Session 1: 2D Path Planning, its importance and its application

Report Authored By: Antriksh Singh Rathore, Kshitij Gupta

Faculty Coordinator: Dr. Neelam Srivastava Ma'am, ECE Dept. IET Lucknow

Mentees of the Session : Antriksh Singh Rathore, Kshitij Gupta, Abhinav Kumar, Anindya Ranjan Samaddar, Ashish Singh, Divyansh Jaiswal, Gaurav Chaurasiya

Mentors of the Session: Aastha Chauhan, Anshuman Singh, Saksham Gautam, Keshav Gupta

Email : antrikshrs28@gmail.com , kshitijgupta084@gmail.com

March 9, 2023

DAY 1

What is 2D Path Planning ?

The **computation of an ideal or practical path for a robot or item to navigate in a two-dimensional** (or even three-dimensional) space is called path planning.

What will be the future of 2D path planning?

As a result of continuous study and technical progress, 2D path planning is anticipated to experience improvements in a number of areas in the future. Here are some prospective developments and trends:

1.) Collision Avoidance:

To ensure that robots or **objects can navigate without colliding with other** items or obstacles in a specific area, 2D path planning's main goal. Path planning algorithms compute safe and obstacle-free courses, which helps to avoid collisions, lessen damage to the robot or its surroundings, and improve overall safety.

2.) Autonomous Navigation:

For autonomous robots and vehicles to successfully **explore challenging areas without human assistance**, path planning is crucial. These systems can perform tasks like autonomous exploration, surveillance, delivery, or transportation by developing optimal pathways that allow them to work effectively, avoid impediments, and arrive at their objectives.

3.) Real-World Applications:

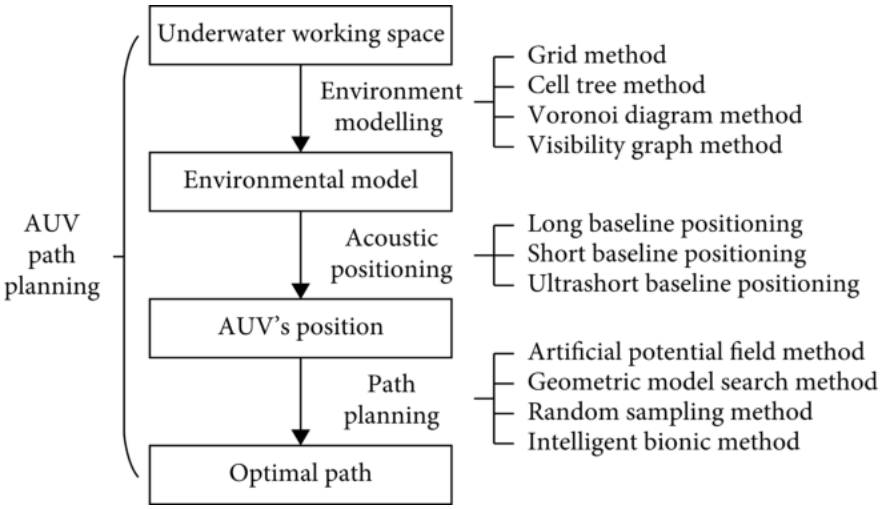
Aerial surveillance, robotics, **manufacturing, logistics, warehouse automation, agriculture, search and rescue operations**, and other fields are just a few of the real-world uses for 2D path planning. It makes it possible to move in complicated and dynamic situations safely and effectively, increasing output, lowering labour requirements, and opening up new possibilities.

4.) Virtual Environments and Simulations:

2D route planning is used to **model the mobility of actors, objects, or entities in computer graphics and virtual simulations.** In video games, virtual reality (VR), augmented reality (AR), or virtual training simulations, path planning algorithms allow realistic and natural-looking movement, interaction, and navigation within the virtual environment.

What are the methods of 2D path planning?

There are several **methods and algorithms for 2D path planning**. Here are some commonly used approaches:



1.) Grid-Based Methods:

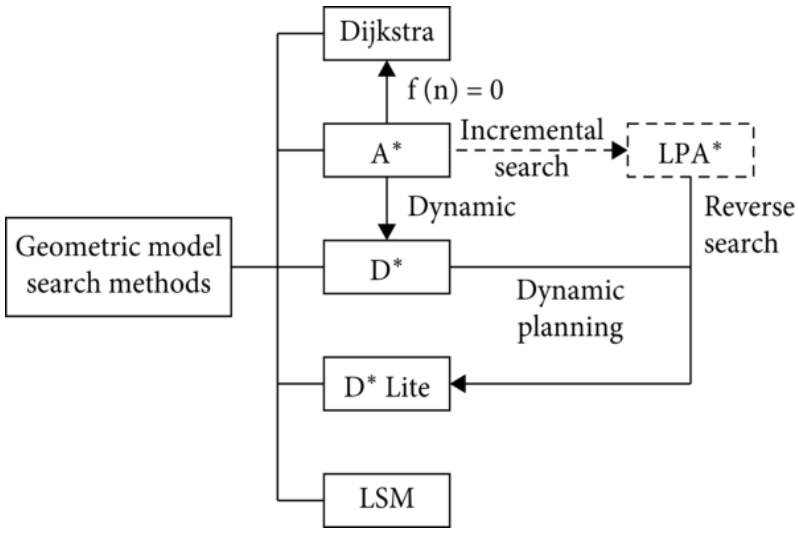
Grid-based algorithms divide the environment into a grid of cells, where each cell represents a portion of the space. These methods include:

- **Dijkstra's Algorithm:**

  It is a graph search algorithm that computes the shortest path from a start point to a goal point by exploring the grid cells.

