SPRING 2024

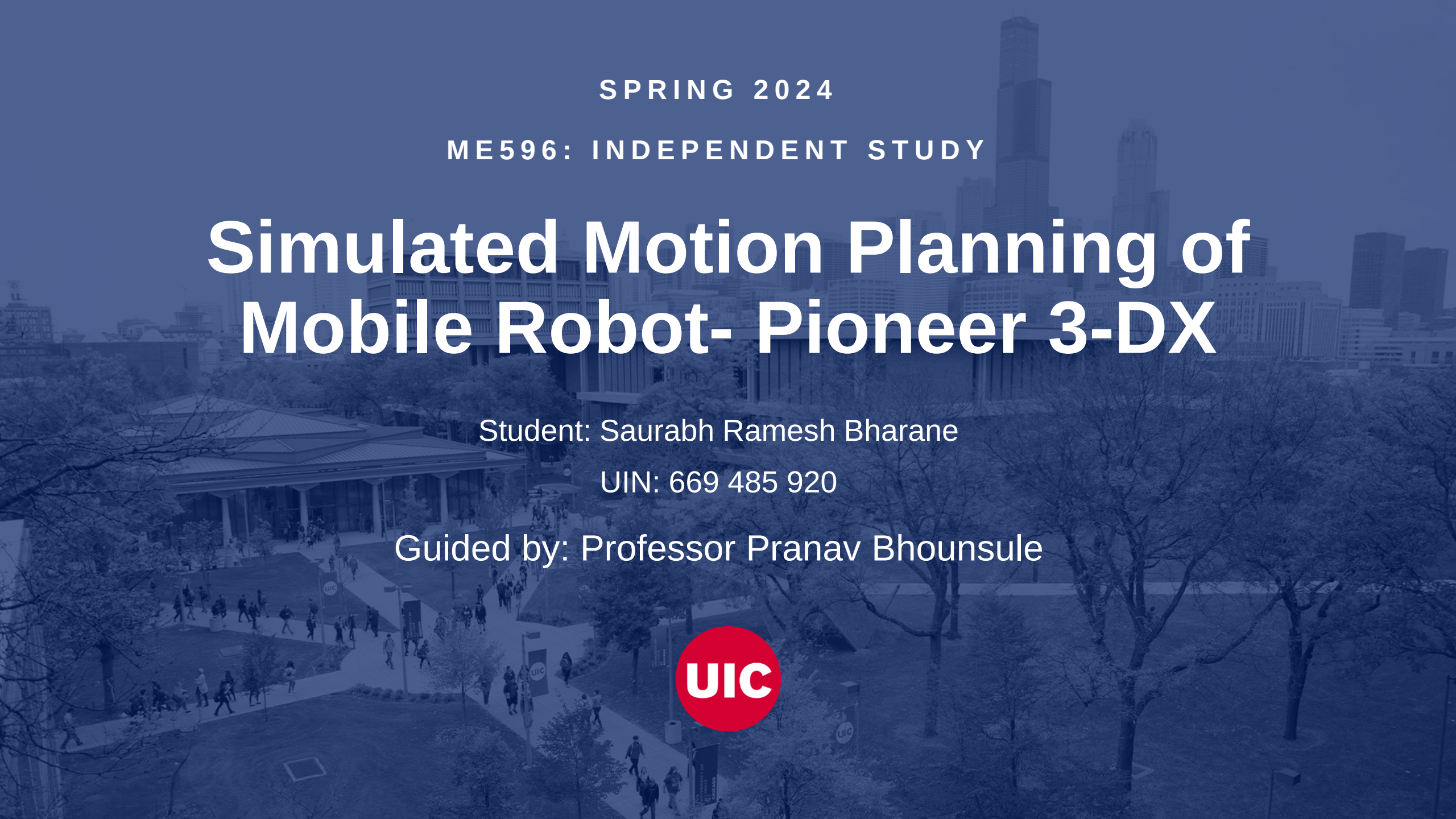
ME596: INDEPENDENT STUDY

Simulated Motion Planning of Mobile Robot- Pioneer 3-DX

Student: Saurabh Ramesh Bharane

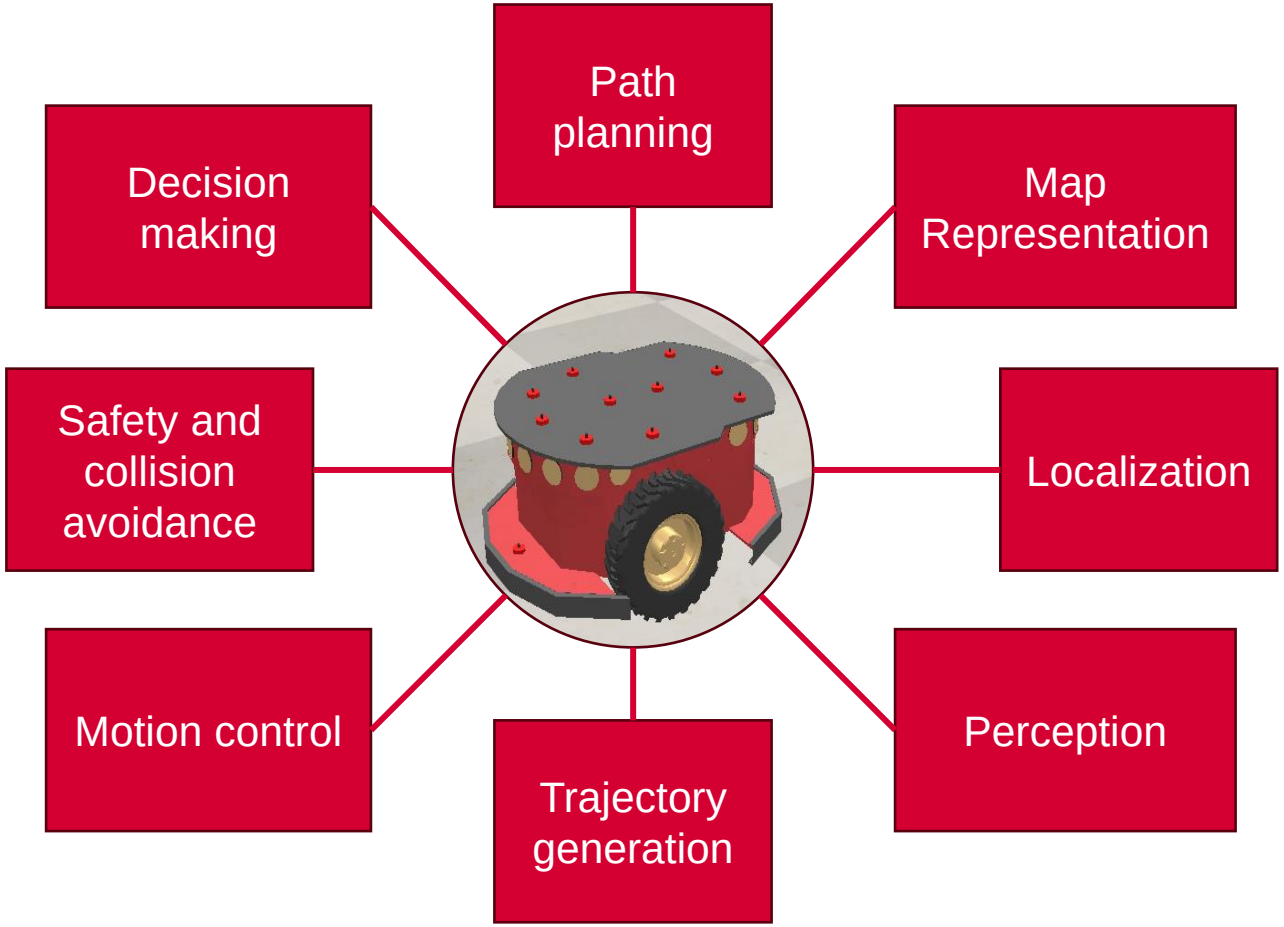
UIN: 669 485 920

Guided by: Professor Pranav Bhounsule



Introduction

Elements of autonomous motion planning



Introduction - Elements of the project

CoppeliaSim

Simulator for executing the simulation of the Pioneer 3-DX

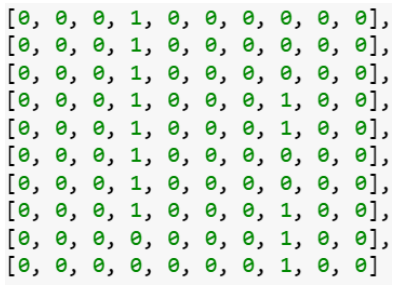
Python

Coding Language

A Star

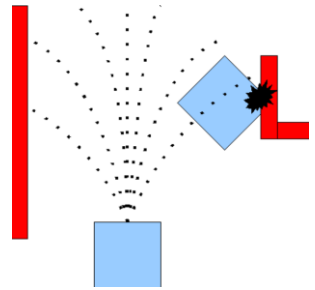
Path planning Algorithm

(Global path planning)



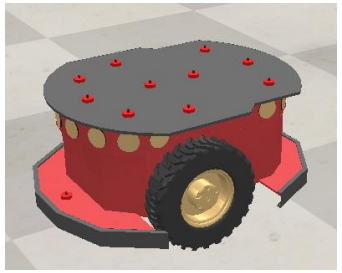
Occupancy Grid

Environment representation using the occupancy Grid



Dynamic Window Approach

Local Path Planning



Pioneer 3-DX

Mobile Robot

Introduction – Overview of work

1. This presentation gives information of the work done in the project.
2. It includes step-by-step explanation to execute and modify the project.
3. The following modules have been explained in the subsequent slides.

- Initial setup of CoppeliaSim and python
- Execute the project using python script.
- Environment setup
- A Star global planner overview and code explanation
- Localization commands
- Trajectory generation
- Motion control
- DWA Local planner
- Computer vision
- Future work

Initial Setup

I have provided following folders with the package.

1. CoppeliaSim setup guidance
2. Project work
3. Individual module development.
4. Google Drive link for complete Project is provided at the last Slide.

Initial Setup

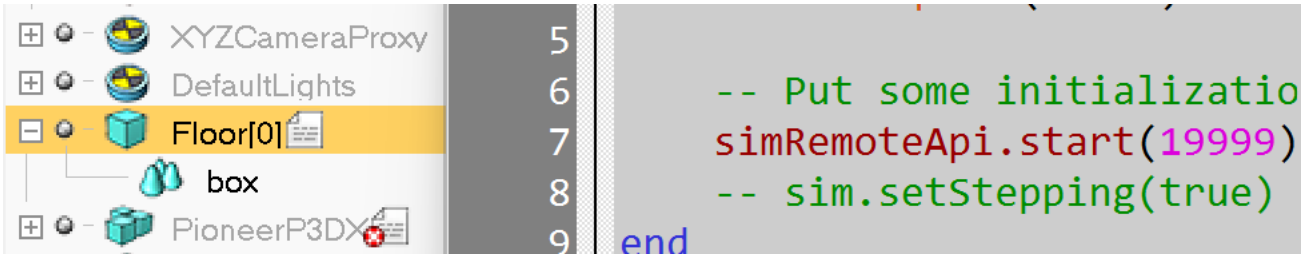
1. Download latest version of the CoppeliaSim and the Python.
2. Enable Python in the CoppeliaSim by providing the python path. Follow instructions in the file "**setting_python_path**" provided in the folder "**CoppeliaSim_setup_guidance**". This needs to be done **once** per system.
