Left Wall Follower

A Solution To Escape the Maze

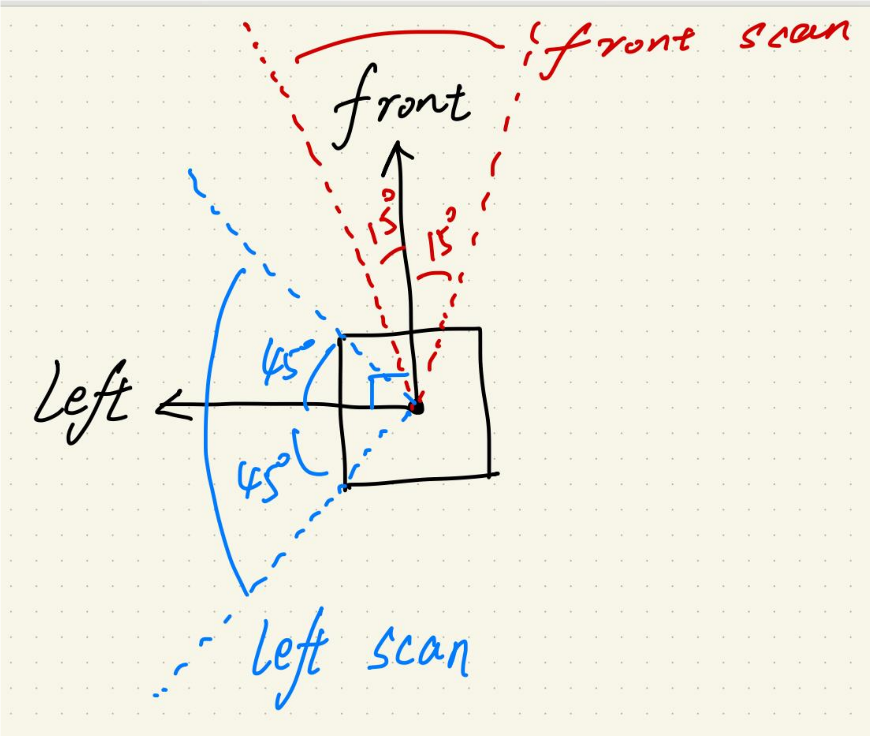
Jeff Pang Liu

Solution Introduction

- The solution is a suboptimal solution, though it can escape the maze in our assignment **in almost 100%**, but it might not solve all maze problems.

Algorithm Introduction - Sensor Base

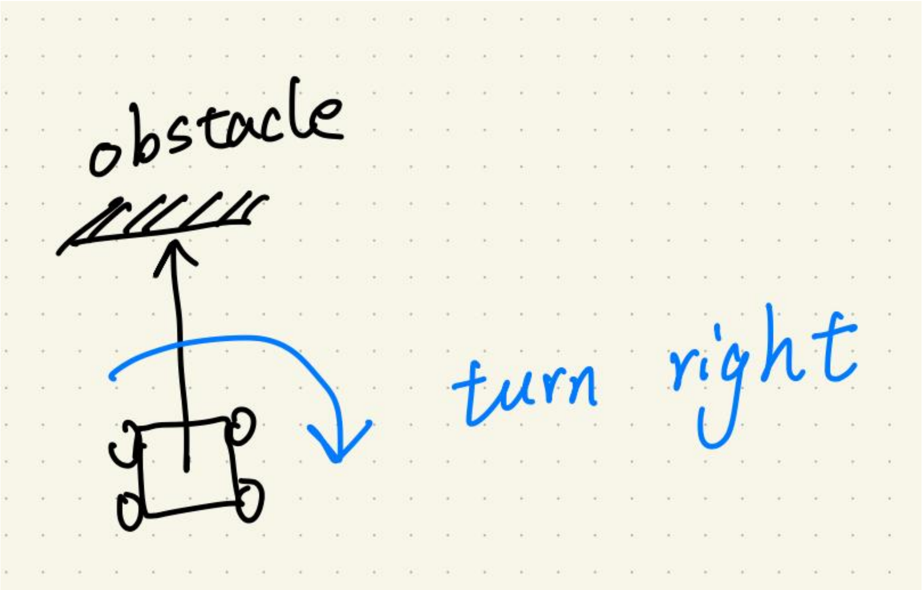
- The algorithm is based on two data points:
- 1. The front side scan between -15 to 15 degrees, to find if there is an obstacle in front
- 2. The left side scan between 45 to 135 degrees, to find the closest distance to the left side, and the degree of that closest distance.