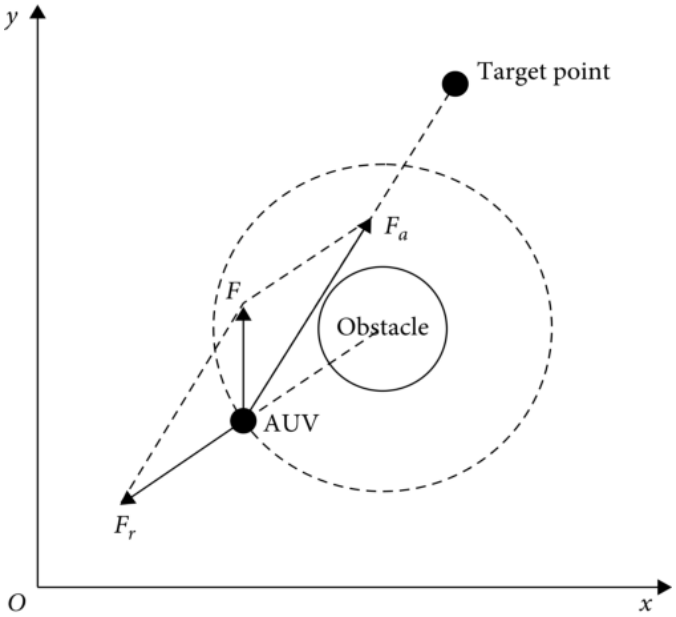
- **A* Algorithm:**

  A* combines Dijkstra's algorithm with heuristics to improve efficiency. It uses a cost function that considers both the path cost from the start point and an estimate of the remaining cost to the goal. A* is widely used for optimal pathfinding.



2.) Potential Field Methods:

Potential field algorithms represent the environment as a field where attractive forces guide the robot towards the goal, while repulsive forces repel it from obstacles. These methods include:

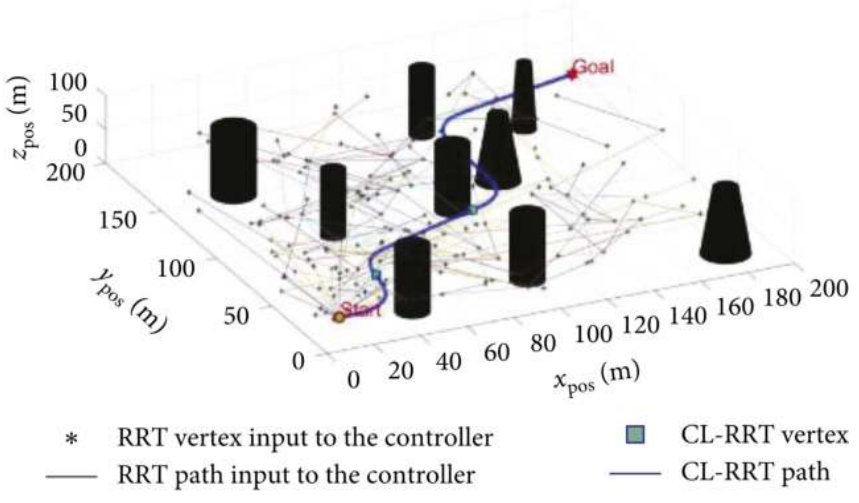


- **Artificial Potential Field (APF):** APF assigns attractive potentials to the goal and repulsive potentials to obstacles. The robot moves along the gradient of the potential field towards the goal while avoiding obstacles.
- **Elastic Band Method:** This method represents the path as a continuous elastic band. The band is deformed to avoid obstacles, and the robot follows the deformed band as it moves towards the goal.

3.) Visibility Graph Methods:

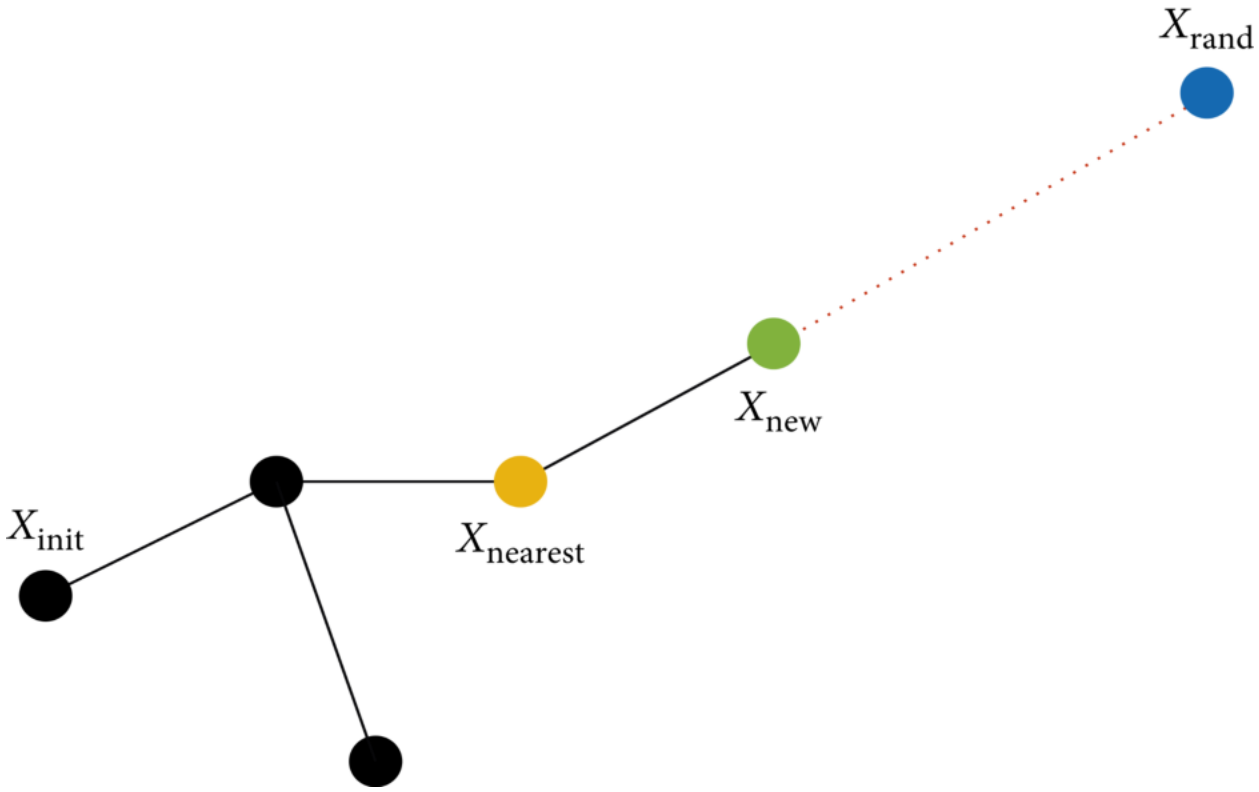
Visibility graph algorithms construct a graph by connecting visible points in the environment and finding the shortest path on this graph. These methods include:

- **Voronoi Diagram:** Voronoi diagrams divide the space into regions based on proximity to obstacles. The paths between Voronoi vertices represent feasible paths.
- **Tangent Bug Algorithm:** The Tangent Bug Algorithm uses the visibility graph and incremental movements to navigate around obstacles. It moves along the obstacle boundary until a point on the visibility graph is reached.



4.) Sampling-Based Methods:

Sampling-based algorithms randomly sample the configuration space and build a roadmap to search for feasible paths. These methods include:



- **Probabilistic Roadmap (PRM):** PRM samples random points in the environment and connects them to build a graph. It then performs graph search algorithms to find a path between the start and goal points.
- **Rapidly Exploring Random Tree (RRT):** RRT incrementally builds a tree structure by extending the tree towards randomly sampled points. It rapidly explores the configuration space and finds a feasible path.

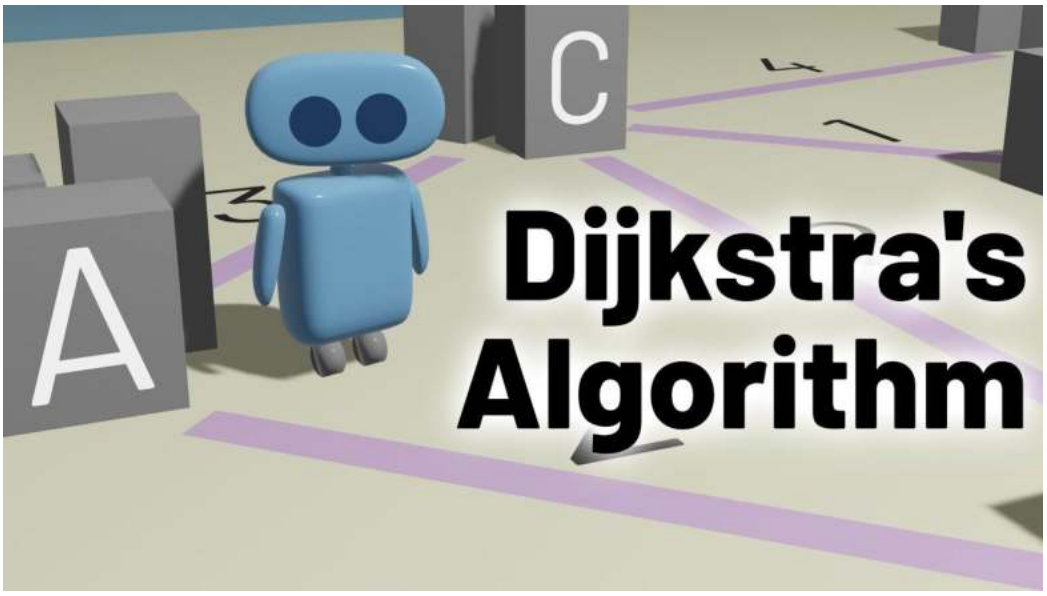
These are just a few examples of the methods used in 2D path planning. Each method has its strengths and weaknesses, and the choice of algorithm depends on the specific requirements of the application, the complexity of the environment, and the desired performance characteristics.

5.) Conclusions:

During the session on 2D path planning, we explored the significance of path planning and discussed various methods. We highlighted grid-based approaches such as Dijkstra and A*, which **divide the environment into a grid and find the optimal path** based on distance or heuristics. Additionally, we covered potential field methods like elastic bands and artificial potential fields that **leverage attractive and repulsive forces to guide the path.** Visibility graph techniques, including Voronoi and tangent-based methods, **utilize visibility relationships between points to determine feasible paths**. Lastly, we delved into sampling-based methods such as Probabilistic Roadmaps (PRM) and Rapidly-Exploring Random Trees (RRT), which **employ random sampling to generate a roadmap or tree structure for path planning.** These methods offer diverse approaches to tackling the challenges of 2D path planning.

