# e-Yantra Robotics Competition Plus

# eYRCPlus-PS1#2678

| | |
|---|---|
| **Team leader name** | **Avick Dutta** |
| **College** | **Acharyya Prafulla Chandra Ray Polytechnic** |
| **Email** | **avickdutta@gmail.com** |
| **Theme assigned** | **Puzzle Solver Robot (GLCD)** |
| **Date** | **25-Jan-16** |

## Scope (5)

**State the scope of the theme assigned to you.**

**Answer:**

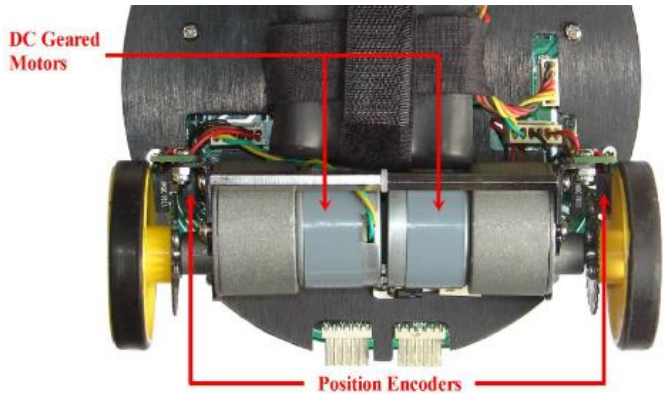
Purpose of such an application can be an ATM machine that scans and detects available notes and coins and sum up the required cash requests in such way that maximum number of requests can be granted logically and physically. Same is also applicable in a banking system. Along with, a courier delivery system requires to find the shortest path the save travelling cost. It can also be used as automatic goods management in factory where the 1st division contains the raw materials which are chosen and transmitted over shortest path to create a product in 2nd division. It can be also used in dockyards to load containers to the ships by maintaining weight-capacity.

## Building Modules (5)

**Identify the major components required for designing the robotic system for the solution of the theme assigned to you.**

**Answer**:

1. Mechanical Systems
    a. Two DC geared motors




Used to move the robot.

b. Castor wheel


It is a support for the movements and it helps to move the robot.

2. Electronic Systems
   a. Firebird V ATMEGA2560 robot

It is the main robot where all components will be assembled and then fit, It contains ATMEGA2560 as master uC and ATMEGA8 as slave uC, which we can program.

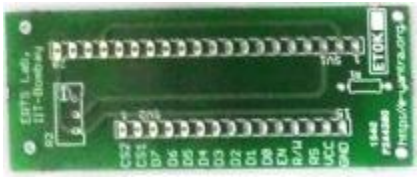
   b. Three white line sensors


White line sensors are used to follow any white line over black surface. In our case, we inverted the logic to follow black lines over white surface.

   c. GLCD

GLCD is used to display the picked up number from 1st division i.e. D1 and it also shows message "Deposit" after we deposit the picked up number to 2nd division i.e. D2. GLCD indicates pickup and deposit operation virtually on the screen. Components provided for GLCD Interfacing Circuit are show below:

      i. Circuit Board



      ii. Graphic LCD

      iii. Resistor (330 Ohms 5% tolerance)

iv. POT (Potentiometer) 10K

Used to control the contrast of the GLCD screen.

v. 2x RGB LED

Indicates pickup direction of the number from 1st division i.e. D1.

vi. 2x 150 ohm, ¼ watt resistor

To convert 5 volt to 3 volt to give proper voltage to LEDs.

vii. Female Relimate Connector (15-Pin)

viii. Male Relimate Connector (15-Pin)

ix. 20 pin male Connector

x. 20 pin female Connector

xi. M-F Jumper Wires

Used to connect the LED terminals to PG0 and PG1 of expansion slot on Firebird V robot.

3. Extra required tools
    a. Soldering iron and soldering wire

    Used for assembling GLCD components.

b. Flat blade 2mm screwdriver
Used to calibrate white line sensors and GLCD POT

## Environment sensing (5)

**Explain the functioning of environment sensing technique used by Firebird V robot in your theme.**

**Answer:**

1. **Three white line sensors:**
White line sensors are type of localization sensor which consists of highly directional photo transistor. They can detect white line on black surface or black line on white surface by inverting the logic. White line sensors are used in our theme to detect and follow 1cm thick black line on white surface. We also used them to detect 3x3cm black boxes on each corner of cells.
Three white line sensors are used in our theme. As given in rulebook, borders of cell are 1cm thick and black squares on corner of cells are 3x3cm. But the distance between two white line sensors are more than 1cm. So, to follow a 1cm black line, we can linearly forward the robot if all three white line sensors are on white surface or the center white line sensor is on the black surface. We decrease velocity of the right wheel and increase velocity of the left wheel if right white line sensor is on the black surface and other twos are on the white surface. Similarly we decrease velocity of the left wheel and increase velocity of the right wheel if left white line sensor is on the black surface and other twos are on the white surface. This way the robot follows a 1cm thick black line as shown in the Figure 1.

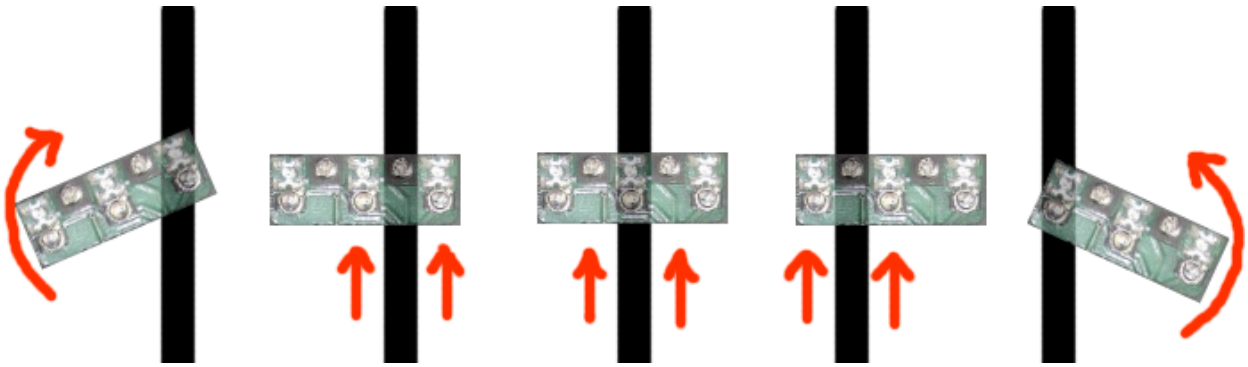


Figure 1

Conditions for detecting 3x3cm black squares are below (Figure 2):

a. If right and center white line sensors are on black surface and left white line sensor is on white surface, we also detect it as a 3x3cm black square.
b. If left and center white line sensors are on the black surface and right white line sensor is on white surface, we detect it as a 3x3cm black square.
c. Again, if all three white line sensors are on black surface, we also detect it as a 3x3cm black square.

