacles**
- **On-the-fly** calculation possible

Localization

Definition:
Determining the position & orientation of the robot with respect to the environment.

Methods:

- Dead-reckoning
- Inertial
- Encoder feedback
- Visual
- Hybrid
- SLAM

Least reliable

Most reliable

Dead Reckoning

(Method used by Medieval sailors for navigation)

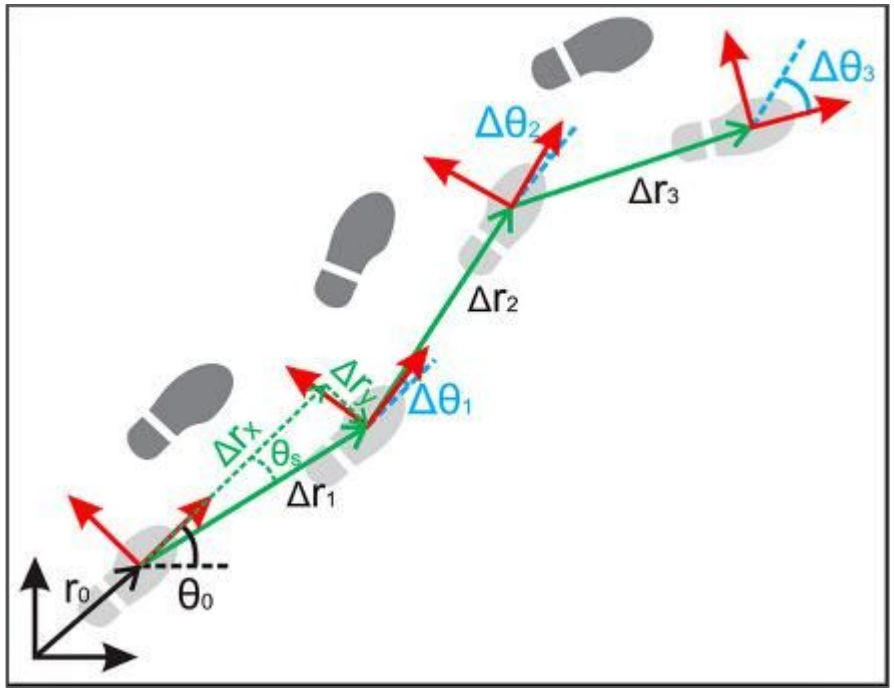
Idea:

- Assume that **initial position** is known
- **Direction** of motion is decided by the sailor
- **Speed** is decided by the sailor
- **Duration** for which the ship is moving is known
- Assuming the ship moves in the desired direction and speed, position is always known.

Issues with Dead Reckoning

- **Bad Assumption:** Actual velocity and direction are the same as commanded velocity and direction
- **No feedback** from sensors
- Similar to trying to walk to your destination with your eyes closed

Drift:
The error between estimated and actual position grows very fast



Methods with Feedback (1)

Inertial Odometry:

An IMU gives an output of linear acceleration in X,Y,Z and rotational velocities about X,Y,Z.