What will we do in the next session?

In the next session, we will understand more about Dijkstra's algorithm and its application.



References

1. www.willowgarage.com , www.thingsinsquares.com for reference images
2. How Dijkstra's Algorithm Works - YouTube Video reference

DAY 2: Dijkstra's Algorithm & Its Application **March17,2023**

Overview:

Dijkstra's algorithm is a popular graph traversal algorithm used to find the shortest path between a starting node and all other nodes in a graph. It was developed by Dutch computer scientist Edsger W. Dijkstra in 1956. The algorithm is commonly used in various applications, including network routing protocols, GPS navigation systems, and graph analysis.

Objective:

- What is the importance of the Dijkstra Algorithm?
- What are the methods of the Dijkstra Algorithm?
- What are the applications of the Dijkstra Algorithm?

Importance Of Dijkstra Algorithm

Dijkstra's algorithm is highly relevant and important in the field of robotics, particularly in the domain of path planning and navigation. Here are some reasons why Dijkstra's algorithm holds significance in robotics:

1.	**Path Planning:**
	In robotics, path planning involves determining a safe and efficient path for a robot to navigate from a starting point to a goal location while avoiding obstacles. Dijkstra's algorithm can be employed to find the shortest path, taking into account obstacles and terrain information encoded in a graph representation of the environment.
2.	**Real-time Navigation:**
	Real-time navigation is crucial for robots to autonomously move in dynamic environments. Dijkstra's algorithm, with appropriate optimizations and data structures, enables efficient and quick calculation of the shortest path.
3.	**Multi-Robot Systems:**
	Dijkstra's algorithm is valuable in multi-robot systems, where multiple robots need to coordinate their paths to avoid collisions and congestion. By calculating the shortest paths for each robot individually, potential conflicts can be resolved, and efficient routes can be planned for the entire robotic system.
4.	**Simplicity and Understandability:**
	Dijkstra's algorithm has a straightforward and intuitive implementation, making it easily understandable and implementable in robotic systems. Its simplicity aids in rapid prototyping, testing, and debugging of navigation algorithms.
5.	**Integration with Sensor Data:**
	Dijkstra's algorithm can be combined with sensor data from various sources, such as cameras, lidar, or depth sensors, to create a more informed graph representation of the environment.

Method Of Dijkstra Algorithm

In robotics, the implementation of Dijkstra's algorithm for path planning and navigation typically involves several key methods. Here are the main steps involved in applying Dijkstra's algorithm in the robotics domain:

1. **Graph Representation:**
   First, the environment in which the robot operates needs to be represented as a graph. The nodes of the graph represent the locations or cells in the environment, and the edges represent the connections between them. The graph can be constructed based on a grid-based representation or a connectivity-based representation, depending on the specific application.
2. **Initialization:**
   Initialize the graph by assigning a distance value to each node. Set the distance of the starting node to 0 and the distance of all other nodes to infinity. Keep track of the visited and unvisited nodes.
3. **Iterative Process:**
   The algorithm proceeds iteratively until all nodes have been visited or the goal node has been reached. In each iteration, the following steps are performed:
   a. Select the node with the smallest distance among the unvisited nodes. This node becomes the current node.
   b. For each neighboring node of the current node:
      - Calculate the tentative distance from the starting node to the neighboring node by adding the weight of the edge connecting them.
      - If the tentative distance is smaller than the previously recorded distance, update the distance and set the current node as the previous node for the neighboring node.

   c. Mark the current node as visited and remove it from the unvisited nodes.
4. **Termination:**
   Once all nodes have been visited or the goal node has been reached, the algorithm terminates. The shortest path from the starting node to each node in the graph is determined based on the recorded distances and previous nodes.
5. **Path Extraction:**
   To obtain the actual path the robot should follow, starting from the goal node, follow the recorded previous nodes back to the starting node. This process retrieves the sequence of nodes that form the shortest path.

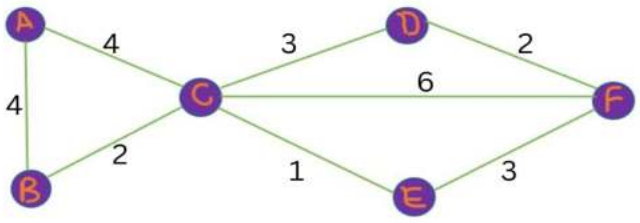
Explanation of Dijkstra's algorithm for finding the shortest path between one vertex in a graph:
Let distance of start vertex from start vertex=0

Let distance of all other vertex from start =∞(infinity)

Repeat

- ✔ Visit the unvisited vertex with the smallest known distance from the start vertex
- ✔ For the current vertex, examine its unvisited neighbours
- ✔ For the current vertex , calculate distance of each neighbour from start vertex
- ✔ If the calculated distance of a vertex is less than the known distance , update the shortest distance
- ✔ Update the previous vertex for each of the updated distances

✔ Add the current vertex to the list of visited vertices until all vertices visited