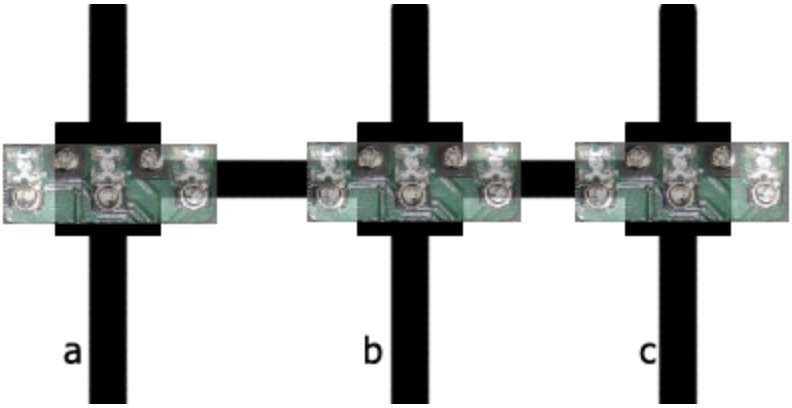
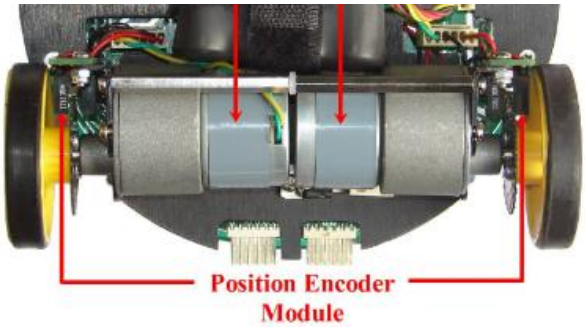



Figure 2

2. **Two position encoders:**
   Position encoders are used to measure amount of rotation of two wheels separately i.e. position and velocity feedback. It consists of slotted disk which rotates between optical encoder and cuts it path, hence it generates square wave. There are 30 slots on a disk, so we get 30 pulses if a wheel rotates perfectly 1 time. Pulse rate gives are velocity feedback.



## Power Management (5)

**Explain the power management system required for a robot in general and for Firebird V robot in particular.**

**Answer:**

Firebird V Atmega2560 is powred by on board rechargeable 9.6V, 2.1Ah Nickel Metal Hydride (NiMH) battery. The battery voltage can vary between 12 volts (when fully charged) to 8 volts (when discharged). The robot can also be powered by auxiliary power supply. When battery pack is connected, the robot can use maximum 2 Amp current, and when we connect the auxiliary power, the robot can use maximum 1 Amp current. On the battery power, the robot can run up to 2 hours when motors are operational at 75% of time. The robot has an on/off switch with two modes BP (Battery Power) and AP (Auxiliary Power). When use battery, the AP is treated as off and vice versa. We can use any one power source at a time.

The power switch is shown in Figure 3 below:

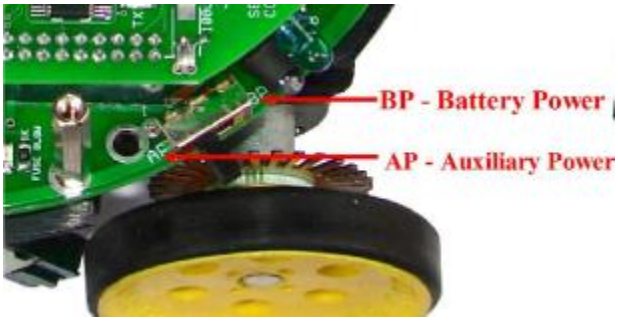



Figure 3

The battery pack and battery connector is shown in Figure 4 below:

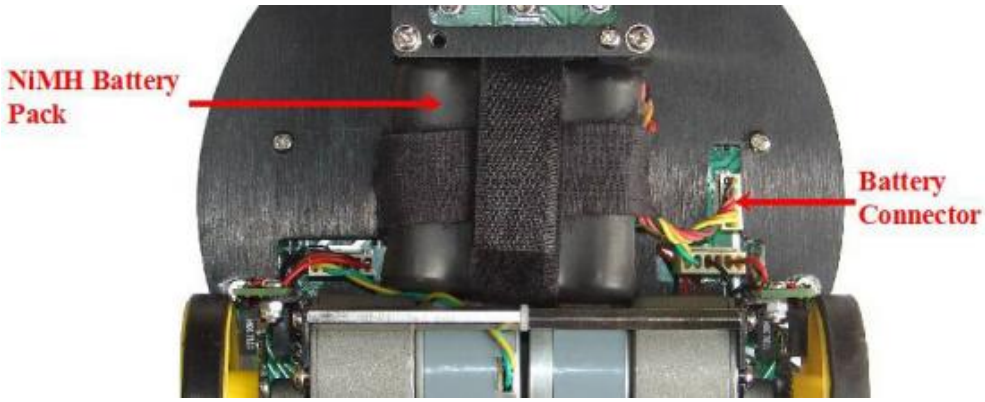



Figure 4

## Testing your knowledge(related to rulebook) (30)

**Describe the process of completing the single Number in D2, by adding different Numbers from D1.**

**Answer:**

- After transferring python generated data, the robot starts traversing.
- It goes to the nearest among 4 points (3x3cm black squares) of cell in D1 that has number to be picked.
- It forwards up to the middle of cell boundary, pickup the number by showing it on GLCD and glows the LED of pickup side.

- Now the robot crosses the bridge and goes to nearest cell point of the cell in D2 that number to be deposited on.
- The robot forwards to be middle of cell boundary in such direction that pickup direction matches deposit direction.
- Then it shows message “Deposit” on GLCD and back to pick next operand from D1.
- When all operand for a number in D2 gets completed, then it buzzes for 1000ms.

**Did you burn the demo code on Firebird V and test the GLCD?**

**Answer**:

Yes, we burned the demo code on Firebird V to test the GLCD.

The GLCD has following pin configurations:

| GLCD Connections | Microcontroller Pins | Pin number on expansion slot |
|---|---|---|
| **CS1** | PL0 | 28 |
| **CS2** | PL1 | 27 |
| **RS** | PD5 | 29 |
| **RW** | PD6 | 33 |
| **EN** | PD7 | 34 |
| **D7** | PJ7 | 40 |
| **D6** | PJ6 | 39 |
| **D5** | PJ5 | 42 |
| **D4** | PJ4 | 41 |
| **D3** | PJ3 | 44 |
| **D2** | PJ2 | 43 |
| **D1** | PJ1 | 46 |
| **D0** | PJ0 | 45 |
| **VCC** | - | 21 |
| **GND** | - | 23 |

Before connecting the GLCD to the expansion socket, we removed the jumper J3 to disable the Bargraph LED and use PJ-PJ7 as GPIO. The expansion socket is numbered as Figure 5. The top row has even numbers and bottom row has odd numbers.