Basic integration can be used to calculate position and orientation

$$\vec{\theta} = \int_0^T \vec{\omega}\, dt$$

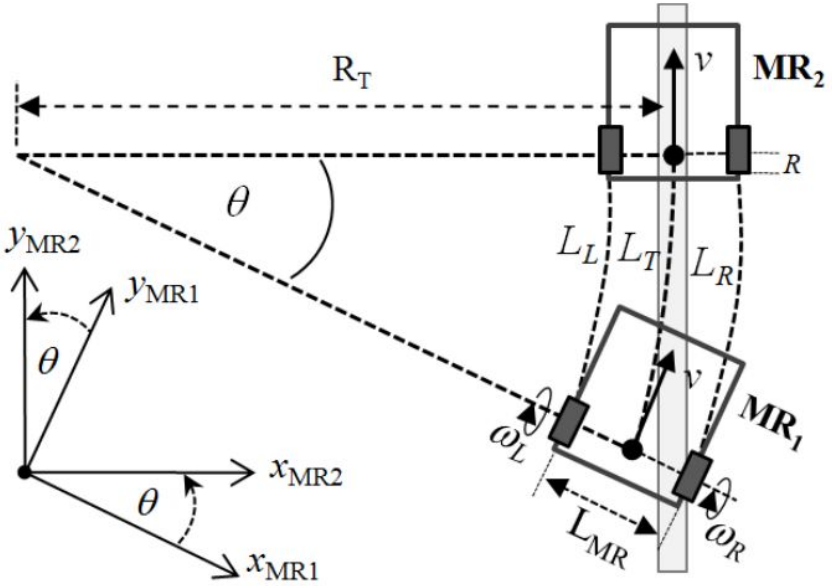
$$\vec{x} = \int_0^T a(t) \cdot t \cdot dt$$

Issue: IMU is not reliable and gives a drift that increases over time

Methods with Feedback (2)

Wheel Odometry:

- Rotary encoders give the angular velocity of each wheel
- Radius of the wheels and the displacement between them are known
- Thus, we can calculate:
  - The linear velocities
  - Position (x, y)
  - Orientation



$$\dot{x} = \frac{(\omega_R + \omega_L)R}{2} \sin\theta$$

$$\dot{y} = \frac{(\omega_R + \omega_L)R}{2} \cos\theta$$

Visual Odometry

- Method used by humans for localization
- When we move, we see the environment move with respect to us
- Our brain keeps a track of static objects around us
- Constant ‘recalibration’ of our position using the positions of these static objects from our frame of reference

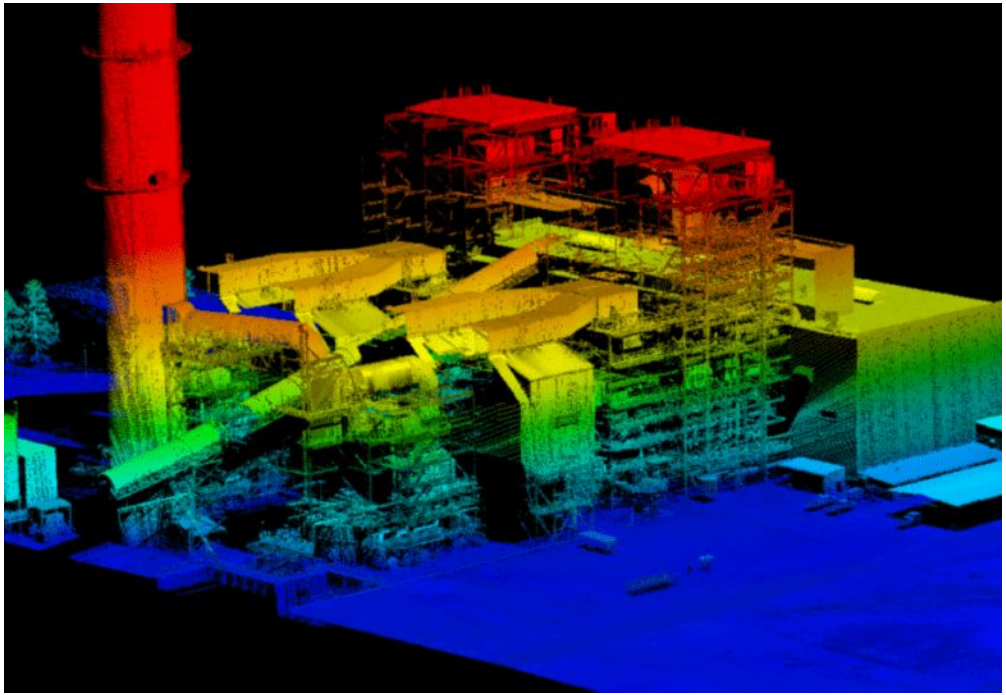
Depth Maps

The first requirement to ‘track’ static features in the surrounding is to ‘see’ in 3D

3D images are represented using depth maps.