3. I have provided **Step 4** files in the project work so; you can skip that step and move on to the next slide. If you want information for the initial setup, you can look at Step 4.

4. To control CoppeliaSim externally using python, use remote API Bindings provided by the CoppeliaSim. Use "**CoppeliaSim Remote API binding.pdf**" provided in folder "**CoppeliaSim_setup_guidance\Set_Coppelia_External_Python Basics_Pioneer_Control**" for setup details. This task must be completed for each new project. Mentioned PDF gives names and location of files to be copied in the project folder.

Execution of the Project

Running the project file

- Open CoppeliaSim and open scene “Scene_Motion_planning_P3DX” provided in folder “Project work\Motion Planning Execution”.
- Python connection command “simRemoteApi.start(19999)” is added in the “Floor” script.

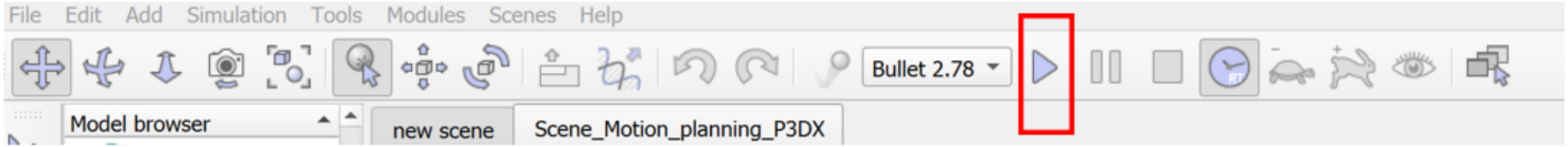


- PioneerP3DX default script is disabled.
- Open “Command prompt” or “Spyder” for code execution.
- Scene “Scene_Motion_planning_P3DX.ttt”, Code “Code_Motion_planning_P3DX.py” and Python API bindings for Windows OS and MacOS are copied in the folder “Project work\Motion Planning Execution”. For details refer slide 3 - point 4.

Execution of the Project

Running the project file

- Run scene “Scene_Motion_planning_P3DX”.



- Run the Code “Code_Motion_planning_P3DX.py” using Spyder or Command Prompt.
- Use following command to run code using command prompt

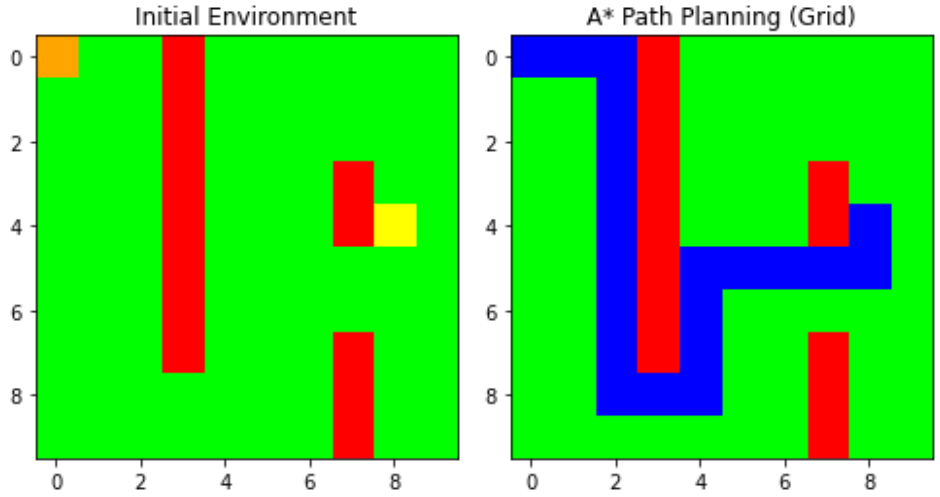
  python <replace with path>\ Motion Planning Execution \ Code_Motion_planning_P3DX.py
- This will start executing the motion planning algorithm and Pioneer 3DX will start moving to reach the goal based on optimized path.