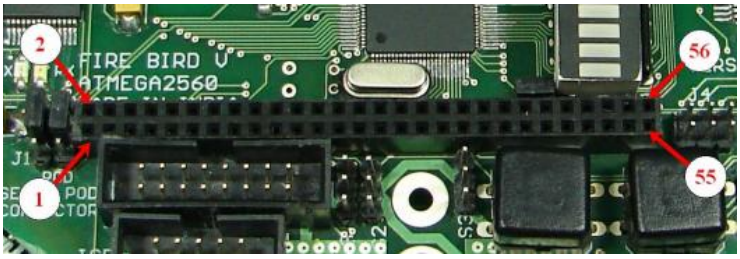
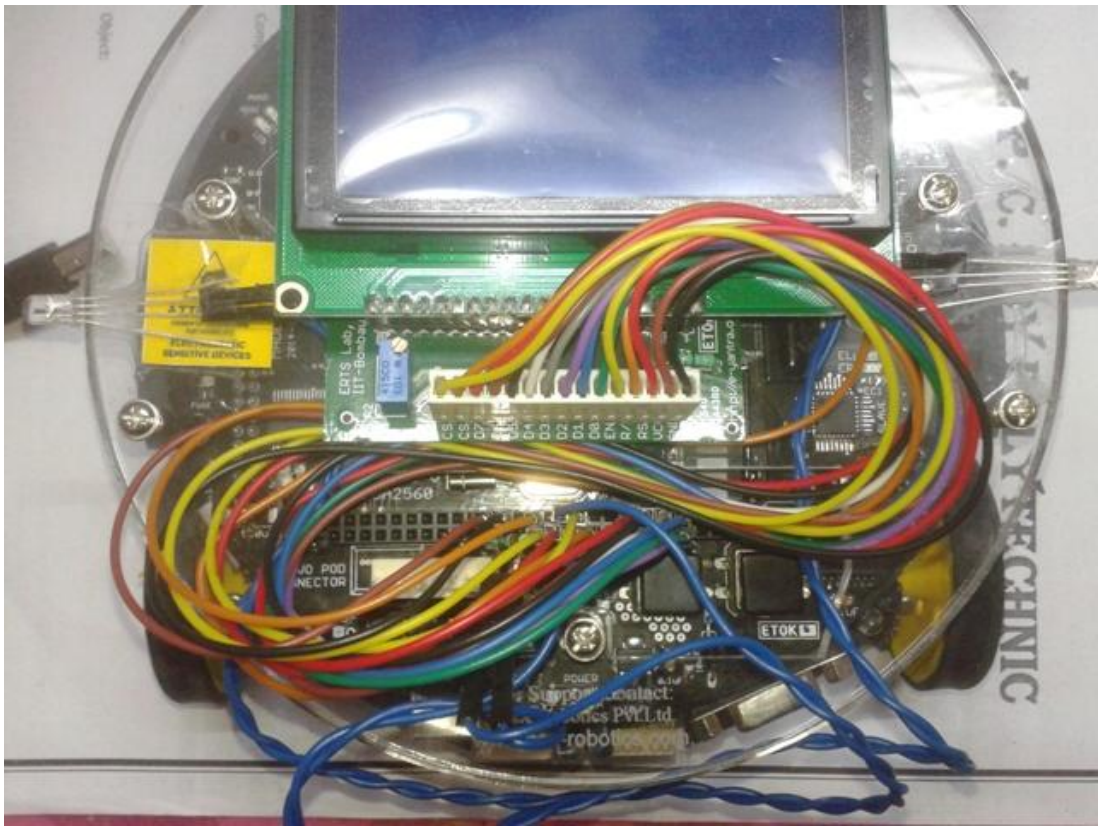
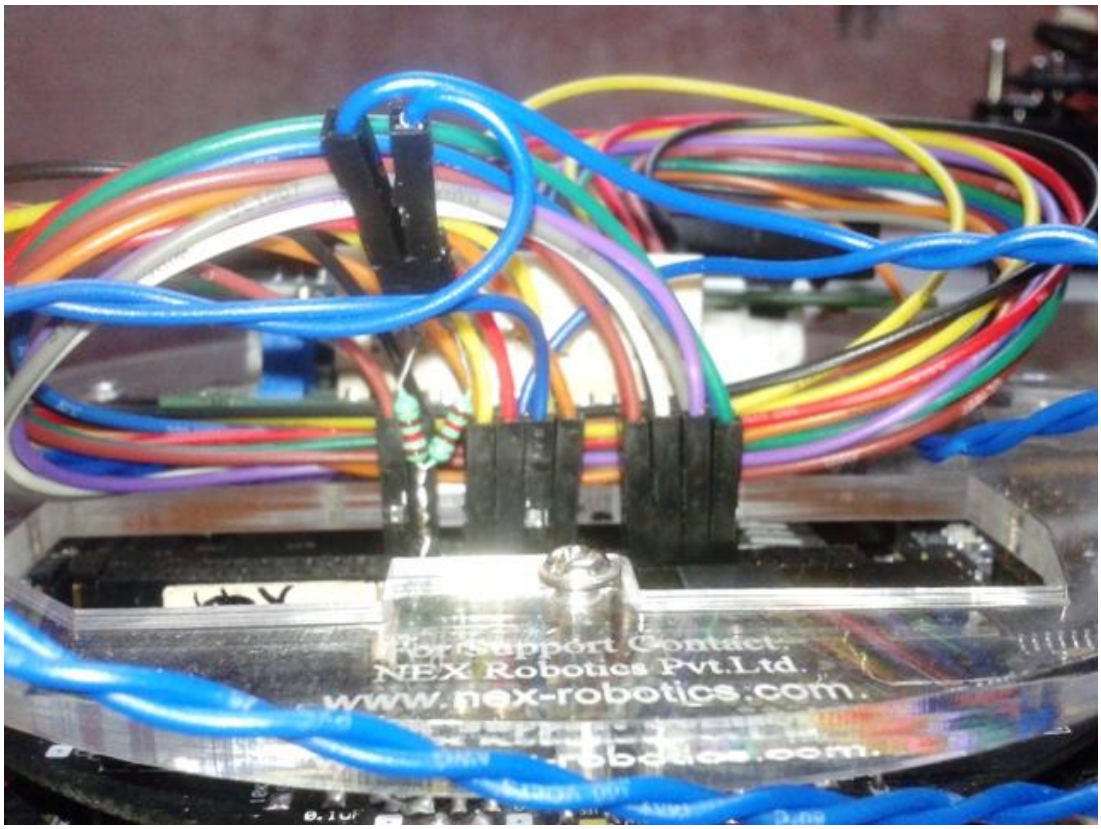



Figure 5

Then we connect other ends of male Relimate connector i.e. Male jumpers to the expansion socket as below.






PG0 and PG1 are used as GPIO to control left and right LEDs respectively.

## Algorithm Analysis (40)

**Q-1 Draw a flowchart illustrating the major functions that are used.**

**Answer**:

At first, we need to map the grids virtually to addressing cells within. The mapping is given below in Figure 6. We will store these data into global arrays and then use them in our functions, flowcharts.