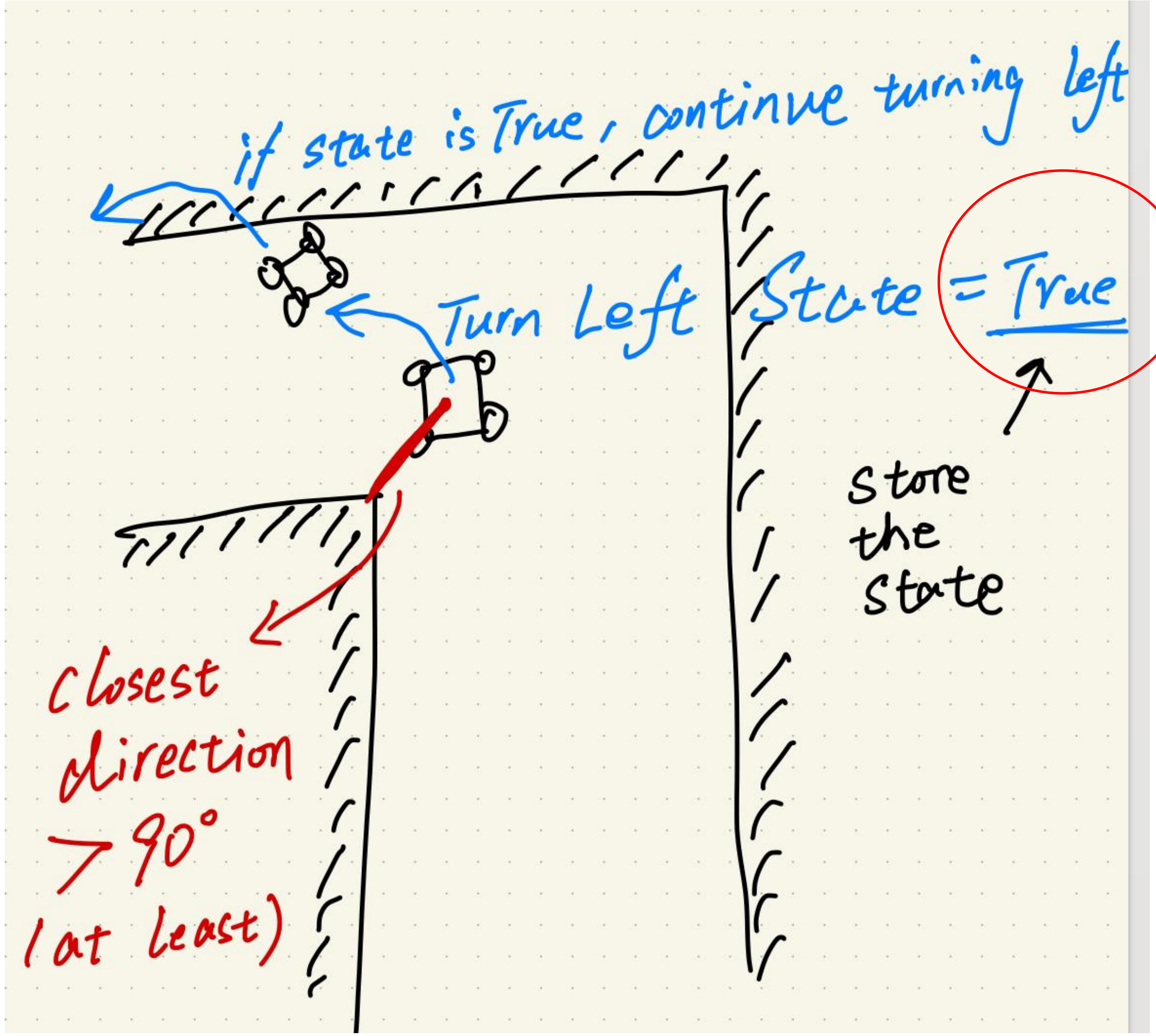


Algorithm Introduction - Logic

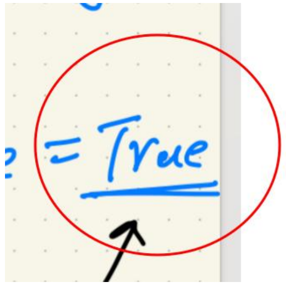
- If there is an obstacle in front:
  - If the state of left turning is True: Turn Left
  - Else: Turn Right
- Else:
  - If the robot is in the dead zone, turn right
  - If the robot is between the dead line and the keep line, move toward the keep line
  - If the robot is between the keep line and the bound line, move toward the keep line
  - If the robot is outside the bound line, move straight to find the wall in the left side