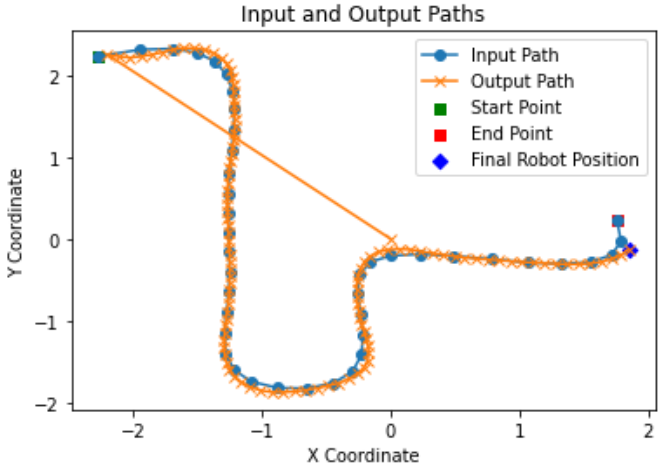
Execution of the Project

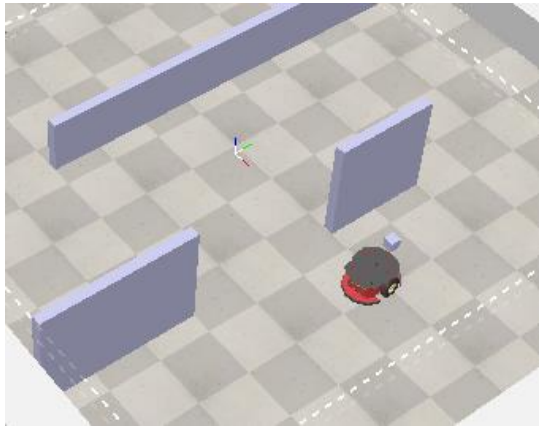
Output of the execution of code

- Based on Spyder or Command prompt. The code will generate following filles and output.
- Optimal Path (Grid) and Optimal Path (World Coordinates) to be followed by the Pioneer 3DX.
- Images for Map, Optimized path avoiding the obstacles and Path followed by the Pioneer 3DX.
- It will also generate the live co-ordinates of the Pioneer 3DX.
- At the end it will provide message “Reached the final goal!”.

```
Robot Position: [1.863982081413269,
Reached the final goal!
```



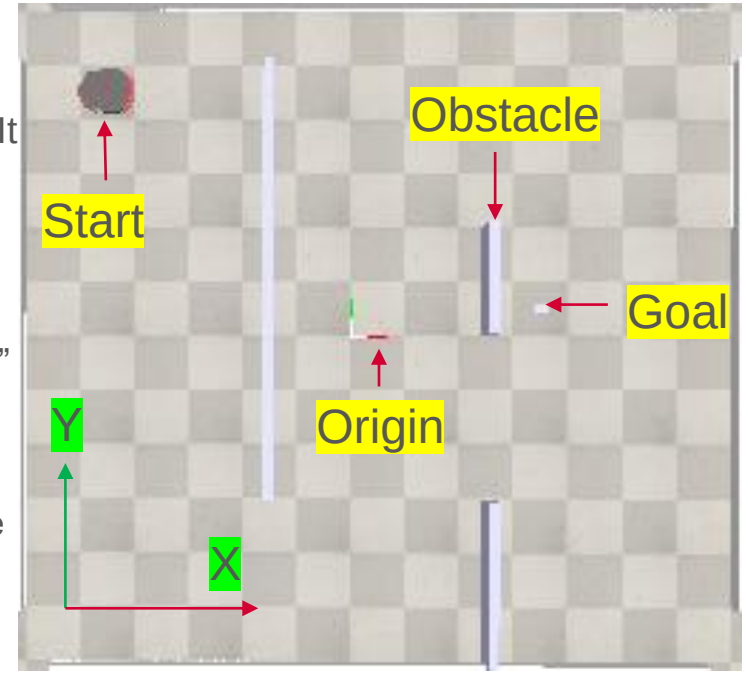




Motion planning code and algorithm explanation

Grid and Environment setup.

- Code "Code_Motion_planning_P3DX.py" is explained in this slide and in further slides. It is provided in the folder "Motion Planning Execution"
- Each block on floor is 0.5m x 0.5 m.
- Bocks with obstacles are represented by "1" and free space is represented by "0" in python code.
- Change position of "1" and "0" to modify the obstacle position.
- Change size of environment by changing Grid environment size and CoppeliaSim environment size.
- Change obstacles as needed.
- Change Start and Goal Position co-ordinates to modify start and goal.
- Provide start coordinates equal to the initial position of Pioneer P3DX.



CoppeliaSim (Environment)

```python
# Grid environment
Grid_environment = [
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
]
```

Representation of Environment in Python

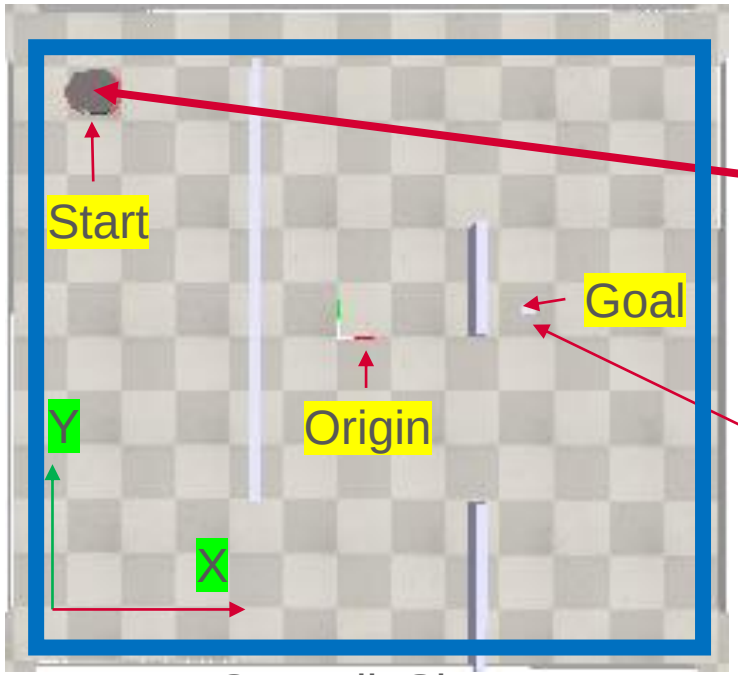
```python
# Start and goal positions
start = (0, 0)
goal = (4, 8)

# Grid size
grid_size = 0.5  # in meters
```

Motion planning code and algorithm explanation

Start and Goal setting.

- Pioneer 3DX is at Zeroth column and Zeroth row of the Grid_environment. So, start is (0, 0).
- Goal is at 4th row and 8th column so, Goal is (4, 8). Measure the row number and column number to set the start and goal. Set the axis of CoppeliaSim as shown in figure for correct orientation.

```
# Start and goal positions
start = (0, 0)
goal = (4, 8)

# Grid size
grid_size = 0.5  # in meters
```



CoppeliaSim (Environment)

Start
Row = 0th
Column = 0th

Goal
Row = 4th
Column = 8th

```
# Grid environment
Grid_environment = [
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
]
```