A depth map can be obtained in the following ways:

- Stereo Vision
- Active sensors like LIDARs



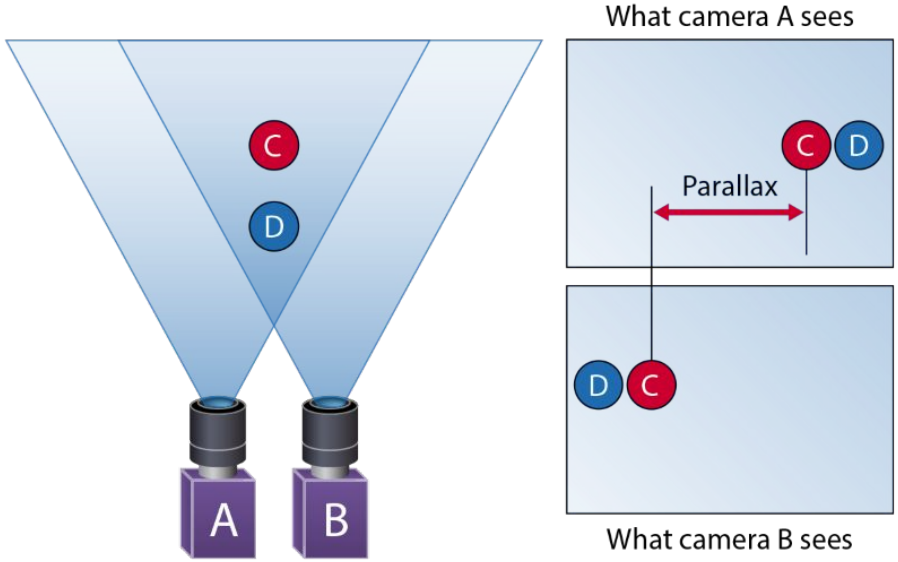
Depth Sensors

Active: 3D LIDARS

Passive: Stereo cameras

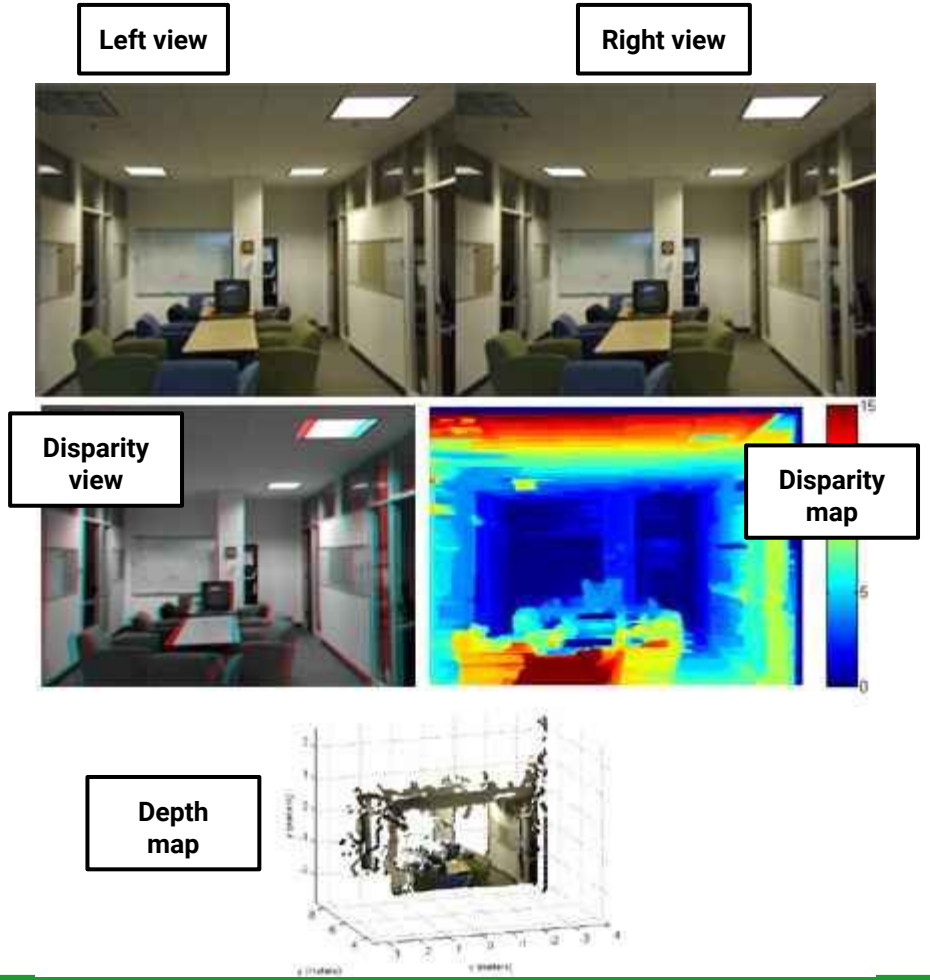
Stereo Vision

- Two cameras are required for depth perception
- Both cameras capture a separate view of the same scene
- These images are compared using image processing algorithms *(will be done in future classes)* to obtain the depth of every point