Dijkstra		
		
Node	Distance from A	Previous Node
A(start)	0	-
B	4	A
C	4	A
D	7	C
E	5	C
F(end)	8	E
A-C-E-F		

Applications Of Dijkstra Algorithm

Dijkstra's algorithm finds significant application in robotics, particularly in the areas of path planning, navigation, and obstacle avoidance. Here are some specific applications where Dijkstra's algorithm is commonly used in robotics:

1. **Robot Path Planning:**
   Dijkstra's algorithm is widely employed to find the shortest path for a robot to navigate from a starting point to a goal location in a known environment. By considering the costs or weights associated with different regions or edges, the algorithm determines the optimal path that minimizes travel time, energy consumption, or other relevant factors.
2. **Robot Navigation in Grid-based Environments:**
   Dijkstra's algorithm is often utilized in grid-based environments, where the robot operates on a discretized grid. It can efficiently determine the shortest path while avoiding obstacles represented as occupied grid cells. This application is commonly used in mobile robotics, such as autonomous robots navigating in indoor environments or drones planning paths in grid-like structures.
3. **Autonomous Ground and Aerial Vehicles:**
   Dijkstra's algorithm is employed in the navigation systems of autonomous ground and aerial vehicles. The algorithm considers the terrain, dynamic obstacles, and other constraints to generate safe and optimal paths.

4. **Real-time Obstacle Avoidance:**
   Dijkstra's algorithm can be used in combination with sensor data, such as lidar or depth sensors, to perform real-time obstacle avoidance. By continuously updating the graph representation based on the sensor readings and recalculating the shortest path.
5. **Robot Localization and Mapping:**
   Dijkstra's algorithm is utilized in simultaneous localization and mapping (SLAM) algorithms to optimize the estimated robot pose and map. It assists in determining the robot's location and building an accurate map of the environment based on sensor measurements and estimated paths.

These are just a few examples highlighting the wide-ranging applications of Dijkstra's algorithm in robotics. Its versatility, efficiency, and ability to handle complex environments make it a valuable tool for enabling safe and efficient robot navigation and path planning in various robotic systems.

Overview For Next Session:

In next session it will discuss about 2D path planning, line follower robot and Introduction To PID Controller

Resources :

Author:

Divyansh Jaiswal

Electronics And Communication Engineering (2nd Year)

Email: divyanshjai95@gmail.com

LinkedIn: https://www.linkedin.com/in/divyansh-jaisawal-8bb504252

Gaurav Chaurasiya

Mechanical Engineering (2nd Year)

Email :

LinkedIn :

Mentors:

Aastha Chauhan, Anshuman Singh, Saksham Gautam, Keshav Gupta

Electronics And Communication Engineering (3rd Year)

Email:

LinkedIn:

Faculty Coordinator :

Dr. Neelam Srivastava

H.O.D (ECE Department)

Institute Of Engineering And Technology , Lucknow

Session 3: 2D Path Planning and Line Follower Robot

Report Authored By: Anindya Ranjan Samaddar, Abhinav Kumar

Faculty Coordinator: Dr. Neelam Srivastava Ma'am, ECE Dept. IET Lucknow

Mentees of the Session : Antriksh Singh Rathore, Kshitij Gupta, Abhinav Kumar, Anindya Ranjan Samaddar, Ashish Singh, Divyansh Jaiswal, Gaurav Chaurasiya

Mentors of the Session: Aastha Chauhan, Anshuman Singh, Saksham Gautam, Keshav Gupta

Email : anindyarsam@gmail.com

April 28, 2023

DAY 3

Overview of the session:

Earlier sessions covered the importance of 2D path planning, its different algorithms, and the Dijkstra algorithm in particular.

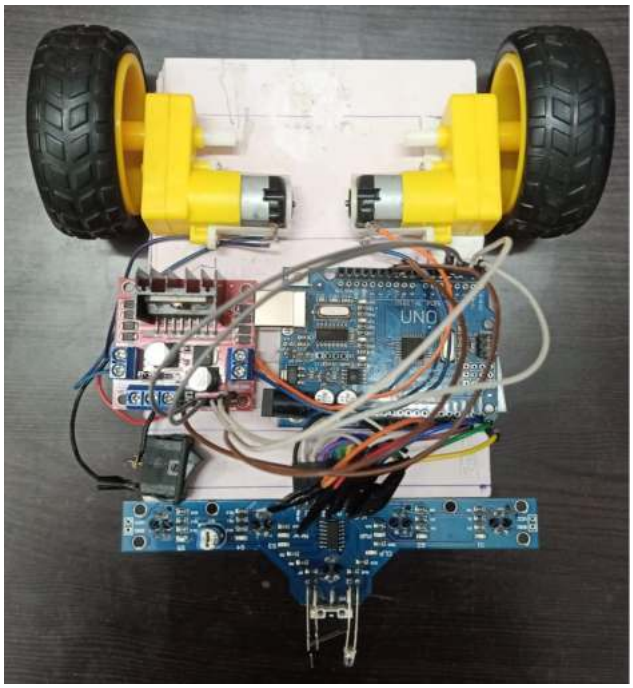
In this session, as part of our ongoing effort to fully comprehend path planning and use it in real-world scenarios, we focused on the hardware design element of mobile 2D path planning.

This session's objectives were to teach students how to construct a Line Follower Robot and to help them understand its fundamental concepts; the subsequent session would let them incorporate 2D path planning into it.