Representation of Environment in Python

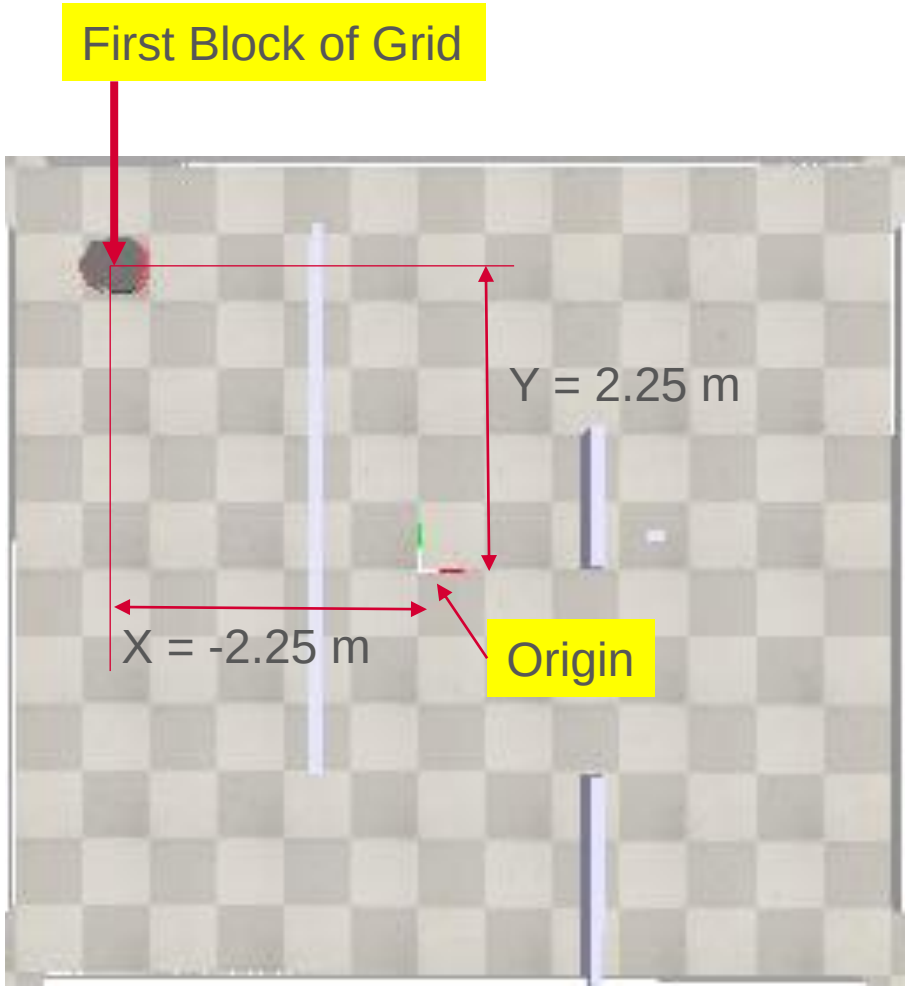
Motion planning code and algorithm explanation

Grid and Environment setup.

- The first block in the grid has coordinates (- 2.25, 2.25).
- Modify “grid_to_world” function whenever environment size changes. Put coordinates of the first block at “x” and “y” variables

```
# Convert grid coordinates to world
def grid_to_world(node):
    x = -2.25 + node[1] * grid_size
    y = 2.25 - node[0] * grid_size
    return x, y
```

- This function assigns world coordinates to the grids of “1” and “0”.

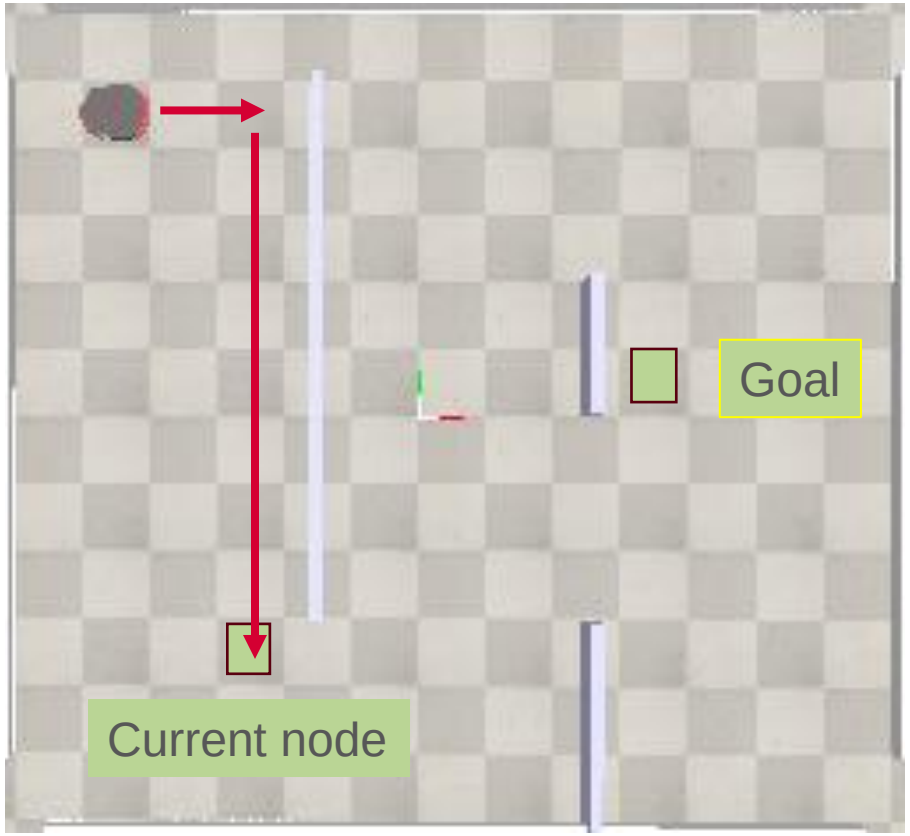


CoppeliaSim
(Environment)

Motion planning code and algorithm explanation

A-star search Algorithm.

- The A-star algorithm is searching algorithm used to find the shortest path between a start and the goal.
- It is optimal algorithm, as it searches for shorter path. It finds the least cost outcome for the problem.
- Important aspect of A* is f = g + h
- f is the total cost of the node and g is the distance between the current node and the start node. h is the heuristic - estimated distance from the current node to the end node.
- f value for all the neighboring nodes is calculated and the node with minimum total cost is selected.
- Set of such minimum cost nodes make the optimal path in the A-Star Search.



Environment

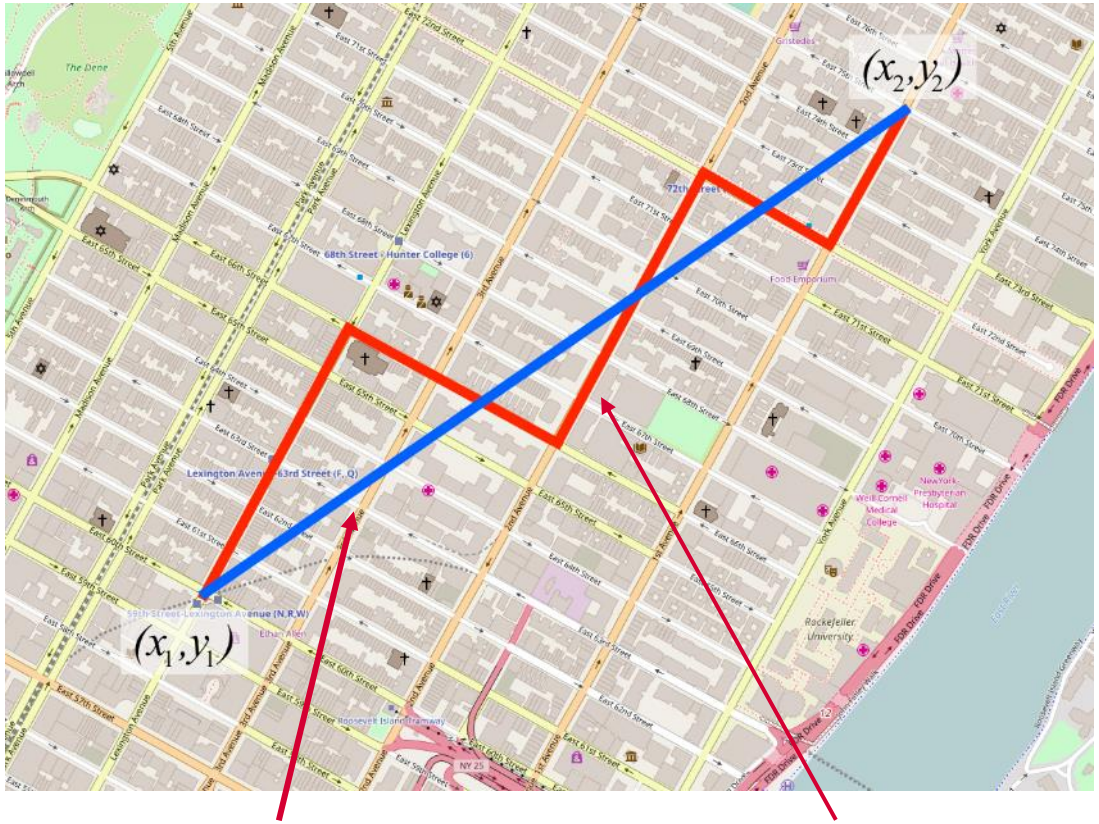
Motion planning code and algorithm explanation

A-star search Algorithm.

