**Name**: Akansha Bhatia - **Student** **ID:** M00975443 – **Subject Code**: PDE4430

Public Git Hub Repository: <https://github.com/Akansha-Robotics/MR-CW1.git>

Contents

[Teleoperation ( tele\_op.py ) 2](#_Toc151737923)

[Description 2](#_Toc151737924)

[Functions 2](#_Toc151737925)

[key\_input & moveBindings 2](#_Toc151737926)

[keys\_msg 2](#_Toc151737927)

[Autonomous Navigation ( auto\_nav.py ) 2](#_Toc151737928)

[Description 2](#_Toc151737929)

[Functions 3](#_Toc151737930)

[user\_input 3](#_Toc151737931)

[to\_goal 3](#_Toc151737932)

[Avoiding Wall Collusion ( crash\_avoid.py ) 3](#_Toc151737933)

[Description 3](#_Toc151737934)

[Functions 4](#_Toc151737935)

[crash\_checker 4](#_Toc151737936)

[robot\_move 4](#_Toc151737937)

[Vacuum Cleaning ( vacuum.py ) 4](#_Toc151737938)

[Description 4](#_Toc151737939)

[Functions 4](#_Toc151737940)

[Pattern 4](#_Toc151737941)

[move\_line 5](#_Toc151737942)

[Vacuum Cleaning With Multiple Vacuums ( vacuum\_multi.py ) 5](#_Toc151737943)

[Description 5](#_Toc151737944)

[Function 6](#_Toc151737945)

[collusion\_priority & collusion\_movement 6](#_Toc151737946)

[pattern\_loop & move\_line 6](#_Toc151737947)

[spawn\_turtle function & start\_pattern\_loop 6](#_Toc151737948)

# Teleoperation ( tele\_op.py )

## Description

The first script involves users being able to move the robot using the keyboard along with being able to adjust the speed. The logical flow of program would include detecting specific keyboard presses related to movement of robot (key\_input function & moveBindings) and move the robot according to keys detected which is done in the main program (if \_\_name\_\_=="\_\_main\_\_": ). For controlling speed, the program would need to be able to detect the key for it, which is +, along with allowing the user to enter a new speed and then adjusting speed according to the new values. Furthermore, the keyboard press commands are communicated to the user which is the first message being printed on the terminal (keys\_msg). Finally, the cleanup actions need to be taken so that whether the terminal is closed properly or not, the robot will be stopped, and the terminal will be returned to normal. This cleanup action is essential as if you are controlling a 100 KGs or so humanoid robot, accident movement controls sent by improper exiting of the terminal can damage the robot and environment.

## Functions

key\_input & moveBindings functions involve reading the keyboard presses inputted by user (key\_input) and move according to actions set out (moveBindings). In moveBindings, the x and y coordinates were set out within the tuple which correspond to the linear x for forward and backward movement along with angular z for left and right movement.

keys\_msgis a variable that prints the control options the user has for moving the robot when the node is started. This action is essential for informing users how the robot can move along with indication that their terminal would be following these commands.

One important aspect to keep in mind is that the program is changing the terminal setting to raw mode allowing the commands to be taken over and restricted to ones set out in the program. Therefore, the program requires the cleanup action to stop the robot along with ensure the terminal setting is changed back to the original and normal setting.

A diagram of a computer

Description automatically generated with medium confidence

# Autonomous Navigation ( auto\_nav.py )

## Description

The second script involves allowing the user to enter the location to move the turtle in which the program will control the turtle to do so. The program would be collecting the x and y coordinate of the goal point ( user\_input function ) and then calculating the distance between goal and current point ( to\_goal function). Based on this calculation, the value of distance to travel would be published to controls of the robot within the main program. In addition, the logic of the program would need to ensure that the second values can only be entered once the first goal point has been reached. Since calculation of distance to travel is dependent on the current point of the robot, ensuring the flow of the robot first reaching the goal point and then allowing second point is essential to making the program efficient. Another aspect to consider for logic of the program would be to ensure that the first input does not need to wait for the robot to reach the goal point. Since turtlesim node has just started, the turtle is not moving along with the location being known resulting in first input required to override the wait condition that has been applied for the second and onwards goal points. The final aspect of logic for the program would be the use of threading to ensure the flow of operations going smoothly in which first input is allowed without waiting along with other input waiting for turtle to reach point and then allowing next user input.

## Functions

user\_input is used to simply collect the goal point x and y input given by the user. The value accepted is a float numerical value and stored in goal\_point.x and goal\_point.y. The function also states that to give prompt to the user once the goal point has been reached and clear along with ensuring that this condition is not applied to first input.

to\_goal function is used to calculate the distance from current point to goal point along with publishing the controls to cmd\_vel to move the turtle. In addition, calculation is done through finding the Euclidean Distance which uses the Pythagorean Theorem which uses to find the length of the straight line within 2 dimensional. Furthermore, the angle in radians is found by finding the difference between goal and current along with getting square root.

Finally, the if loop in function moves the robot based on distance between points along with indicating when the turtle has reached the point. This loop also ensures that once the turtle has reached, it prints the message on the terminal along with stopping the turtle by entering 0 values in the goal point. One important aspect is that if the distance is not calculated properly or stop instruction is not published at the end, the program will face an error in which the turtle will spiral all over the window as shown in the picture beside. Therefore, cleanup action of stopping and testing is essential to make the program efficient.A screenshot of a video game

Description automatically generated

A diagram of a computer network

Description automatically generated

# Avoiding Wall Collusion ( crash\_avoid.py )

## Description

The third script involves creating a program that would ensure that wall collusion is avoided by placing controls that override the turtle when it is near one. The logical flow of the program would include detecting when the wall is near ( crash\_checker function) and moving the robot according to the situation ( robot\_move function).