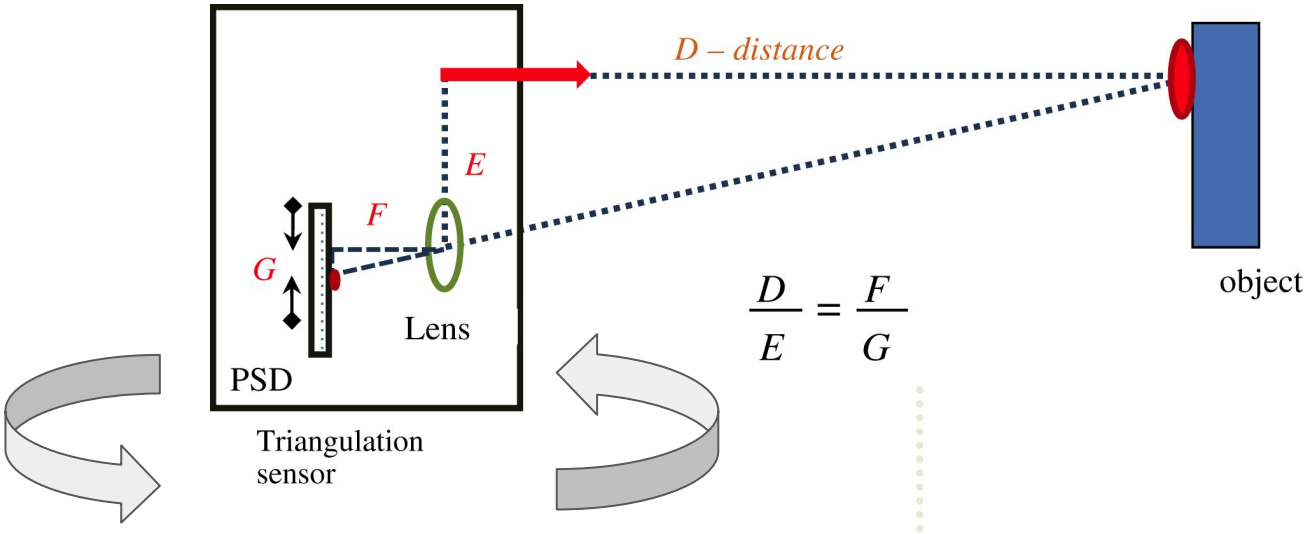
Disparity

- Smart Image processing methods are used to match points in the right view to corresponding points in the left view. *(Will be done in future classes)*
- The displacement between 2 different views of the same point is called **disparity.**
- Disparity is **inversely proportional** to the distance of that point from the camera