- Calculation of the heuristics – Two methods Exact and approximate. For current problem, Manhattan Distance, the approximate heuristics is selected to estimate distance from current node to goal node. As we have obstacles in the path. It is calculated by following formula.

$$|x_1 - x_2| + |y_1 - y_2|$$

- Manhattan distance, also known as taxicab distance or city block distance, is a measure of the distance between two points in a grid-based environment. It is calculated by following formula.
- This heuristics can be used when we can move in 4 directions- Up, Down, Left, Right.



Motion planning code and algorithm explanation

A-star search Code.

- Heuristic calculation code (Manhattan distance)

```
# Heuristic function (Manhattan distance)
def heuristic(node, goal):
    x1, y1 = grid_to_world(node)
    x2, y2 = grid_to_world(goal)
    return abs(x1 - x2) + abs(y1 - y2)
```

- Successor node generation

```
# Define the neighbors function
def get_neighbors(node):
    x, y = node
    neighbors = []
    if x > 0 and Grid_environment[x - 1][y] == 0:
        neighbors.append((x - 1, y))
    if x < 9 and Grid_environment[x + 1][y] == 0:
        neighbors.append((x + 1, y))
    if y > 0 and Grid_environment[x][y - 1] == 0:
        neighbors.append((x, y - 1))
    if y < 9 and Grid_environment[x][y + 1] == 0:
        neighbors.append((x, y + 1))
    return neighbors
```

Motion planning code and algorithm explanation

A-star search Code.

- A-Star function starts with these lines. This function can be replaced by another search algorithm as per requirement.

```
# Define the A* algorithm
def astar(grid, start, goal):
    open_set = []
    heapq.heappush(open_set, (0, start))
    came_from = {start: None}
    cost_so_far = {start: 0}
```

- The f cost, g cost ang h cost is given by these lines.

```
for neighbor in get_neighbors(current_node):
    new_cost = cost_so_far[current_node] + 1  # Assuming a cost of 1 for
    if neighbor not in cost_so_far or new_cost < cost_so_far[neighbor]:
        cost_so_far[neighbor] = new_cost
        priority = new_cost + heuristic(neighbor, goal)
        heapq.heappush(open_set, (priority, neighbor))
        came_from[neighbor] = current_node
```

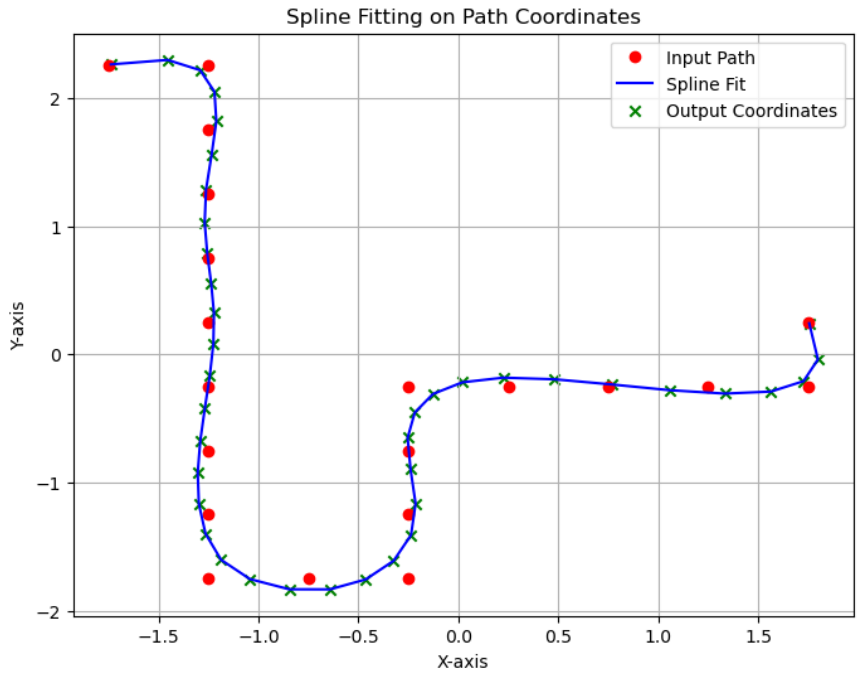
f = g + h

Motion planning code and algorithm explanation

Spline fit for smoothening of A star path.

- The path obtained by the A-star search have sharp curves, it is made rounder for smoother turns using spline fit. Change “num_points_between” and “s” to change degree of curvature. Red dots in the image are coordinates given by the A-star search and blue line is spline fit.

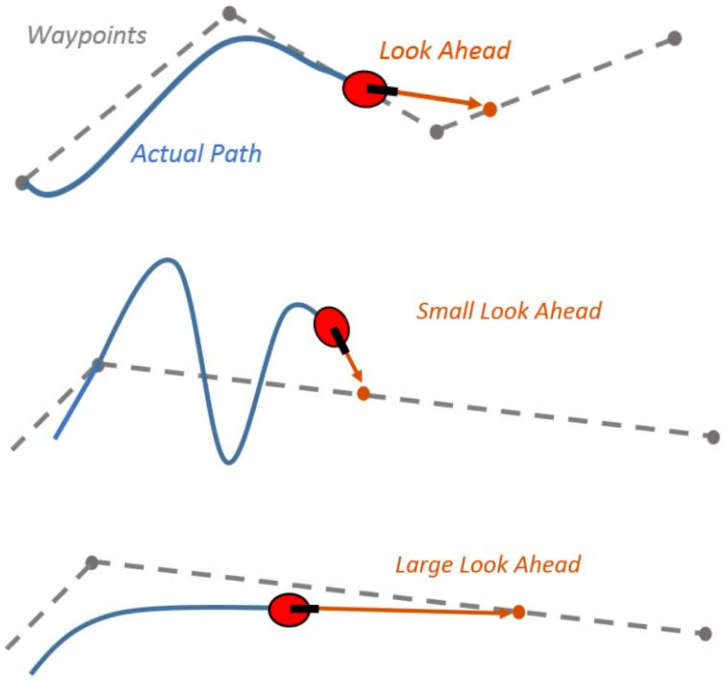
```python
#Spline fit for smoothening of A star path.
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import splprep, splev

# Input path coordinates
astar_path = np.array(optimal_path_coordinates)

# Number of points between successive input path coordinates
num_points_between = 2

# Extract x and y coordinates from the input path
x_input = astar_path[:, 0]
y_input = astar_path[:, 1]

# Fit a spline to the input path
tck, u = splprep([x_input, y_input], s=0.1)
u_new = np.linspace(u.min(), u.max(), (len(astar_path)-1)*num_points_between + 2)
spline_fit_curve = splev(u_new, tck)
```



Motion planning code and algorithm explanation

Pioneer 3-DX movement using “Pure Pursuit” algorithm.