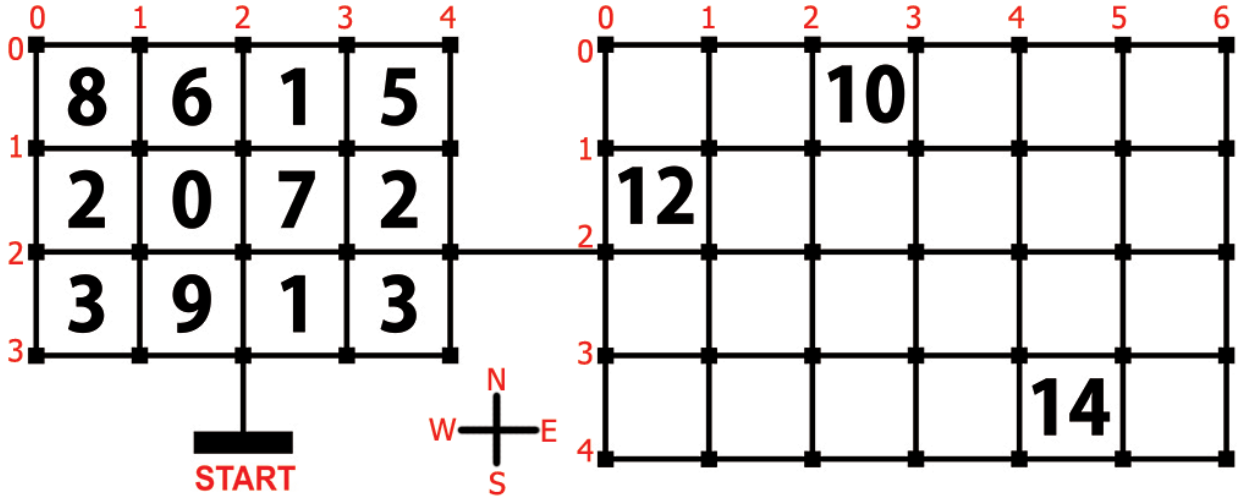



Figure 6

**Global variables that are accessed by all functions and flowcharts**:

```
// mapping / setting cell co-ordinates in D1
// 12 cells, 4 points in each cell, 2 index for row-column in each point
// 4 points for each cell are sorted in this order: top-left, top-right, bottom-right,
bottom-left
integer array d1_position_map[12][4][2] = {
       {{0, 0}, {0, 1}, {1, 1}, {1, 0}}, {{0, 1}, {0, 2}, {1, 2}, {1, 1}},
       {{0, 2}, {0, 3}, {1, 3}, {1, 2}}, {{0, 3}, {0, 4}, {1, 4}, {1, 3}},
       {{1, 0}, {1, 1}, {2, 1}, {2, 0}}, {{1, 1}, {1, 2}, {2, 2}, {2, 1}},
       {{1, 2}, {1, 3}, {2, 3}, {2, 2}}, {{1, 3}, {1, 4}, {2, 4}, {2, 3}},
       {{2, 0}, {2, 1}, {3, 1}, {3, 0}}, {{2, 1}, {2, 2}, {3, 2}, {3, 1}},
       {{2, 2}, {2, 3}, {3, 3}, {3, 2}}, {{2, 3}, {2, 4}, {3, 4}, {3, 3}}
};
// mapping / setting cell co-ordinates in D2
// 24 cells, 4 points in each cell, 2 index for row-column in each point
// 4 points for each cell are sorted in this order: top-left, top-right, bottom-right,
bottom-left
integer array d2_position_map[24][4][2] = {
       {{0, 0}, {0, 1}, {1, 1}, {1, 0}}, {{0, 1}, {0, 2}, {1, 2}, {1, 1}},
       {{0, 2}, {0, 3}, {1, 3}, {1, 2}}, {{0, 3}, {0, 4}, {1, 4}, {1, 3}},
       {{0, 4}, {0, 5}, {1, 5}, {1, 4}}, {{0, 5}, {0, 6}, {1, 6}, {1, 5}},
       {{1, 0}, {1, 1}, {2, 1}, {2, 0}}, {{1, 1}, {1, 2}, {2, 2}, {2, 1}},
       {{1, 2}, {1, 3}, {2, 3}, {2, 2}}, {{1, 3}, {1, 4}, {2, 4}, {2, 3}},
       {{1, 4}, {1, 5}, {2, 5}, {2, 4}}, {{1, 5}, {1, 6}, {2, 6}, {2, 5}},
       {{2, 0}, {2, 1}, {3, 1}, {3, 0}}, {{2, 1}, {2, 2}, {3, 2}, {3, 1}},
       {{2, 2}, {2, 3}, {3, 3}, {3, 2}}, {{2, 3}, {2, 4}, {3, 4}, {3, 3}},
       {{2, 4}, {2, 5}, {3, 5}, {3, 4}}, {{2, 5}, {2, 6}, {3, 6}, {3, 5}},
       {{3, 0}, {3, 1}, {4, 1}, {4, 0}}, {{3, 1}, {3, 2}, {4, 2}, {4, 1}},
       {{3, 2}, {3, 3}, {4, 3}, {4, 2}}, {{3, 3}, {3, 4}, {4, 4}, {4, 3}},
       {{3, 4}, {3, 5}, {4, 5}, {4, 4}}, {{3, 5}, {3, 6}, {4, 6}, {4, 5}}
};
```

```
integer current_velocity = 127;          // default velocity 100
character current_direction = 'N';       // E/W/N/S
character pickup_direction = '';         // values can be L or R i.e. left or right
respectively
integer current_grid = -1;               // 1, 2 i.e. D1, D2, initially invalid
integer current_cell_no = -1;            // initially a invalid one
integer current_coordinate[2] = {-1, -1}; // co-ordinate of the cell initially invalid

float left_velocity_float, right_velocity_float;
integer left_velocity, right_velocity;
```

**Functions, Flowcharts, and Algorithm legends:**

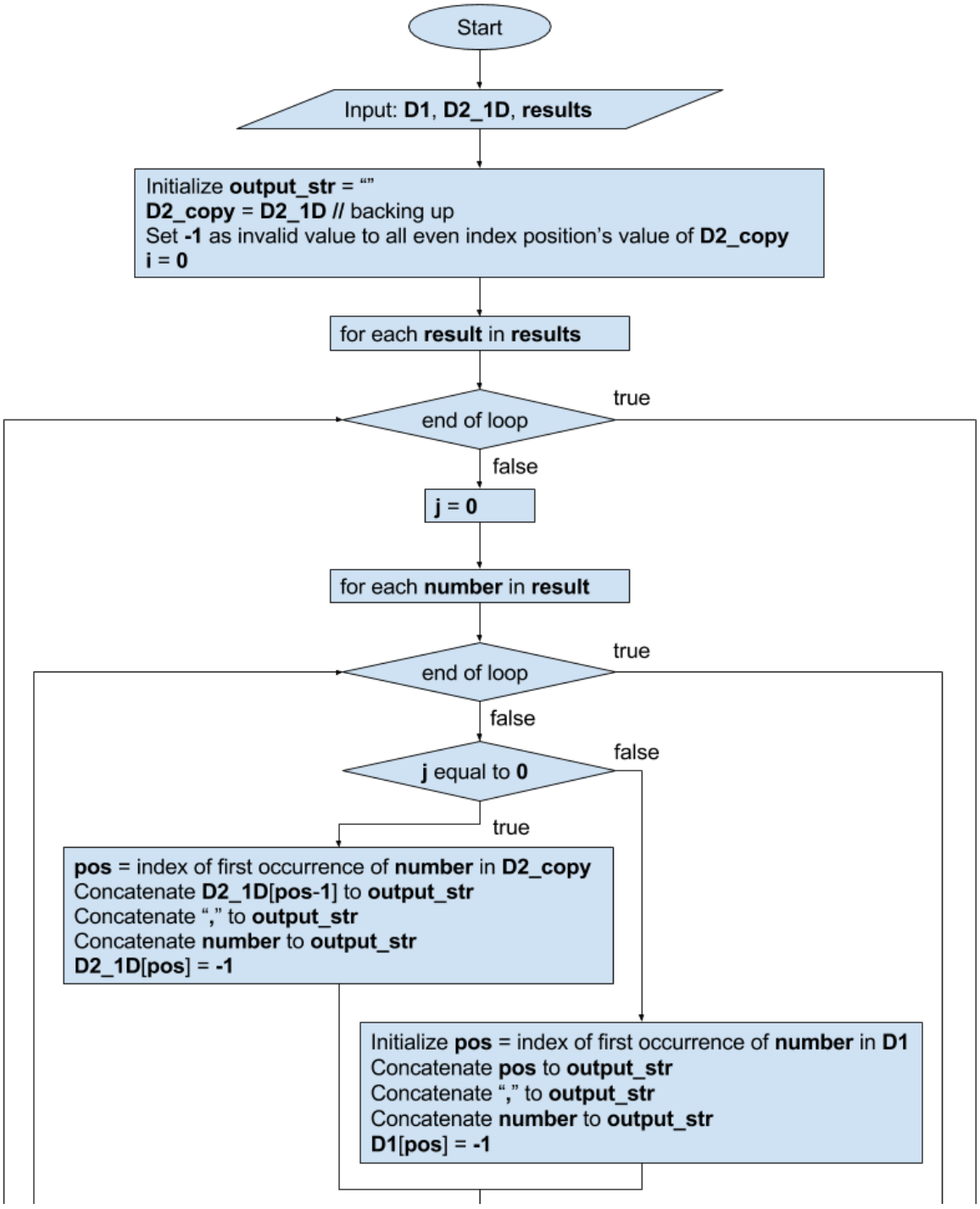
- Function, Algorithm names are in this format: **algorithm_name**
- Variables, algorithm names, constants, constructs are written in **bold**.
- Comments are in this format: *(This is a comment)* or // This is a comment