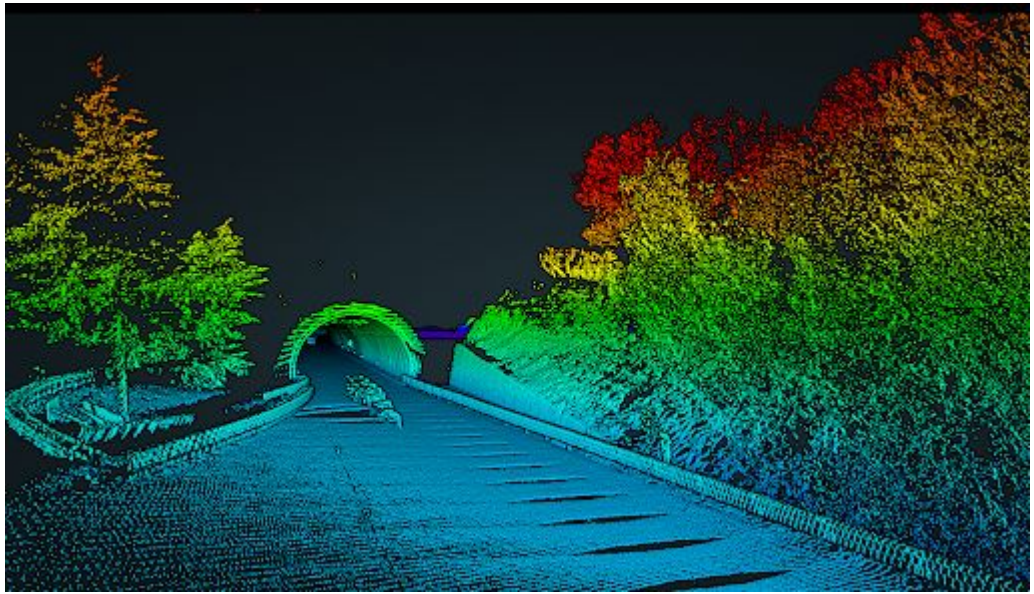
Active Sensors

Laser Rangefinders: Working



Point Cloud

A depth map made of a 2D array of distances obtained from a rangefinder



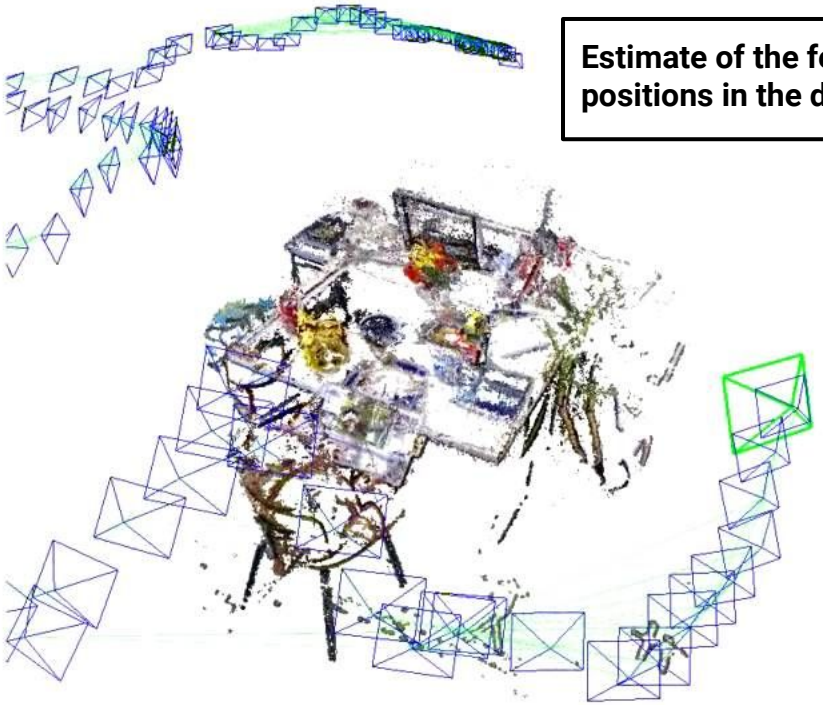
Feature Extraction

- Specific parts of the depth map need to be tracked for comparing over different time instances
- These are usually selected such that they are **unique** and **distinguishable** over multiple frames
- Corners of furniture, borders of the room, irregularities in the floor, etc. are commonly chosen features.