- Pioneer 3-DX moves on the A-star search and spline fit curve using the “Pure pursuit” algorithm .
- “Pure pursuit” is designed to allow a vehicle to follow a desired path by continuously adjusting its steering angle based on the current state and the path geometry.
- Path Representation: Desired path represented as waypoints or smooth curve. These points are the output given by the spline fit.
- Lookahead Distance: Defines how far ahead vehicle looks along desired path, influencing steering adjustments.
- Finding the Target Point: Vehicle calculates target point based on current position and orientation.
- Steering Angle Calculation: Calculate steering angle to move towards target point.
- Control Action: Vehicle adjusts steering angle to move towards target point.
- Iterative Process: Continuously repeat process as vehicle moves along path.



Motion planning code and algorithm explanation

Pioneer 3-DX movement using pure “Pure Pursuit” code explanation.

- Following code establishes connection between CoppeliaSim and Python to execute A-star, spline fit on Pioneer 3-DX using pure pursuit.

```
#This is pure pursuit code for robot trajectory generation and P-3DX movement.
import sim
import math
import time
import sys
import matplotlib.pyplot as plt

# Connect to CoppeliaSim
sim.simxFinish(-1)
clientID = sim.simxStart('127.0.0.1', 19999, True, True, 5000, 5)

if clientID != -1:
    print('Connected Successfully.')
else:
    print('Connection Failed.')
    exit()
```

- This code obtains robot handles and motor handles from the CoppeliaSim for execution of python code.

```
# Get robot handles
error_code, left_motor_handle = sim.simxGetObjectHandle(clientID, '/PioneerP3DX/leftMotor', sim.simx_opmode_oneshot_wait)
error_code, right_motor_handle = sim.simxGetObjectHandle(clientID, '/PioneerP3DX/rightMotor', sim.simx_opmode_oneshot_wait)
# Get robot handles
error_code, robot_handle = sim.simxGetObjectHandle(clientID, '/PioneerP3DX', sim.simx_opmode_oneshot_wait)
```

Motion planning code and algorithm explanation

Pioneer 3-DX movement using pure “Pure Pursuit” code explanation.

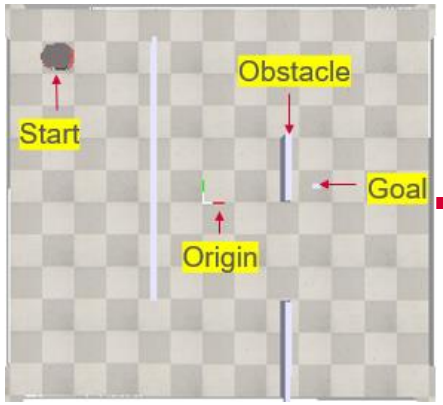
- “get_robot_pose” Function obtains real time coordinates of the robot from the CoppeliaSim.
- “pure_pursuit” function executes the “Pure pursuit” based on A-star search, spline fit, robot and motor handles and robot's current position.
- Change ”lookahead_distance”, “linear_velocity” and “goal radius” to modify path curvature, velocity and goal radius (code execution completes when robot reaches in that radius from the goal).

```
# Pure pursuit control parameters
lookahead_distance = 0.15  # Adjust as needed
linear_velocity = 1.5    # Adjust as needed
goal_radius = 0.1        # Adjust as needed

# Function to get robot pose
def get_robot_pose():
    error_code, robot_position = sim.simxGetObjectPosition(clientID, robot_handle, -1, sim.simx_opmode_streaming)
    error_code, robot_orientation = sim.simxGetObjectOrientation(clientID, robot_handle, -1, sim.simx_opmode_streaming)
    return robot_position, robot_orientation[2]  # Extract yaw angle

# Pure pursuit implementation
def pure_pursuit():
    # Add a flag to track if the final goal has been reached
    if hasattr(pure_pursuit, 'final_goal_reached') and pure_pursuit.final_goal_reached:
        return True  # Return True when the final goal is reached

    robot_position, robot_yaw = get_robot_pose()
```

- Further code is added to plot the output graphs and images. This completes the code “Code_Motion_planning_P3DX.py”

Motion planning code and algorithm explanation

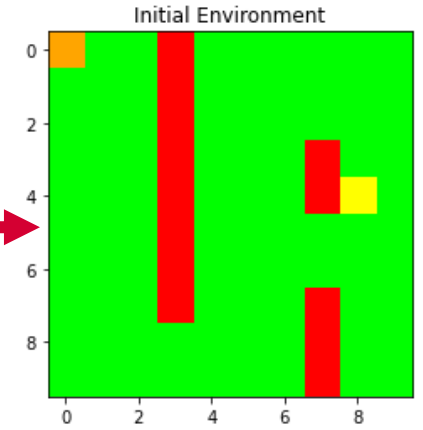
Code- “Code_Motion_planning_P3DX.py” and Motion planning through the graphics.



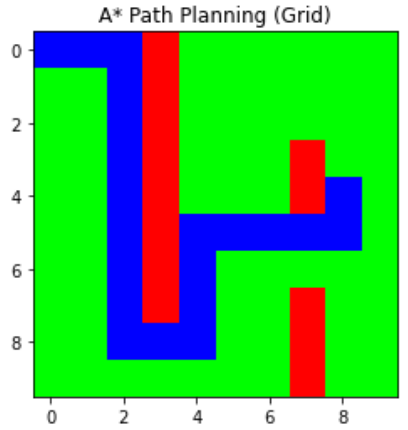
CoppeliaSim Environment

```
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
```

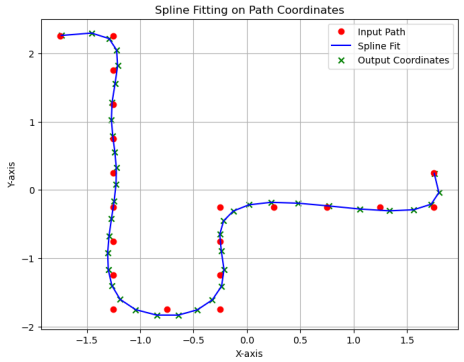
Grid Representation (Python)



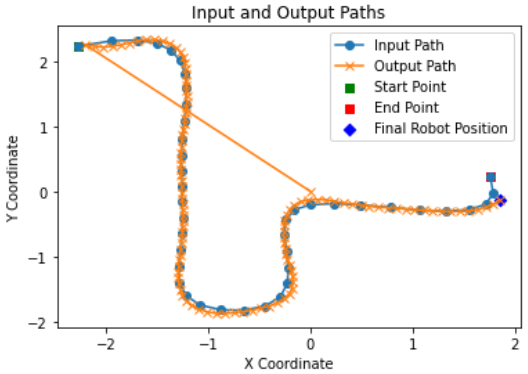
Grid in Image form (Python)



A-star path (Python)