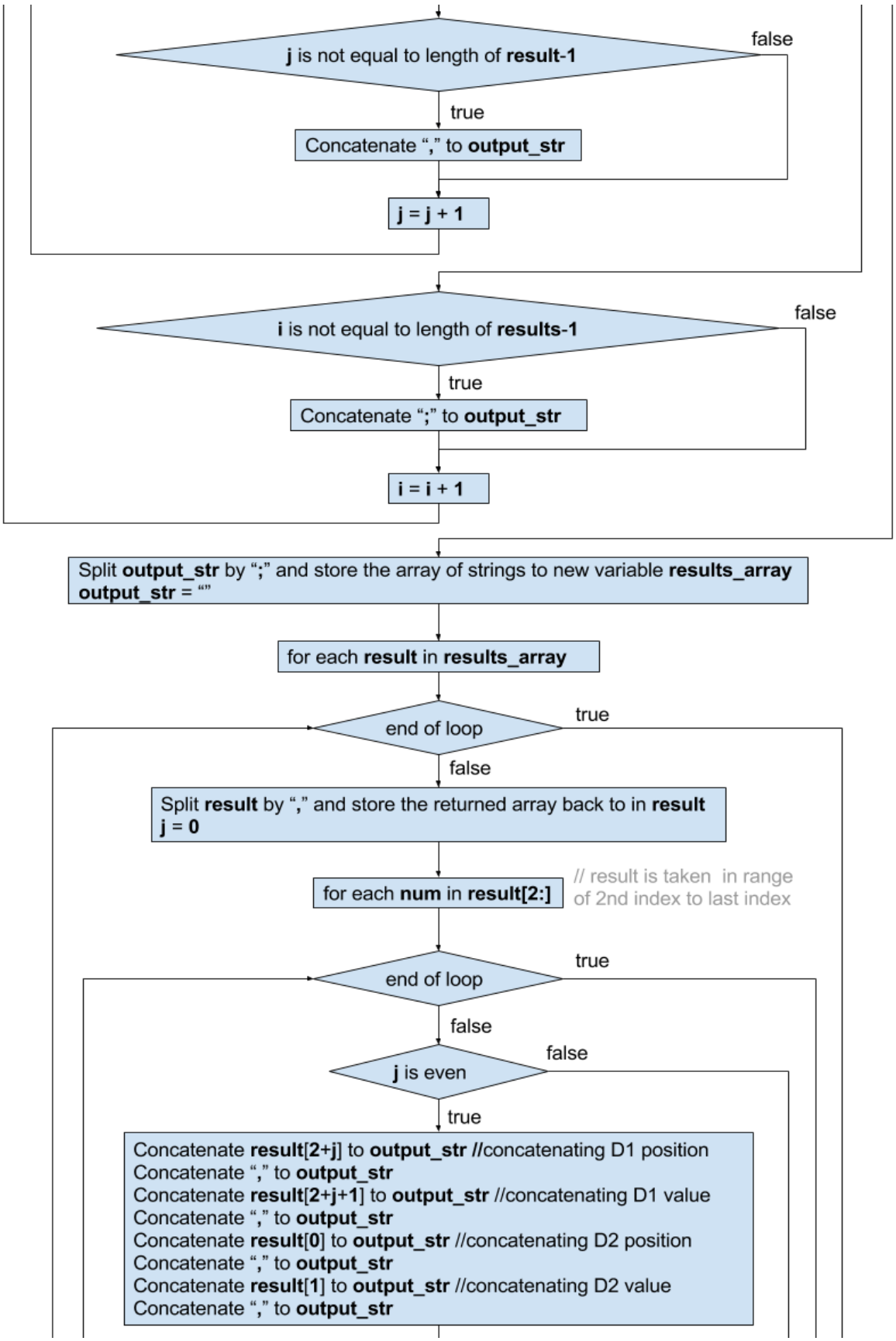
Flowchart images are provided separately in '**flowcharts'** directory for ease.

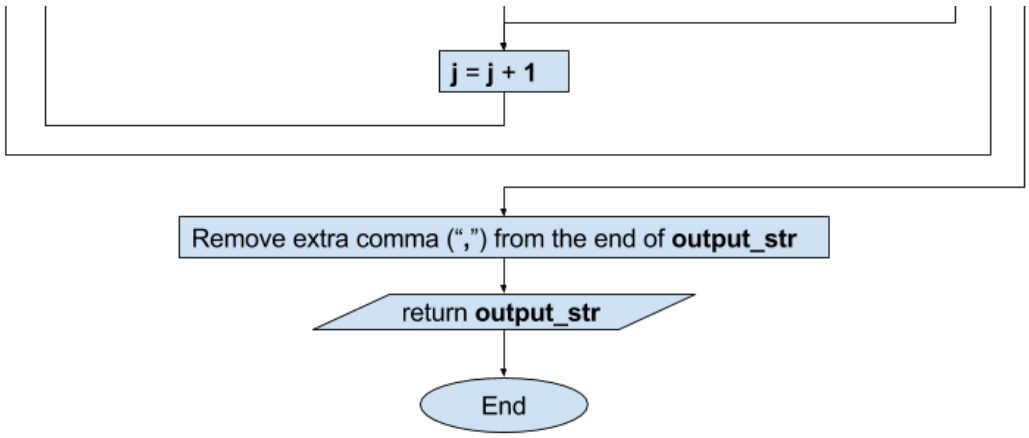
**Flowcharts for the major functions that are used**:

- Function **generate_traversal_path** (int array **D1**, int array **D2_1D**, int array_2d **results**):
  - **Details**:
    **D1** and **D2** are arrays of integer numbers, **results** is a 2D array of integer numbers provided to the algorithm. This function generates traversal path of the robot on arena.