Algorithm Introduction - Obstacle

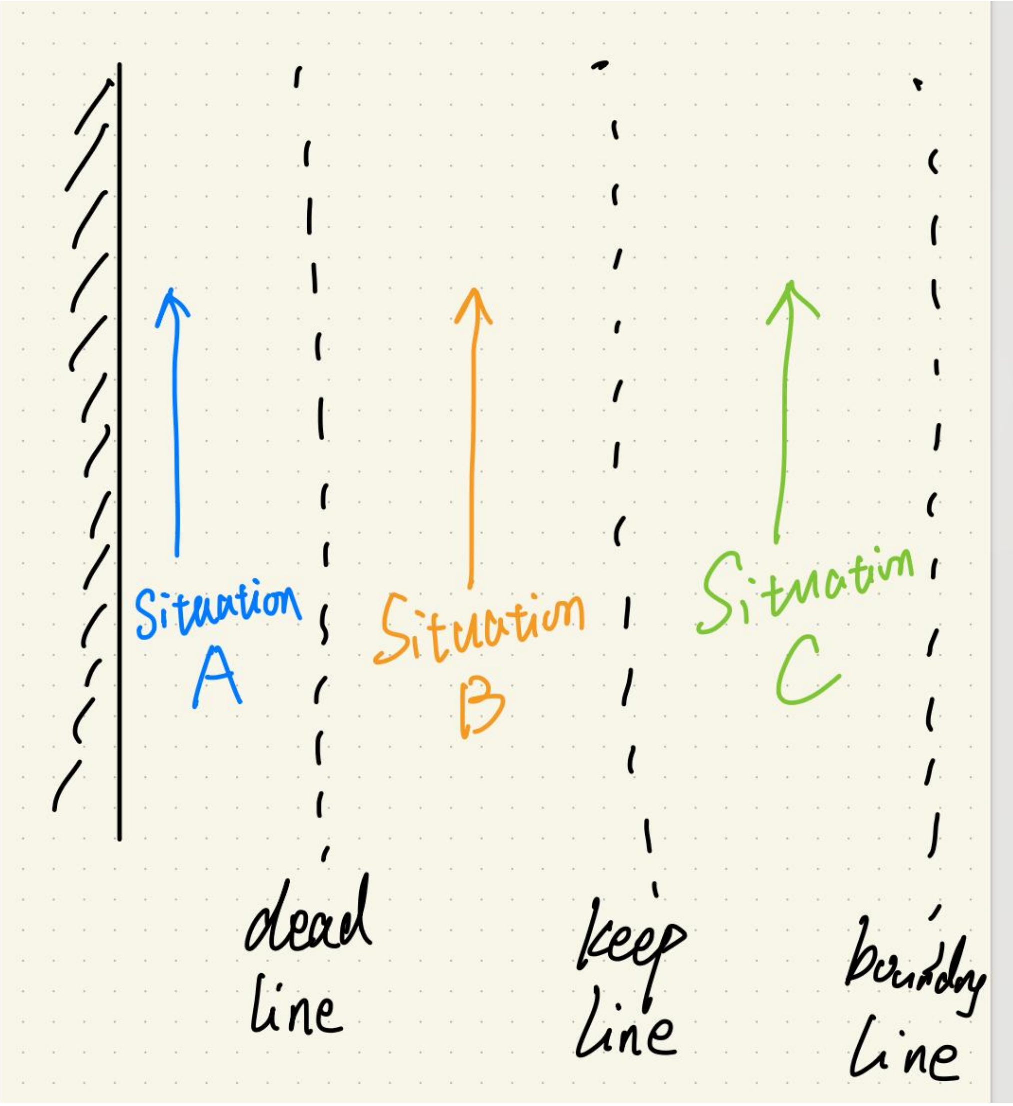


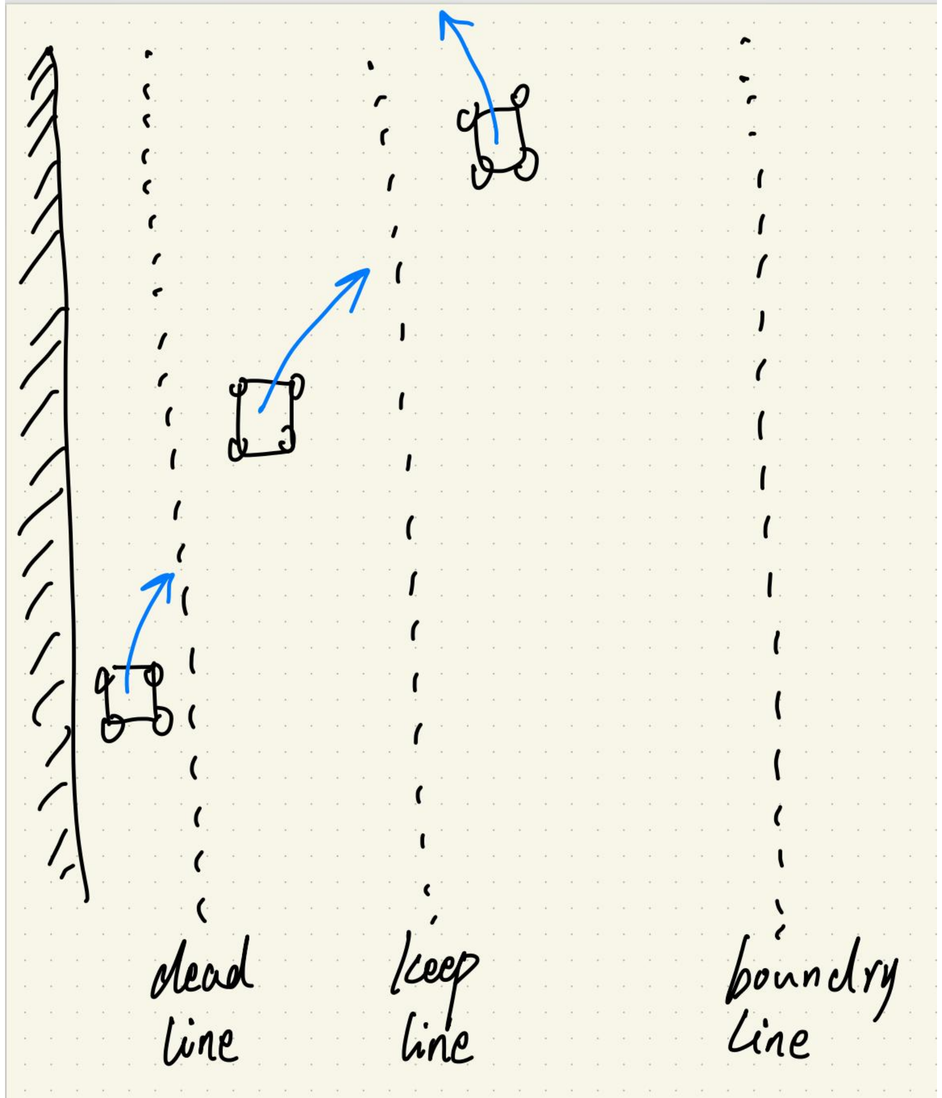


- Note: This is a state storage variable



Algorithm Introduction - No Obstacle





Codes Introduction

class LeftWallFollower:

- def __init__(self)
- def clst_dtc_and_dir(self, start_degree, end_degree)
- def scan_cb(self, msg)
- def follow_left_wall(self)
    - 1. set velocity, three area divided
    - 2. main algorithm (Bang Bang Control)

```python
#!/usr/bin/env python3
import math
import rospy
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist

# BANG BANG CONTROL
class LeftWallFollower:
    def __init__(self):
        """
        Initialize the publisher and subscriber
        Initialize the data points p, and the state of turning left
        """
        self.cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
        self.scan_sub = rospy.Subscriber('/scan', LaserScan, self.scan_cb)

        # Store all lidar data, use scan_cb() to get and refresh data points
        self.p = [9.9] * 360

        # State of Left Turning: This value is True If and Only If in Turning
        self.turn_left_state = False
```

```python
def clst_dtc_and_dir(self, start_degree, end_degree):
    """
    Find the closest distance and direction
    """
    min_dtc = self.p[start_degree]
    min_dir = start_degree
    for i in range(start_degree, end_degree):
        if min_dtc > self.p[i]:
            min_dtc = self.p[i]
            min_dir = i
    return min_dtc, min_dir

def scan_cb(self, msg):
    """
    Scan and get the Lidar data, and store in the list p
    """
    degree = 0
    for i in range(0,360):
        if msg.ranges[degree] == float('inf') or msg.ranges[degree] == 0.0:
            self.p[i] = 9.9  # 9.9 means infinite
        else:
            self.p[i] = msg.ranges[degree]
        degree += 1
```

```python
def follow_left_wall(self):
    """
    The Algorithm of Following the left_side: (Bang Bang Control)
    1. If there is no obstacle in front, follow the left side wall, and keep the
        1-1. If the distance is less than dead distance, move toward keep line (m
        1-2. If the distance is between the dead line and the bound line, move to
        1-3. If the distance is larger than bound line, move straight to find a w
    2. If there is an obstacle in front:
        2-1. If the left turn state is not true, turn right.
        2-2. If the left turn state is true, turn left.
    """
    twist = Twist()

    left_clst_dtc, left_clst_dir = self.clst_dtc_and_dir(45,135)

    # common use speed
    lvs = 0.2
    avs = 0.2
    av = avs * 3

    # area divided
    dead = 0.2
    keep = 0.3
    bound = 0.5

    # obstacle detect, the range is -15 to 15 degrees
    obstacle_left_detect_dtc, obstacle_left_detect_dir = self.clst_dtc_and_dir(0,
    obstacle_right_detect_dtc, obstacle_right_detect_dir = self.clst_dtc_and_dir(
    obstacle = True if obstacle_left_detect_dtc < keep or obstacle_right_detect_c
```

```python
# BANG BANG CONTROL
if obstacle:
    print("Obstacle In Front")
    if self.turn_left_state == True:
        twist.linear.x = 0
        twist.angular.z = av
        self.turn_left_state = True ;
    else:
        twist.linear.x = 0
        twist.angular.z = -av
        self.turn_left_state = False
```

```
else:
    print("No Obstacle In Front")
    if left_clst_dtc < dead:
        twist.linear.x = lvs
        twist.angular.z = -avs
        self.turn_left_state = False ; print("Trying to move along the Keep l
    elif left_clst_dtc < keep:
        if left_clst_dir < 70: # 70 is a special number, means the direction
            twist.linear.x = lvs
            twist.angular.z = -avs
            self.turn_left_state = False ; print("Trying to move along the Ke
        elif left_clst_dir > 90: # 90 is the degree of normal left, larger th
            twist.linear.x = lvs
            twist.angular.z = av * 2
            self.turn_left_state = True ; print("Left Side Disapear, Turn Lef
        else:
            twist.linear.x = lvs
            twist.angular.z = 0
            self.turn_left_state = False ; print("Trying to move along the Ke
    elif left_clst_dtc < bound:
        if left_clst_dir < 90: # 90 is the degree of normal left, smaller tha
            twist.linear.x = lvs
            twist.angular.z = av
            self.turn_left_state = False ; print("Trying to move along the Ke
        else:
            twist.linear.x = lvs
            twist.angular.z = av * 2
            self.turn_left_state = True ; print("Left Side Disapear, Turn Lef
    else:
        twist.linear.x = lvs
        twist.angular.z = 0
        self.turn_left_state = False ; print("No obstacle, No left side wall,
```

```python

            # PUBLISH THE TWIST
            self.cmd_vel_pub.publish(twist)

if __name__ == '__main__':
    rospy.init_node('LeftWallFollower')

    LeftWallFollower = LeftWallFollower()

    rate = rospy.Rate(10)

    while not rospy.is_shutdown():
        LeftWallFollower.follow_left_wall()
        rate.sleep()
```