Spline-Fit

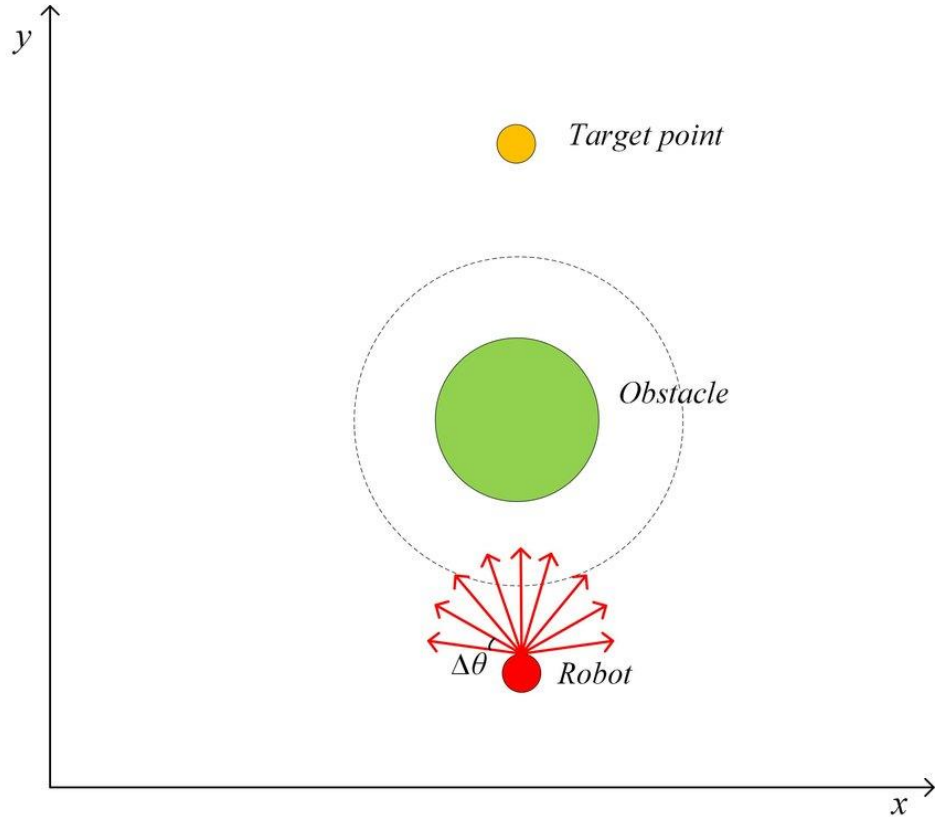


Pure pursuit execution (P 3-DX movement)

DWA Local planner development

DWA (Dynamic Window Approach)

- "Dynamic Window Approach." It's a motion planning algorithm commonly used in robotics, particularly for mobile robot navigation and obstacle avoidance. DWA is especially popular in scenarios where robots need to navigate in dynamic environments, reacting to changes in real-time.
- Velocity Sampling: DWA operates by sampling a range of velocities for the robot within a predefined range of linear and angular velocities. This range is called the “Dynamic window.“
- Collision Prediction: For each sampled velocity, DWA predicts the future position of the robot based on its current state and the selected velocity.
- Trajectory Evaluation: DWA evaluates each sampled velocity by considering two main criteria: closeness to the goal and proximity to obstacles.
- Velocity Selection: After evaluating all sampled velocities, DWA selects the velocity associated with the highest-scoring trajectory as the desired velocity for the robot.



DWA Local planner development

DWA Code explanation.

- Code “DWA_code.py” and scene “DWA_floor_scene” provided in the “DWA” folder.
- Run the Scene in CoppeliaSim and then run the code using “Spyder” or “Command Prompt”.
- Code related to Motor handles and python- CoppeliaSim is explained previously.
- Modify the DWA parameters in the __init__ function to get the optimum results. Speed_cost_gain, to_goal_cost_gain and obstacle_cost_gain are important parameters to calculate the optimum results based on requirements. The further code is developed to define these cost functions and combine them as DWA implementation.
- Modify obstacle location coordinates in self.ob

```
class Config:
    def __init__(self):
        self.max_speed = 0.5  # [m/s]
        self.min_speed = -0.5  # [m/s]
        self.max_yaw_rate = 40.0 * math.pi / 180.0  # [rad/s]
        self.max_accel = 0.2  # [m/ss]
        self.max_delta_yaw_rate = 40.0 * math.pi / 180.0  # [rad/ss]
        self.v_resolution = 0.01  # [m/s]
        self.yaw_rate_resolution = 0.1 * math.pi / 180.0  # [rad/s]
        self.dt = 0.1  # [s]
        self.predict_time = 1.0  # [s]
        self.to_goal_cost_gain = 0.15
        self.speed_cost_gain = 1.0
        self.obstacle_cost_gain = 0.8 #1.0
        self.robot_stuck_flag_cons = 0.001
        self.robot_type = RobotType.circle
        self.robot_radius = 0.35/2  # Adjust based on robot size
        self.ob = np.array([[1, 0],
                            [2, -1],
                            [2, -1.2],
                            [2.0, -1.4],
                            [2.0, -1.6],
                            [2.0, -1.8],
                            [2.2, -1.8],
                            [2.4, -1.8],
                            [2.6, -1.8],
                            #[8.0, 10.0],
                            #[9.0, 11.0],
                            #[12.0, 13.0],
                            #[12.0, 12.0],
                            #[15.0, 15.0],
                            #[13.0, 13.0]
```

DWA Local planner development

DWA Code explanation.

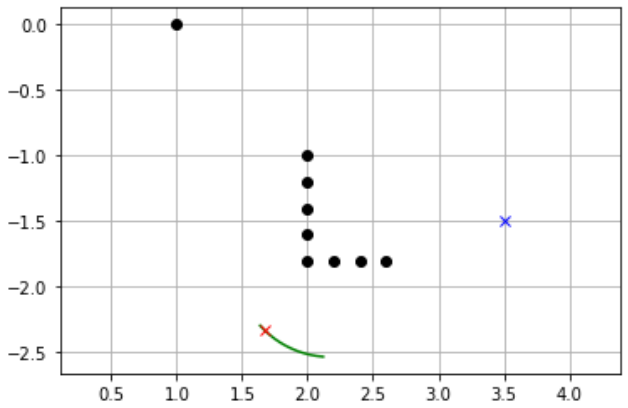
- Modify “gx” and “gy” to change the location of the goal coordinates. Modify “wheel_base” according to robot size.

```
def main(gx=3.5, gy= -1.5, robot_type=RobotType.circle):
    print("Dynamic Window Approach motion planning start!!")
    x = np.array([0.0, 0.0, math.pi / 8.0, 0.0, 0.0])  # initia
    goal = np.array([gx, gy])  # goal position [x(m), y(m)]
    config.robot_type = robot_type
    trajectory = np.array(x)
    ob = config.ob
    while True:
```

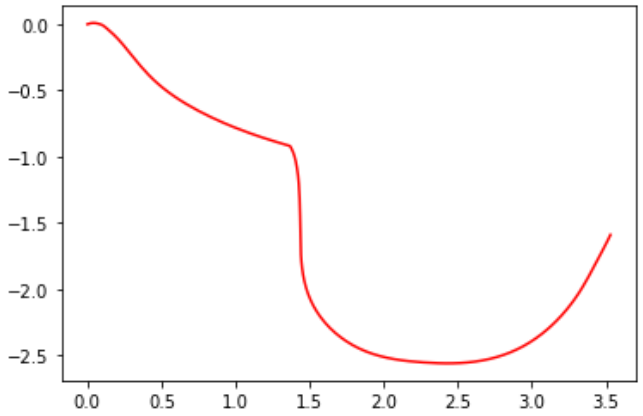
```
        # Example: Set motor velocities based on control commands
        wheel_base = 0.5  # Wheel base of the robot

        # Calculate left and right wheel velocities
        left_velocity = x[3]*2 - x[4] * wheel_base / 2
        right_velocity = x[3]*2 + x[4] * wheel_base / 2
```

- Output includes DWA generated path.
- Real time image is generated to show the DWA implementation in python window. It completes the “DWA_code.py”.