**Flowchart**: **generate_traversal_path**




j is not equal to length of result-1
false
true
Concatenate "," to output_str
j = j + 1
i is not equal to length of results-1
false
true
Concatenate ";" to output_str
i = i + 1
Split output_str by ";" and store the array of strings to new variable results_array
output_str = ""
for each result in results_array
end of loop
true
false
Split result by "," and store the returned array back to in result
j = 0
for each num in result[2:]
// result is taken in range of 2nd index to last index
end of loop
true
false
j is even
false
true
Concatenate result[2+j] to output_str //concatenating D1 position
Concatenate "," to output_str
Concatenate result[2+j+1] to output_str //concatenating D1 value
Concatenate "," to output_str
Concatenate result[0] to output_str //concatenating D2 position
Concatenate "," to output_str
Concatenate result[1] to output_str //concatenating D2 value
Concatenate "," to output_str



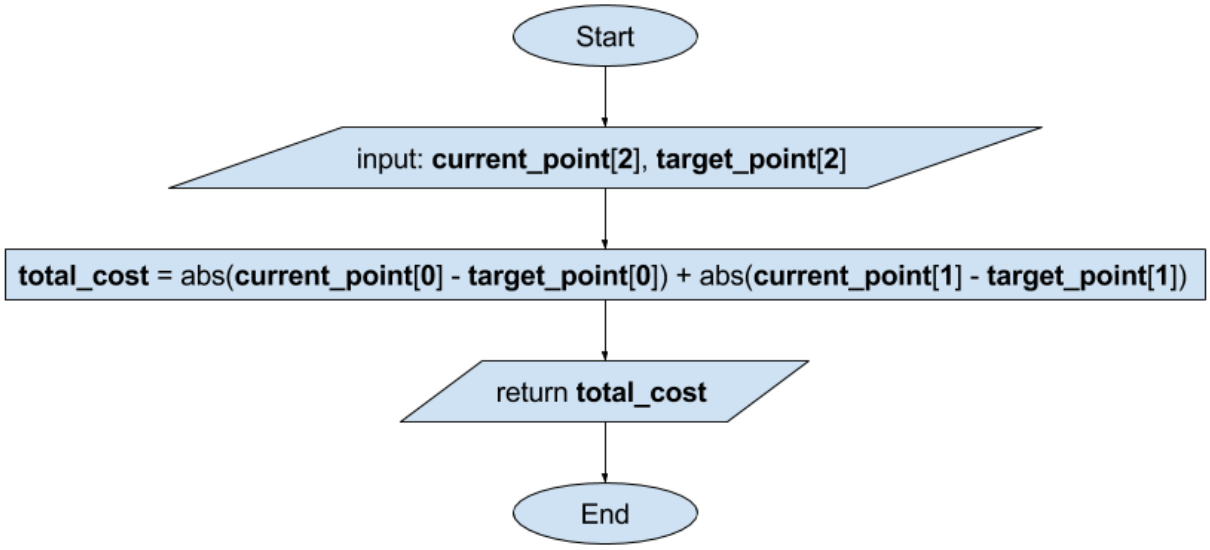

**Algorithm**: generate_traversal_path

1. **Step 1:**
    a. Initialize **output_str** as an empty string
    b. Initialize **D2_copy** and copy **D2_1D** to **D2_copy**
    c. Set **-1** as invalid value to all even index position's value of **D2_copy**
2. **Step 2**:
    *(generates solution set in this form: D2 position, D2 value, D1 position, D1 value, D1 position, D1 value... rest operands in D1;*
    *D2 position, D2 value, D1 position, D1 value, D1 position, D1 value...; ... rest solution set in D2, D1*
    *e.g. - 1,16,3,8,10,8;10,14,1,9,2,5;23,10,6,7,7,3 for test_image2.jpg)*
    a. Initialize **i** = **0**
    b. **for** (each **result** in **results**), do as following:
        i. Initialize **j** = **0**
        ii. **for** (each **number** in **result**), do as following:
            1. **if** (**j** is equal to **0**), do as following:
                a. Initialize **pos** = index of first occurrence of **number** in **D2_copy**
                b. Concatenate **D2_1D**[**pos**-**1**] to **output_str**
                c. Concatenate "**,**" to **output_str**
                d. Concatenate **number** to **output_str**
                e. **D2_1D**[**pos**] = **-1**
            2. **else**, do as following:
                a. Initialize **pos** = index of first occurrence of **number** in **D1**
                b. Concatenate **pos** to **output_str**
                c. Concatenate "**,**" to **output_str**
                d. Concatenate **number** to **output_str**
                e. **D1**[**pos**] = **-1**
            3. **if** (**j** is not equal to length of **result**-**1**), do as following:
                a. Concatenate "**,**" to **output_str**
            4. **j** = **j** + **1**
        iii. **if** (**i** is not equal to length of **results**-**1**), do as following:
            1. Concatenate "**;**" to **output_str**
        iv. **i** = **i** + **1**

3. **Step 3:**
    a. Split **output_str** by "**;**" and store the array of string to new variable **results_array**
    b. **output_str** = ""
4. **Step 4**:
    *(generating the traversal path of the robot in comma separated format*
    *D1 position, D1 value, D2 position, D2 value, D1 position, D1 value ...*
    *e.g. - 3,8,1,16,10,8,1,16,1,9,10,14,2,5,10,14,6,7,23,10,7,3,23,10 for test_image2.jpg)*
    a. **for** (each **result** in **results_array**), do as following:
        i. Split **result** by "**,**" and store the returned array back to in **result**
        ii. **j** = **0**
        iii. **for** (each **num** in **result**[**2:**]), do as following:
        *(result is taken in range of 2nd index to last index)*
            1. **if** (**j** is even), do as following:
                *a.* Concatenate **result**[**2**+**j**] to output_str
                *(concatenating D1 position)*
                b. Concatenate "**,**" to **output_str**
                c. Concatenate **result**[**2**+**j**+**1**] to output_str
                *(concatenating D1 value)*
                d. Concatenate "**,**" to **output_str**
                e. Concatenate **result**[**0**] to output_str
                *(concatenating D2 position)*
                f. Concatenate "**,**" to **output_str**
                g. Concatenate **result**[**1**] to output_str
                *(concatenating D2 value)*
                h. Concatenate "**,**" to **output_str**
            2. **j** = **j** + **1**
    b. Remove extra comma ("**,**") from the end of **output_str**
    c. return **output_str**
5. **Step 5:**
    a. End of algorithm **generate_traversal_path**

- Function **get_point_cost** (int array **current_point**, int array **target_point**):
    - **Details**:
      **current_point** and **target_point** are arrays of integer numbers, each contains two elements i.e. row and column of a point's co-ordinate. This function returns traversal cost between two points based on distance.