Objectives:

1. Explore the design and implementation of a Line Follower Robot (LFR).
2. Discover the PID controller's operation and its guiding principles.
3. Gain practical experience in wiring, connections, and code development of the hardware system.

Line Follower Robot (LFR):

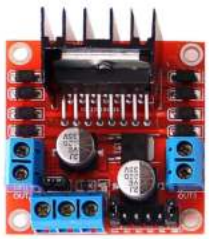


INTRODUCTION:

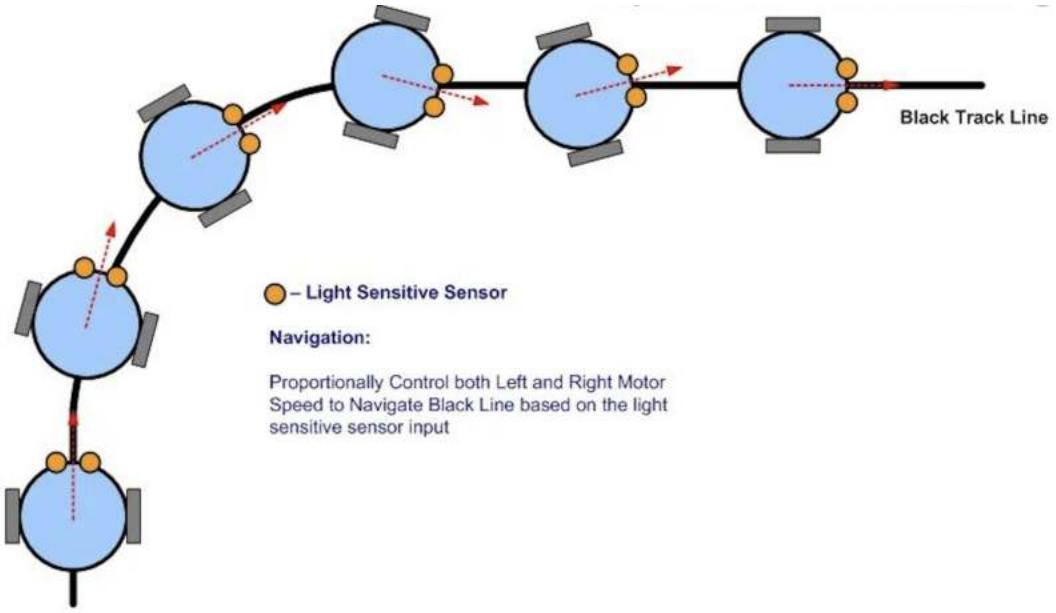
A line follower is a type of mobile robot that uses sensors to detect and follow a line or path marked on the ground. The robot's control system processes sensor data to adjust its movements, keeping it on track. Line followers are often used for educational purposes, teaching programming and robotics concepts. They can also be used in industries such as automation, logistics, and surveillance.

COMPONENTS USED:

- Arduino Uno
- L298N motor driver
- Plastic gear motor
- Array of 5 IR sensor
- Li-ion battery 2000MAh
- Jumper wires
- Switch



WORKING PRINCIPLE:



Line Tracking Navigation Principle on The Line Follower Robot (LFR)

1. IR Sensor Detection:

- The LFR is equipped with **infrared (IR) sensors** placed close to the ground.
- The sensors detect the **contrast between the line and the surface**.
- When the sensors are over the line, they receive less reflected light compared to when they are over the surface.

2. Sensor Readings:

- Each IR sensor generates analog or digital signals **based on the amount of reflected light** it receives.
- The sensor readings **provide information** about the **position of the line relative to the sensors.**

3. PID Controller:

- The PID controller is a **feedback control algorithm** used to adjust the robot's movements based on the error between the desired position (on the line) and the actual position (sensor readings).

- It consists of three components: **proportional, integral, and derivative.**
- The PID controller calculates control signals that regulate the robot's speed, direction, and turning.

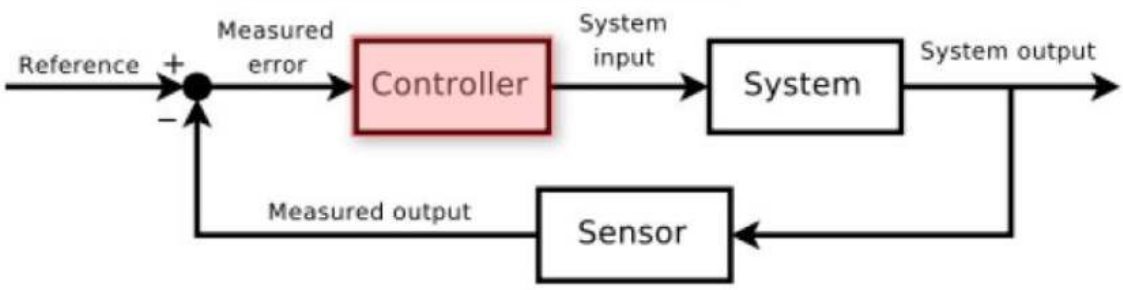
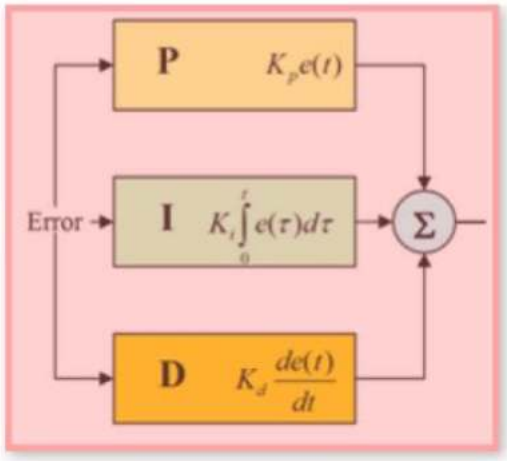