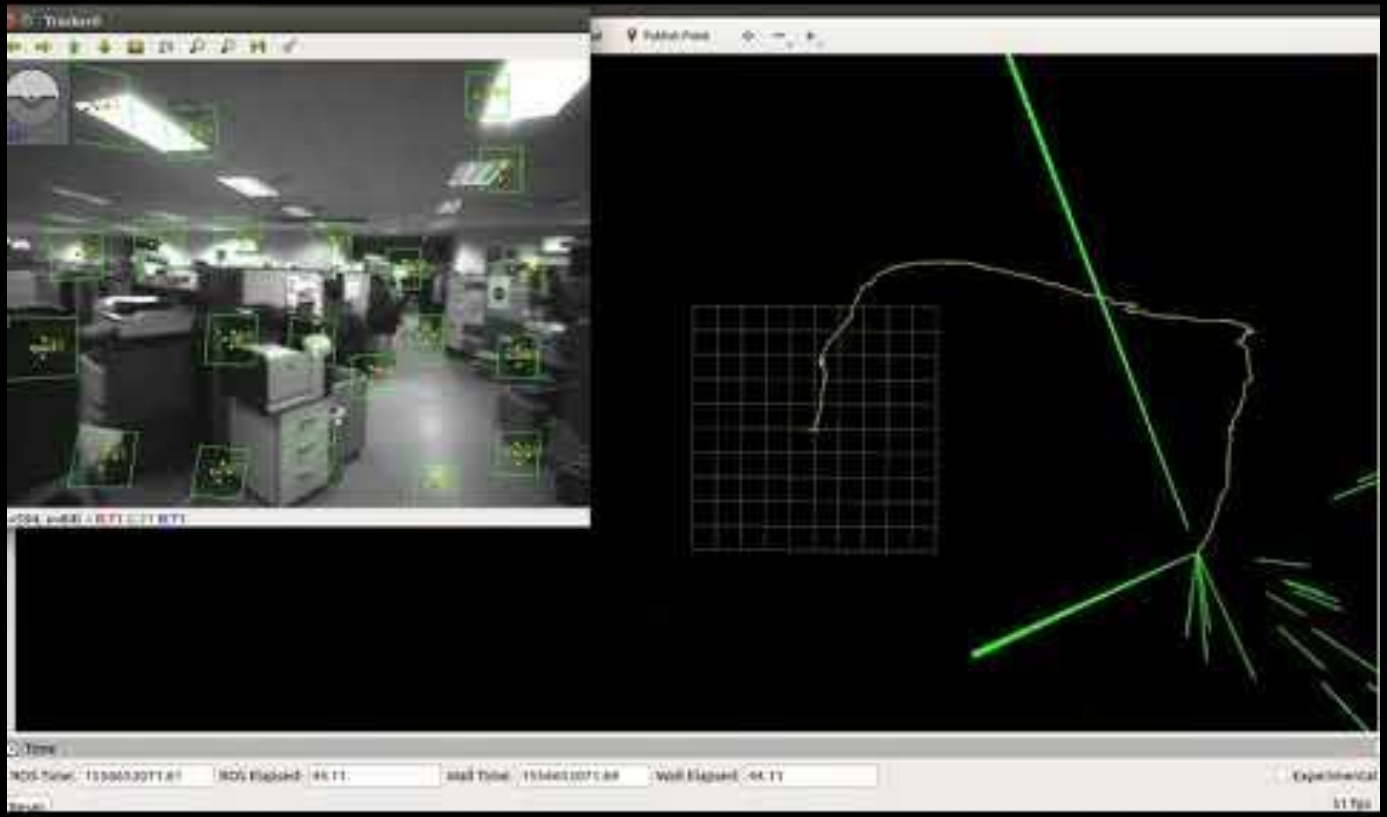
Feature Extraction

Features

Estimate of the feature positions in the depth map



Video: RoVIO



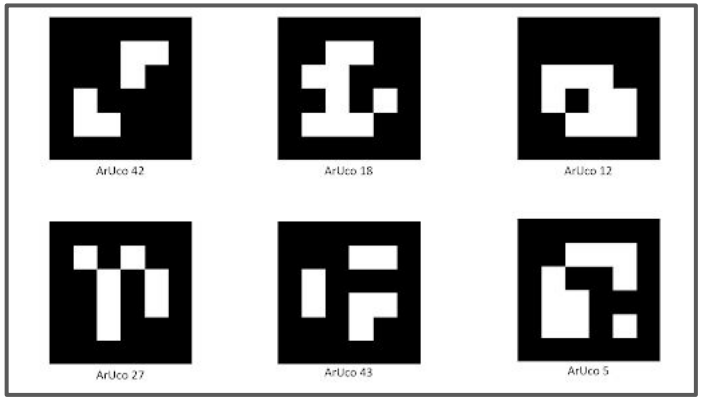
Tags: ArUco Markers

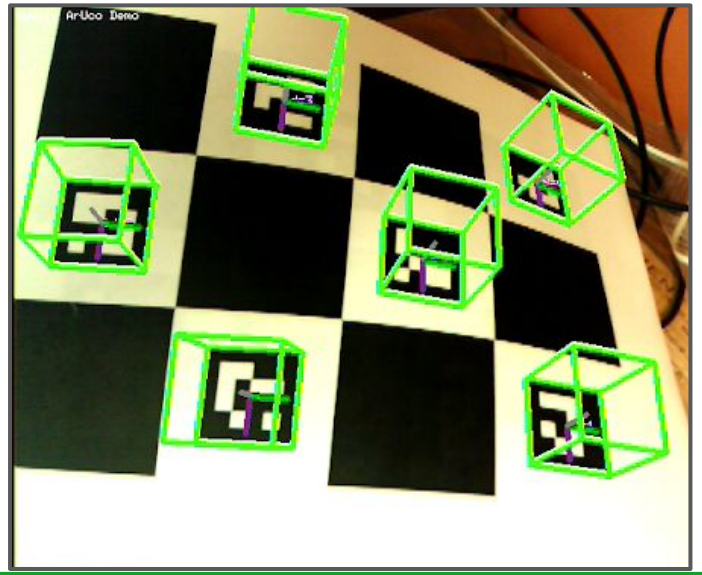
Certain standardized **‘tags’** are often used as artificial features.

Every tag has an ID.

Easier to identify tags uniquely as compared to naturally occuring features.

Being flat squares, easy to estimate distance and orientation.





Sensor Fusion

- Estimates from a single method are usually not reliable.
  Therefore, a combination of multiple methods needs to be used.

- Each method is given a weight based on the reliability of the sensor and the algorithm used.