**Flowchart**: **get_point_cost**

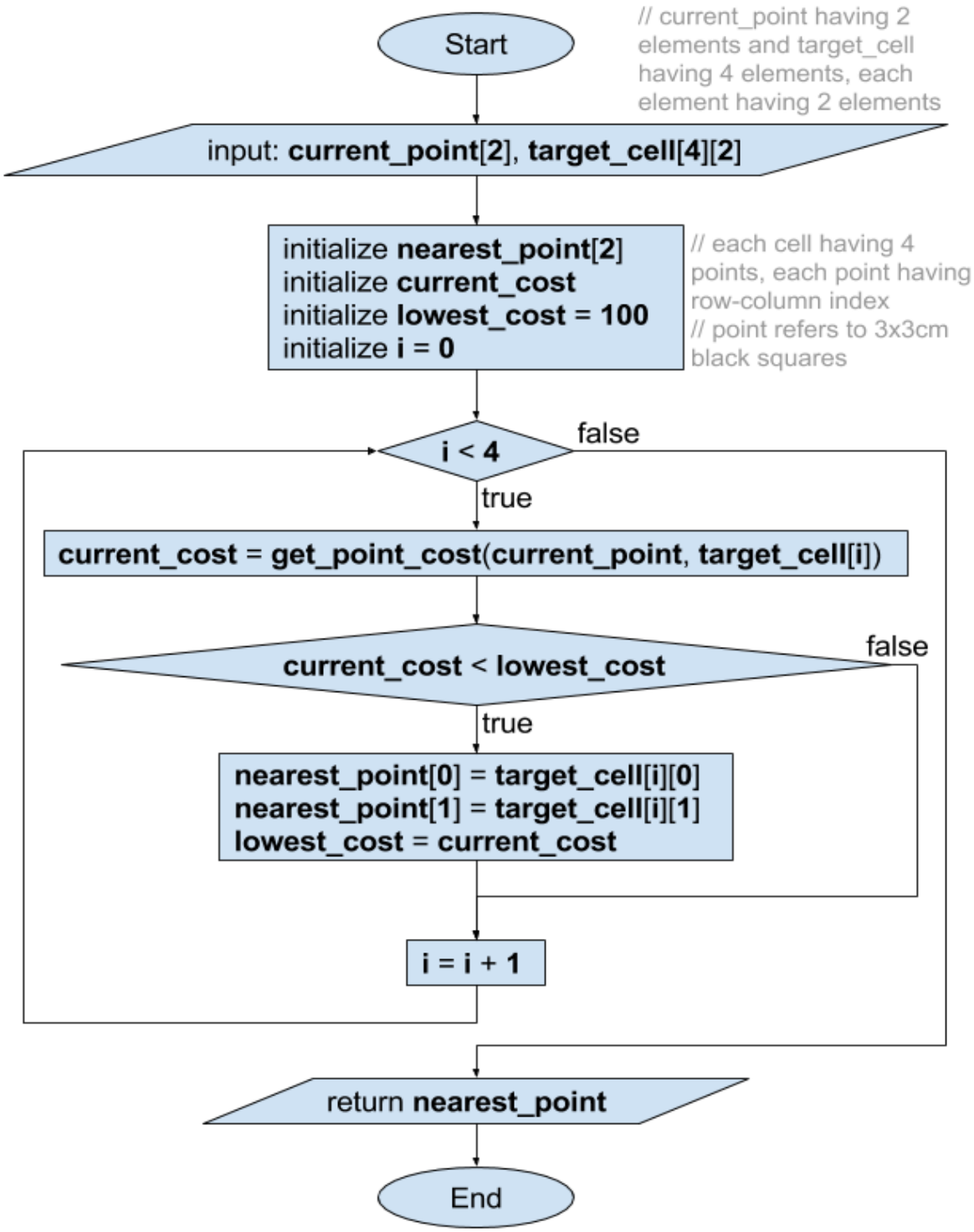



**Algorithm**: **get_point_cost**

1. **Step 1**:
    a. Take **current_point** and **target_poing** as input.
2. **Step 2**:
    a. **total_cost** = absolute value of (**current_point**[**0**] – **target_point**[**0**])
    + absolute value of (**current_point**[**1**] – **target_point**[**1**])
    *(row difference + column difference)*
3. **Step 3**:
    a. Return **total_cost**
4. **Step 4**:
    a. End of algorithm **get_point_cost**

- Function **get_nearest_point** (int array **current_point**, int array **target_cell**)
    - **Details**:
      **current_point** is an integer array and **target_cell** is a 2D arrays of integer arrays i.e. points, each contains two elements i.e. row and column of a point's co-ordinate. This function finds the nearest point of a cell.