Test Results

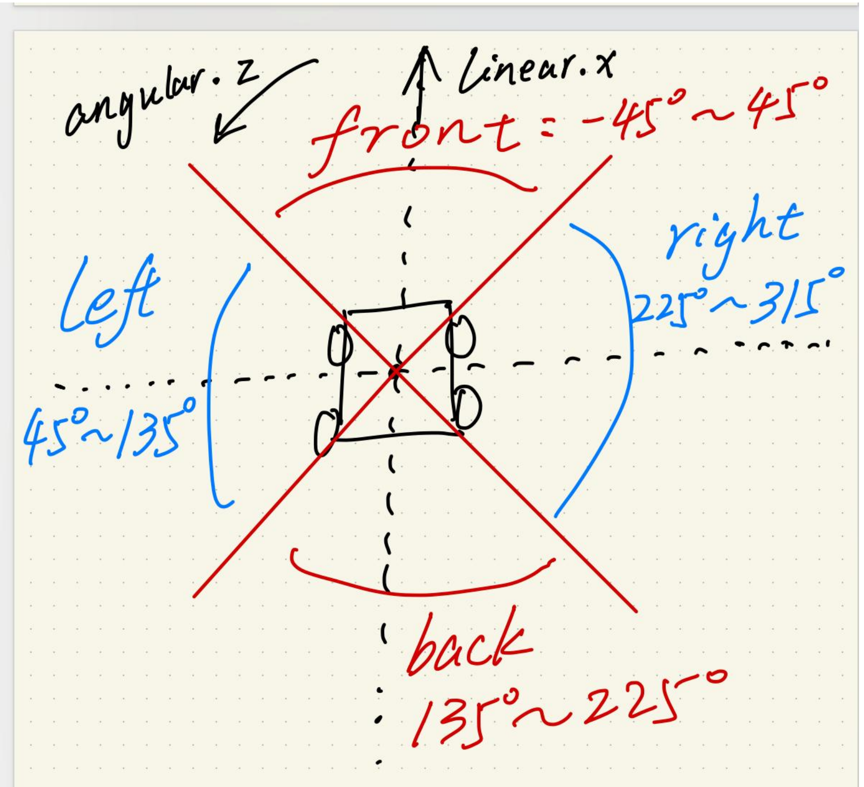
- Test: 10 Times, in different positions
- **Successfully Escape Rate: 100%**
  - In Perfect Route: 60%
  - In Not Perfect Route: 40% (But still escape the maze!)
- The result represents this algorithm is an suboptimal solution, it can tolerant fault, with robustness.

Fault Tolerance

- There are two situations the robot will not follow the perfect route, but not influence the result:
  - 1. In dead end with three walls, turn left with state True, but it will not influence the escape of dead end because that state will eliminate when it toward the wall again.
  - 2. In the small room at the exit, sometimes the robot detect the wall as an obstacle, then escape the maze directly, without entering the small room. This is not crucial in this maze, and I don’t fix it because the sensor accuracy is not good, and the data less than 0.1 meters is not very accurate, so I can't make the robot walk close to the wall. To solve this fault, I will explain the complete solution in the next slide.

(Abandoned Solution)

- At fisrt, I try to use more data points.



(Abandoned Solution)