FIG - GENERAL STRUCTURE OF PID CONTROLLER

➢ **Error Calculation and PID Tuning:**

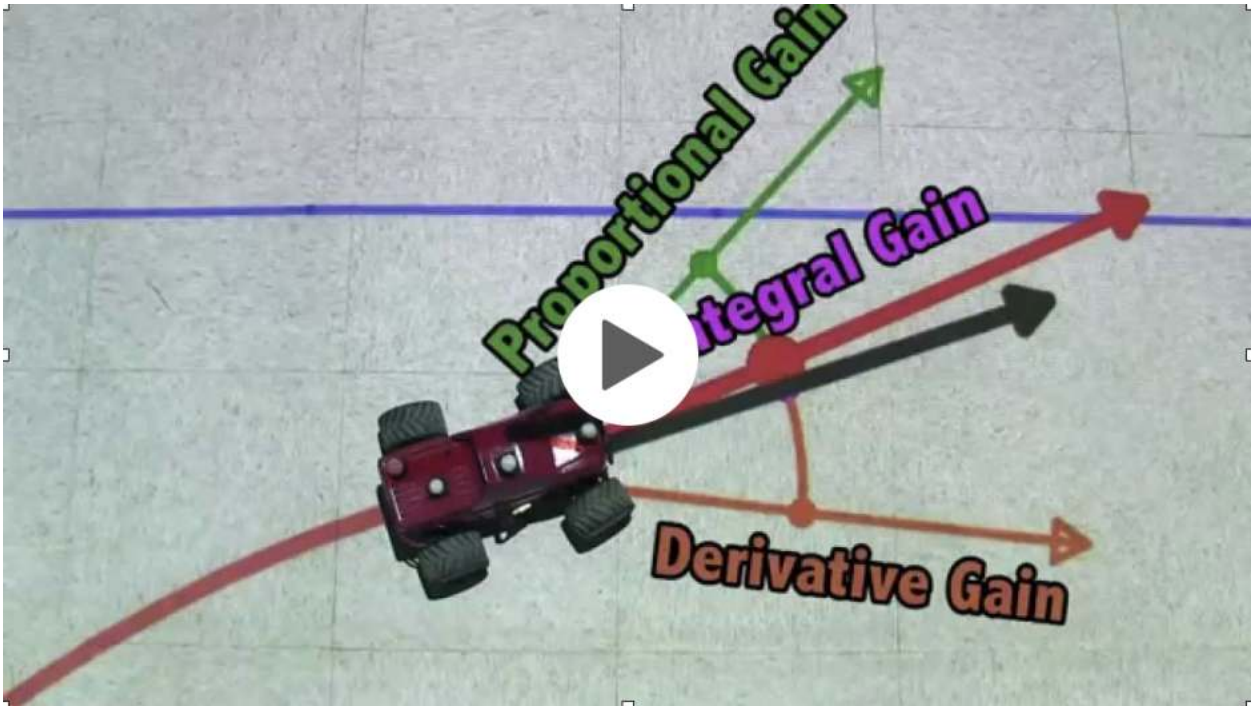


- The PID controller compares the **desired position** (on the line) with the **actual position** (sensor readings) to calculate the **error**.
- The error represents the deviation of the robot's position from the desired position.
- We have used the **Ziegler-Nichols method** to tune the PID controller.

➢ **PID Control Signal Calculation:**

- The PID controller uses the error to calculate control signals that adjust the robot's movements.
- The **proportional component** contributes to **immediate adjustments** based on the **current error**.
- The **integral component** accounts for **cumulative errors** over time, addressing any steady-state errors.

- The **derivative component** anticipates **future errors** based on the rate of change of the error.