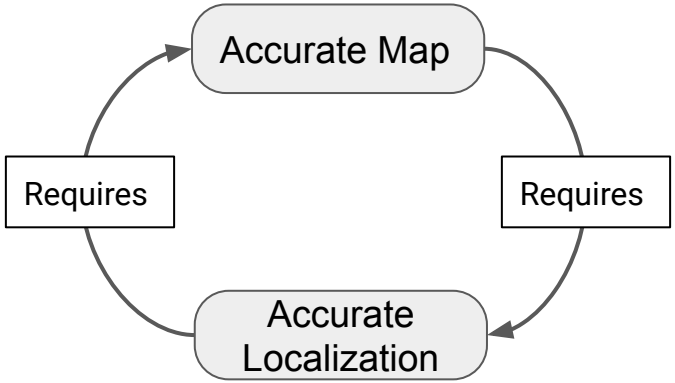
- An efficient method of dynamically deciding these weights is the **Kalman Filter.**

- A modified version of the Kalman Filter, known as **Extended Kalman Filter (EKF)** is widely used in localization of robots.

Simultaneous Localization and Mapping (SLAM)

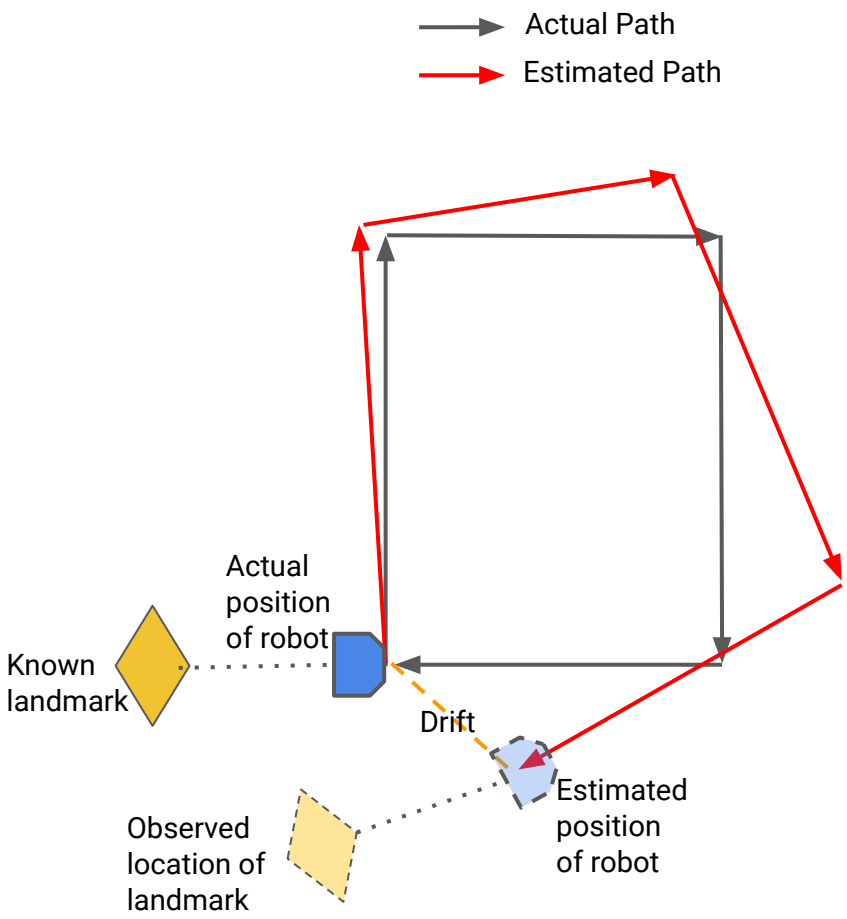
Definition: The process of mapping an unknown environment while simultaneously carrying out localization within the map

- Groups of features are stored as **‘landmarks’**. The location of these landmarks is stored in the map.
- The known map & landmarks are together used for localization
- A reliable position is in turn used to update and correct the map



Loop Closure

- When a robot moves around and returns to its original location, the calculated displacement is always more than zero, due to drifts.
- If, in such a case, a robot observes a known landmark, it can immediately reset its position accordingly.



Next class -

- Turtlebot navigation stack
- gmapping
- move_base
- RViz