- Right picture is the version that I try to use four directions…
- Also, use too much data points can cause logic very chaos, I have a 300 lines code with four directions detection version, but finally I didn't use it because it is really hard to modify the codes. So after several times' effort, I finally achieve the goal of only use two directions' detection!
- The behavior is even worse than the final version of two directions' detection.
- This makes me realize the importance of simplicity.

```
# STEP 2 : BANG BANG CONTROL to move along the left side wall
else:
    # SITUATION A: The left side is in the dead line
    if self.left_closest_distance < self.dead_distance:
        print("=====SITUATION A: LEFT IN DEAD=====")
        if self.front_normal_distance > self.boundry_distance:
            print("A-1: IF NO OBSTACLE ON THE FRONT, THEN MOVE FOWARD")
            if self.left_closest_dir < 105:
                print("A-1-1: THE DIRECTION IS GOOD, KEEP GOING\n")
                twist.linear.x = lvs
                twist.angular.z = 0
            else:
                print("A-1-2: THE DIRECTION IS NOT GOOD, TURN A LITTLE BIT RIGHT AND KEEP GOING\n")
                twist.linear.x = lvs
                twist.angular.z = -avs

        elif self.left_closest_dir < 100:
            # IF OCCURS FRONT OBSTACLE, AND LEFT DIR < 100;
            # '<100' means the robot is not toward the keep line,
            # and use 100 because the sensor is not really precise especially when the robot is very close to the wall
            print("A-2: WHEN THE ROBOT IS NOT TOWARD THE KEEP LINE")
            if self.right_closest_distance < self.dead_distance and self.back_closest_distance < self.dead_distance and
                # this if statement means the robot is trapped in a very narrow space
                # the condition is: right, back, left sides are walls, only front is not wall
                # again, use rospy.sleep for 1 second is because in narrow space, the sensors are not precise
                # to avoid sensors up and down frequently, just let the robot toward the front and go for 1 second
                print("A-2-1: THE ROBOT IS TRAPPED IN A VERY NARROW SPACE\n")
                twist.linear.x = lvs
                twist.angular.z = 0
                self.cmd_vel_pub.publish(twist)
                rospy.sleep(1)
            else:
                print("A-2-2: THE ROBOT IS NOT IN THE TRAP, SO IT SHOULD MOVE NEAR TO THE KEEP LINE")
                if self.front_normal_distance > self.keep_distance and self.right_normal_distance > self.keep_distance
                    print("A-2-2-1: Straighten the front of the car")
                    twist.linear.x = lvs
                    twist.angular.z = - avs / 2 # In this special case, needs very slow rotate
                elif self.front_closest_distance < self.dead_distance and self.right_closest_distance < self.dead_dista
                    print("A-2-2-2: The front of the car straightened, but hit the wall and stuck at the wall")
                    # move back and turn right, to solve the stuck situation
                    twist.linear.x = - lvs
                    twist.angular.z = - avs
                elif self.front_closest_dir > 35:
                    print("A-2-2-3: The front of the car is straightened, and not stuck")
                    # continue to move, and go toward the keep line (turn a little bit to right)
                    twist.linear.x = lvs
                    twist.angular.z = -avs
                else:
                    print("A-2-2-4: Get out of trouble by move back and turn right")
                    twist.linear.x = - lvs
                    twist.angular.z = - av

        elif self.left_normal_distance > self.dead_distance:
            print("A-2-3: Suddenly nothing on the left\n\n")
            # Suddenly nothing on the left, means need a left turn or a V turn
            twist.angular.z = av
            twist.linear.x = lvs
        else:
            print("A-2-4: In all other cases, move left to correct the error.\n\n")
            # Errors includes all other cases, which are not what we concerned as "wall on the left and move toward rig
            twist.linear.x = lvs
            twist.angular.z = avs

    # SITUATION B: The left closest distance is between dead line and keep line
    elif self.left_closest_distance < self.keep_distance:
        print("=====SITUATION B: LEFT between DEAD and KEEP=====")
        if self.left_closest_dir < 95:
            print("B-1: Turn a little to right to move toward keep line.\n\n")
            twist.linear.x = lvs
            twist.angular.z = -avs
        elif self.left_normal_distance > self.keep_distance:
            print("B-2: Nothing on the left, turn left to find the left wall.\n\n")
            twist.linear.x = lvs
            twist.angular.z = av * 2 # This is to help move quickly to the right path
        else:
            print("B-3: Already toward the keep line, keep moving.\n\n")
            twist.linear.x = lvs
            twist.angular.z = 0

    # SITUATION C: Closest Left Distance is between KEEP LINE and BOUNDRY LINE
    elif self.left_closest_distance < self.boundry_distance:
        print("=====SITUATION C: LEFT BETWEEN KEEP AND BOUNDRY=====")
        if self.front_closest_distance < self.keep_distance:
            print("C-1: The front wall occurs")
            # When the front wall occurs,
            # The robot can turn left or right based on different conditions
            if self.right_normal_distance < self.boundry_distance:
                print("C-1-1: Right wall exists, so not turn right")
                if self.front_closest_distance < self.dead_distance:
                    print("C-1-1-1: Very close to the front wall, move back and turn left")
                    twist.linear.x = -lvs
                    twist.angular.z = avs
                elif self.right_closest_distance < self.keep_distance:
                    print("C-1-1-2: Very close to the right wall, turn left to avoid collision")
                    twist.linear.x = lvs
                    twist.angular.z = av
                else:
                    print("C-1-1-3: Move forward")
                    twist.linear.x = lvs
```

Thanks!