Understanding the PID Control

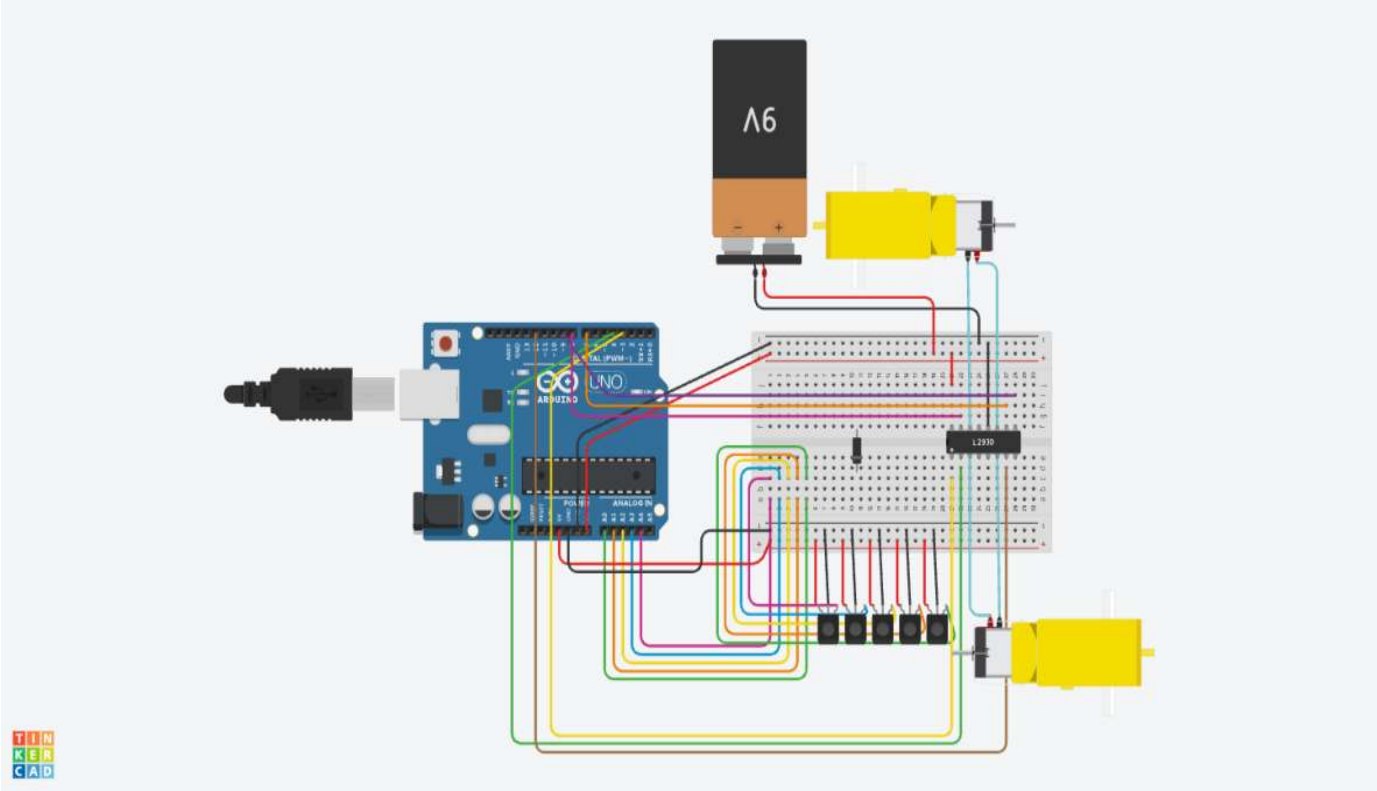
4. Adjusting Movements:

- Based on the PID control signals, the LFR **adjusts its motor speeds** or **steering mechanisms to correct its position**.
- For example, if the robot detects that it is deviating from the line to the right, it might decrease the speed of the right motor or increase the speed of the left motor to steer back onto the line.

5. Continuous Monitoring and Adjustment:

- By **continuously monitoring** the **IR sensor readings**, **calculating control signals** using the PID controller, and **adjusting its movements** accordingly, the LFR can **accurately follow a line** or path on the ground.

BLOCK DIAGRAM:



ARDUINO CODE:

To view the code, click the link:

https://create.arduino.cc/editor/hexacom06/5bba2d8e-742e-4287-afd3-ac220c626f6d/preview

Code brief:

1. ***Initialization:*** In the **setup()** function, the code sets up the necessary pins.

2. ***Sensor Readings:*** In the **loop()** function, the code reads the *analog sensor values from five different pins (A0 to A4)* using the **analogRead()** function. The sensor values represent the intensity of the reflected light.

3. ***Digital Conversion:*** The *analog sensor readings are compared to a threshold value (thres)*. If a *sensor reading is above the threshold*, the corresponding element in the a_digital array is *set to 1*, indicating **a white surface**. Otherwise, it is *set to 0,* representing **a black surface.**

4. ***Error Calculation:*** Based on the digital sensor readings, the code determines the *current position of the line relative to the center of the robot*. This is done by evaluating the combinations of black and white surfaces detected by the sensors. The resulting *error value represents the position error of the robot with respect to the line.*

5. ***PID Calculation:*** The code applies a **PID (Proportional - Integral - Derivative) control algorithm** to compute a control signal based on the error value. The *proportional, integral, and derivative terms (KP, KI, and KD, respectively) are multiplied by their respective error components (error, i, and d) to calculate the PID value*.

6. ***Motor Control*:** The *motor speeds are adjusted based on the PID value.* The speed_left and speed_right variables are calculated by adding or subtracting the PID value from a base speed of 100.

7. ***Motor Output:*** The **analogWrite()** function is used to *control the motor speeds,* and the **digitalWrite()** function sets the appropriate pins to *control the motor directions*.

8. ***Serial Output:*** The *analog sensor readings* are printed to the *serial monitor* using the **Serial.print()** function for monitoring and debugging purposes.

CONCLUSION:

The workshop provided an extensive overview of the LFR with a PID controller. It provided useful insights into the operating principle and control mechanism of this autonomous robot. The LFR proved to be a useful and dependable solution for autonomous line following in a range of conditions by employing infrared (IR) sensors to detect the line and a PID controller to manage its motions. The LFR demonstrated effective line-following capabilities.

Here are some clicks from the session:

What will we do in the next session?

In the next session, we will learn how to **Integrate the Path Planning Logic and the Line Following Logic.**

Reference Links:

Here are a few links to quench your craving for more knowledge:

1. More on PID control:
Understanding PID Control - YouTube

2. Similar LFR project on Arduino:
Line Follower Robot (with PID controller) | Arduino Project Hub

3 .Embedded video on PID Control
Controlling Self Driving Cars