In addition, we would need to keep a good enough distance for the robot to turn as the turning move would not be an immediate right angle but rather a slow curve movement. If the robot is too close to the wall, it could end up hitting the wall while turning which would defeat the purpose of this program. Finally, the user should be informed that the robot is near danger and be able to understand why the robot is turning.

## Functions

crash\_checkeris a simple function that would calculate whether the robot is near the wall by comparing the current position ( robot\_pose ) with the limits and safe distance along with saving and outputting the True or False value in near\_crash. In addition, the function also has a for loop that would display the message on the terminal if the robot is near the wall.

robot\_moveis the main function for controlling the movement of a robot based on whether the robot is near the wall or not. The if loop (if crash\_checker() loop ) states that if the crash detected is True, it will have 1 linear and 1 angular movement allowing the robot to turn and move forward. Otherwise, it will continue to move forward with linear equal 2.

A diagram of a diagram

Description automatically generated

# Vacuum Cleaning ( vacuum.py )

## Description

The fourth script involves making the turtle move in a vacuum, ensuring the whole window is covered in a manner that is efficient considering the time and path. The logic of the program is that the vacuum will move to the corner of the screen and start moving in a pattern that goes to all the rows and columns (pattern function). In addition, the function would be calculating where the next target should go based on the line value. For example, we understand that line 1 starts at the bottom left side of the window at 0, so the target point would be 11 which is the other end of the line. However, when the vacuum moves to line 2, it is already at 11 and would need to move to 0 point. Therefore, odd line numbers should have a target of 11 while even line numbers would have a target of 0. This program logic works for the requirement as the requirement given is only about covering the window in which the limits and area are known. If the requirement was changed toward implementing within a home with obstacles, the logic would change towards creating a path based on points set out across the floor.

Finally, the program also ensures that the movement is linear (move\_line function) so that the movement is well controlled, and the line is being completely cleaned without any spot missing. Messages are printed on terminal to indicate when a row or column has been completed along with when a set of row and column has been completed resulting in the whole window cleaned. An offset has also been added resulting in the position to be slightly offsetted in every loop. This offset ensures that the vacuum keeps looping the row and column ensuring that any spot missed would be covered.

## Functions

Pattern function loops the path for rows and columns, so it moves line after line going from start to end. This path is created by changing the target point from 0 to 11 depending on whether the line number is even or odd which is possible to do since the start point and location is known and set by program. In addition, a message gets printed on terminal once a row or column is completed to indicate the user on the progress.

move\_line function does the calculation of distance between the goal and current point. In addition, it also differentiates between the x and y axis to control whether it is moving forward or backwards or whether it is turning left or right. A screenshot of a computer game

Description automatically generated

This calculation is then stored in vel\_msg to be published to cmd\_vel allowing the vacuum to be controlled. The function ends with a cleanup action of placing 0 in both x and y to stop vacuum. The flow of the program is essential as calculating target point based on the line number and ensuring only linear movement occurs along with clean up action ensure the vacuum does not end up spiralling and losing control as seen on the picture.

Finally, the main program uses both functions to keep the vacuum running in a loop to cover the window along with adding an offset. This increase in offset is stored in offset\_increase in which it is added to row\_offset and column\_offset every time a set of row and column is completed until getting restarted when values is greater or equal to 10.

A diagram of a diagram

Description automatically generated

# Vacuum Cleaning With Multiple Vacuums ( vacuum\_multi.py )

## Description

The last script would be moving multiple turtles with vacuum behaviour to clear the whole window in an efficient manner in terms of time and path. This program would be using similar functions in vacuum.py with controlling the movement to be linear and moving robot to end of the row based the start and limit points ( pattern\_loop & move\_line functions ). However, the main difference in this program would be controlling multiple vacuum together along with ensuring they do not run into each other. Therefore, the logic of the program involves splitting the vacuums into 4 different quadrants and looping their path in their area allowing for the whole window to get cleaned faster.

Even though we are splitting the area, the program still ensures collusion between vacuum does not occur. If one vacuum is near to another one, one of the vacuums stops to allow the other one to move. Once the path is cleared of no vacuums, the other vacuum is allowed to move (collusion\_movement function ). In addition, the vacuum that stops is decided by a priority function (collusion\_priority) in which 1 has the highest priority while 4 has the lowest,

The program would also be spawning and numbering the turtles from 1 to 4 along with printing messages for successful service calls and location of vacuum ( spawn\_turtle function ). In addition, the start\_pattern\_loop function would loop this thread of events along with activating the pattern\_loop function.

Finally, the main program sets the quadrants to each turtle in which their start and limits along with assigns them to their quadrants. In addition, the publishers and subscribers initialize in looped through the numbering of turtles so that all 4 get started.

## Function

collusion\_priority & collusion\_movement function is used to ensure that the vacuum does not hit each other. collusion\_movement ensures that one turtle stops for another turtle to give way and then continues to move when the path is cleared. While, collusion\_priority gives indication on which turtle should stop based on the number from turtle.

pattern\_loop & move\_line functions are similar to the vacuum.py program in which it loops the row and columns to move around the window along with ensuring it moves in a linear form from start to end. The main difference here is that the limits and start points entered are variables as those values will be based on the quadrant assigned.

spawn\_turtle function & start\_pattern\_loop function is used to spawn multiple turtles and setting their location with spawn\_turtle along with looping the operations to connect the turtles with pattern\_loop that will enable them to clean their rows and columns.

A diagram of a diagram

Description automatically generated