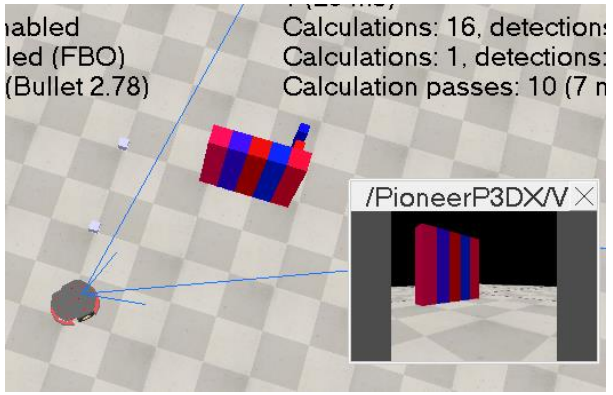
Real time image



Path traversed



CoppeliaSim Window

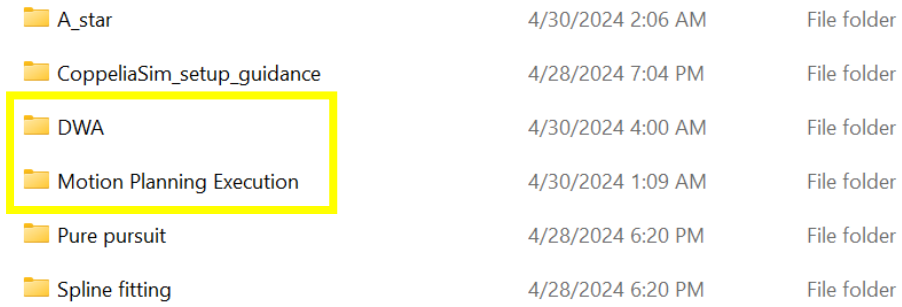


Future Scope

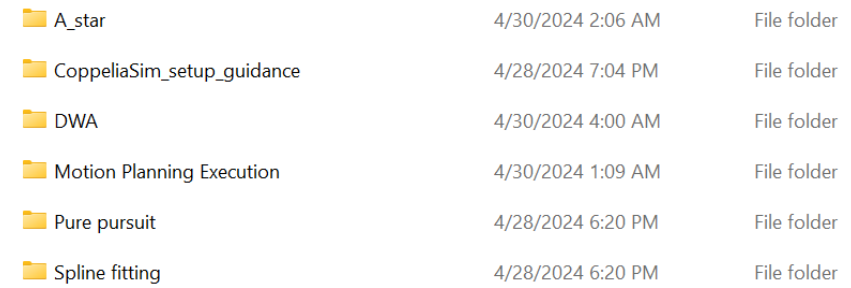
- DWA can be implemented using the computer vision.
- Integration of DWA local planner with the A-star global planner.
- Implementing the simulation with new simulator.
- Implementing the Motion planning on Hardware.

Supplementary Material

- The complete code and scene of Motion planning and DWA is provided with required API bindings in “Project work” folder and following sub folders.

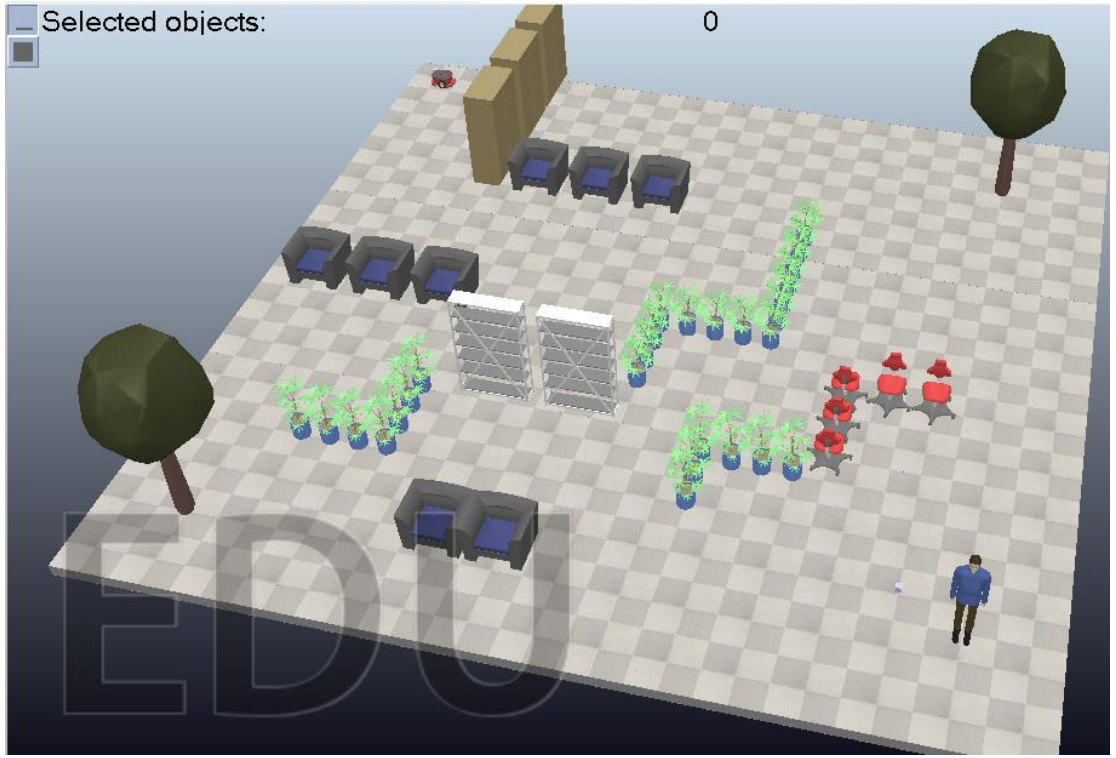


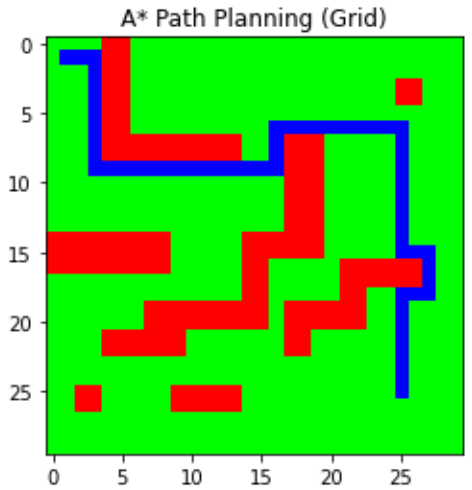
- The individual development codes for A-star, spline fit, and pure pursuit is provided in the respective folders. CoppeliaSim setup guidance is provided in respective folder and explained in the earlier slides.

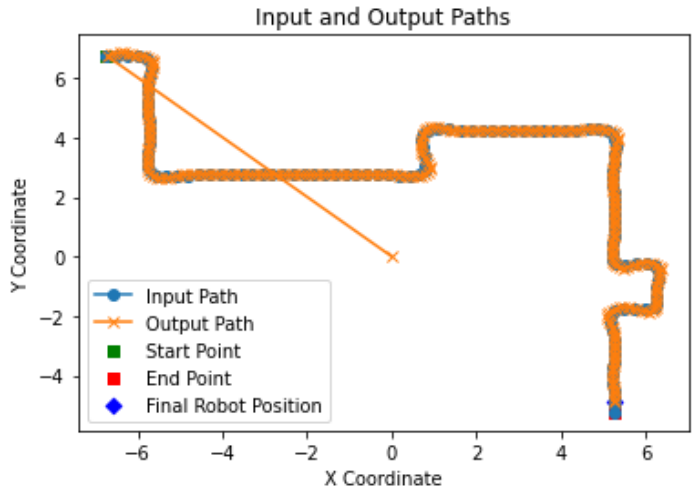


Supplementary Material

- Implementation of Motion planning on the bigger environment is provided in the “Motion Planning Execution” folder with code “Code_Motion_planning_P3DX_version_2.py” and scene “Scene_Motion_planning_P3DX_version_2”







Google Drive link for the “Project work” folder

- Download “Project work” folder from the following Google Drive link. It is accessible to anyone with the link.

https://drive.google.com/drive/folders/1l3ni4km1TrvQbEWUJoVjoCS8c8aAXtpc?usp=sharing

- YouTube Link for the working model of Motion planning on P-3DX.

P 3DX version 2 Video- https://youtu.be/2yMcM8VPj3c

P 3DX version 1 Video- https://youtu.be/OjDhxGP9NaU

DWA CoppeliaSim Python demo (Dynamic window Approach)- https://youtu.be/P98ZjHzeOtQ


UIC