AUTONOMOUS ROBOT

Assumptions and Limitations

- Assumptions:
  - The warehouse environment is a rectangular grid.
  - Obstacles are detected within 0.2m using sensors.
  - The robot moves with a constant linear velocity (1 m/s) and can rotate with an angular velocity of 3 rad/s.
  - The robot starts from a predefined point and tries to navigate towards a target goal while avoiding obstacles.
- Limitations:
  - No consideration for dynamic obstacles (obstacles are static).
  - Sensors have a limited detection radius (0.2m).
  - We assume a simple 2D plane for the simulation.

Environment and Robot Setup

- Robot Wheel Specs:
  - Diameter = 10 cm (0.1 m)
  - Wheel Distance = 30 cm (0.3 m)
- Obstacle Setup:
  - There are 10 obstacles randomly placed on the floor, with at least 1m separation between them.
- Robot Sensors:
  - Simulate LIDAR-like behavior using a simple proximity sensor model that triggers when an obstacle is within 0.2m.

Algorithm Design: Obstacle Avoidance Logic

A good choice is the Bug 2 Algorithm:

- Step 1: The robot moves towards the goal directly unless it encounters an obstacle.
- Step 2: If an obstacle is detected within 0.2m, the robot rotates left/right and follows the obstacle boundary.
- Step 3: Once clear, the robot continues towards the goal.

Code Implementation:

Code: Autonomous Robot Navigation using pygame

AUTONOMOUS ROBOT

First, install the required dependencies

pip install pygame

Then we will create the code:

```
import pygame

import math

import random


pygame.init()

WIDTH, HEIGHT = 800, 600

screen = pygame.display.set_mode((WIDTH, HEIGHT))

pygame.display.set_caption("Robot Navigation")


ROBOT_RADIUS = 15

OBSTACLE_RADIUS = 20

SENSOR_RANGE = 40

LINEAR_VELOCITY = 1

ANGULAR_VELOCITY = 3


WHITE, BLACK, RED, BLUE = (255, 255, 255), (0, 0, 0), (255, 0, 0), (0, 0, 255)


robot_pos, robot_angle = [100, 100], 0

goal_pos = [700, 500]


obstacles = [(random.randint(50, WIDTH - 50), random.randint(50, HEIGHT - 50)) for _ in range(10)]


def distance(p1, p2):

  return math.hypot(p2[0] - p1[0], p2[1] - p1[1])


def detect_obstacle(robot_pos):

  return next(((obs, obs) for obs in obstacles if distance(robot_pos, obs) <= SENSOR_RANGE + 
OBSTACLE_RADIUS), (None, None))
```

AUTONOMOUS ROBOT

```
def rotate(angle, direction):
    return (angle + (ANGULAR_VELOCITY if direction == "left" else -ANGULAR_VELOCITY)) % 360


def move_towards_goal()
    global robot_angle
    angle_to_goal = math.degrees(math.atan2(goal_pos[1] - robot_pos[1], goal_pos[0] - robot_pos[0]))
    if abs(robot_angle - angle_to_goal) > 5:
        robot_angle = rotate(robot_angle, "left" if angle_to_goal > robot_angle else "right")
    else:
        robot_pos[0] += LINEAR_VELOCITY * math.cos(math.radians(robot_angle))
        robot_pos[1] += LINEAR_VELOCITY * math.sin(math.radians(robot_angle))


def avoid_obstacle(obstacle_pos):
    global robot_angle
    robot_pos[0] -= 10 * math.cos(math.radians(robot_angle))  # Back off slightly


    angle_to_obstacle = math.degrees(math.atan2(obstacle_pos[1] - robot_pos[1], obstacle_pos[0] - robot_pos[0]))
    angle_diff = (angle_to_obstacle - robot_angle) % 360
    if 0 < angle_diff > 180:  # Obstacle is in front
        robot_angle = rotate(robot_angle, "right")  # Turn right
    else:
        robot_angle = rotate(robot_angle, "left")  # Turn left


    robot_pos[0] += LINEAR_VELOCITY * math.cos(math.radians(robot_angle))
    robot_pos[1] += LINEAR_VELOCITY * math.sin(math.radians(robot_angle))


running = True
while running:
    screen.fill(WHITE)
```

AUTONOMOUS ROBOT

```
    pygame.draw.circle(screen, BLUE, goal_pos, 10)
    for obs in obstacles:
        pygame.draw.circle(screen, RED, obs, OBSTACLE_RADIUS)
    pygame.draw.circle(screen, BLACK, (int(robot_pos[0]), int(robot_pos[1])), ROBOT_RADIUS)

    obstacle_detected, obstacle_pos = detect_obstacle(robot_pos)
    if obstacle_detected:
        avoid_obstacle(obstacle_pos)
    else:
        move_towards_goal()

    if distance(robot_pos, goal_pos) < 10:
        print("Goal reached!")
        running = False

    pygame.display.flip()
    pygame.time.delay(50)

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

pygame.quit()
```