**Flowchart**: **get_nearest_point**

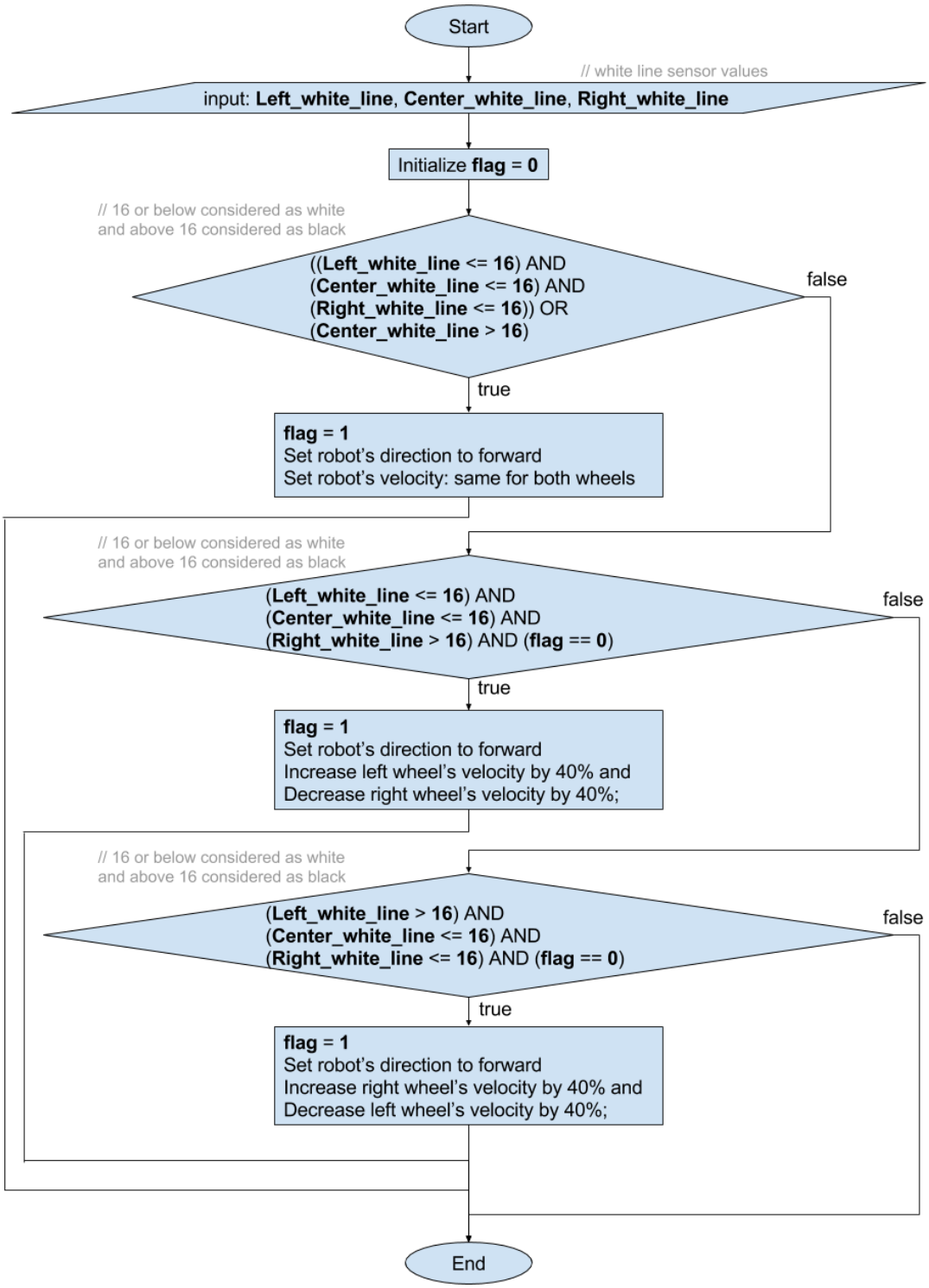


**Algorithm**: get_nearest_point

1. **Step 1**:
    a. Take input **current_point** and **target cell**
    *(**current_point** having 2 elements i.e. row and column. **target_cell** having 4 elements i.e. 4 points of a cell, each elements having 2 elements i.e. row and column)*
    b. Initialize **nearest_point**
    c. Initialize **current_cost**
    d. Initialize **lowest_cost** = **100**
    e. Initialize **i** = **0**
2. **Step 2**:
    a. **if** (**i** < **4**), then do as following:
        i. **current_cost** = get_point_cost(**current_point**, **target_cell**[**i**])
        ii. **if** (**current_cost** < **lowest_cost**), then do as following:
            1. **nearest_point**[**0**] = **target_cell**[**i**][**0**]
            2. **nearest_point**[**1**] = **target_cell**[**i**][**1**]
            3. **lowest_cost** = **current_cost**
        iii. **i** = **i** + **1**
        iv. Repeat **Step 2: a.**
3. **Step 3**:
    a. Return **nearest_point**
4. **Step 4:**
    a. End of algorithm get_nearest_point

- Function **follow_black_line** (int **Left_white_line**, int **Center_white_line**, int **Right_white_line**)
    - **Details**:
      **Left_white_line**, **Center_white_line**, and **Right_white_line** are 3 integer numbers. This function follows a 1 cm thick black line on white surface.

**Flowchart**: **follow_black_line**

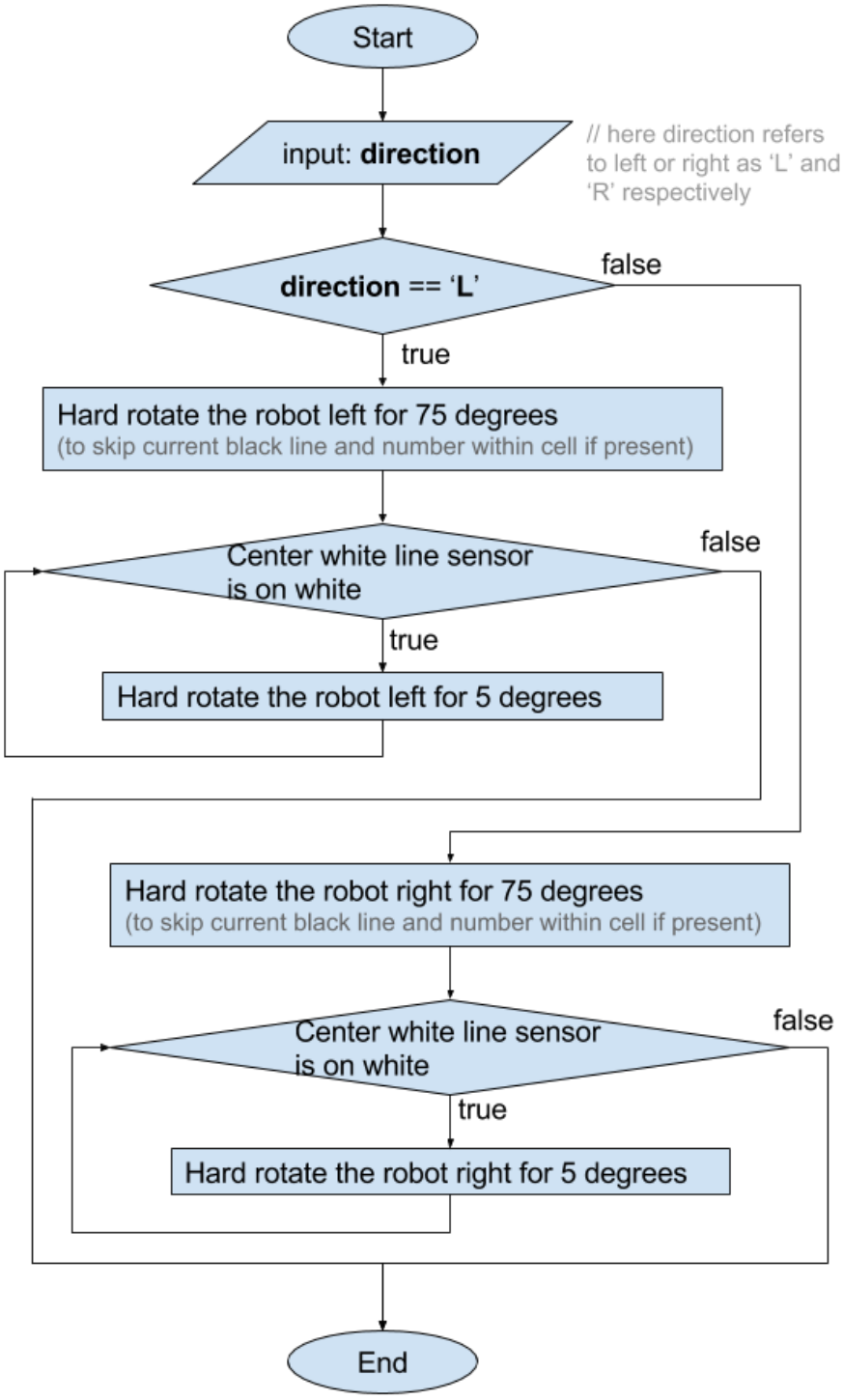


**Algorithm**: **follow_black_line**

1. **Step 1**:
    a. Input **Left_white_line**, **Center_white_line**, **Right_white_line**
    *(these are white line sensor values)*
    b. Initialize **flag** = **0**
2. **Step 2**:
    a. **if** (((**Left_white_line** <= **16**) AND (**Center_white_line** <= **16**) AND (**Right_white_line** <= **16**)) OR (**Center_white_line** > **16**)), then do as following:
    *(16 or below considered as white and above 16 considered as black)*
        i. **flag** = **1**
        ii. Set robot's direction to forward
        iii. Set robot's velocity: same for both wheels
3. **Step 3**:
    a. **if** (((**Left_white_line** <= **16**) AND (**Center_white_line** <= **16**) AND (**Right_white_line** > **16**)) AND (**flag** == **0**)), then do as following:
    *(16 or below considered as white and above 16 considered as black)*
        i. **flag = 1**
        ii. Set robot's direction to forward
        iii. Increase left wheel's velocity by 40% and
        iv. Decrease right wheel's velocity by 40%
4. **Step 4**:
    a. **if** (((**Left_white_line** > **16**) AND (**Center_white_line** <= **16**) AND (**Right_white_line** <= **16**)) AND (**flag** == **0**)), then do as following:
    *(16 or below considered as white and above 16 considered as black)*
        i. **flag = 1**
        ii. Set robot's direction to forward
        iii. Increase right wheel's velocity by 40% and
        iv. Decrease left wheel's velocity by 40%
5. Step 5:
    a. End of algorithm **follow_black_line**

- Function **turn_robot** (char **direction**)
    - **Details**:
      **direction** is a character type variable. char can be '**L**' or '**R**'. This function. Rotates the robot perfect 90 degrees. The robot will rotate 75 degrees first to skip current black line and number present within cell and then rotates 5 degrees until encounter a black line.

**Flowchart**: **turn_robot**

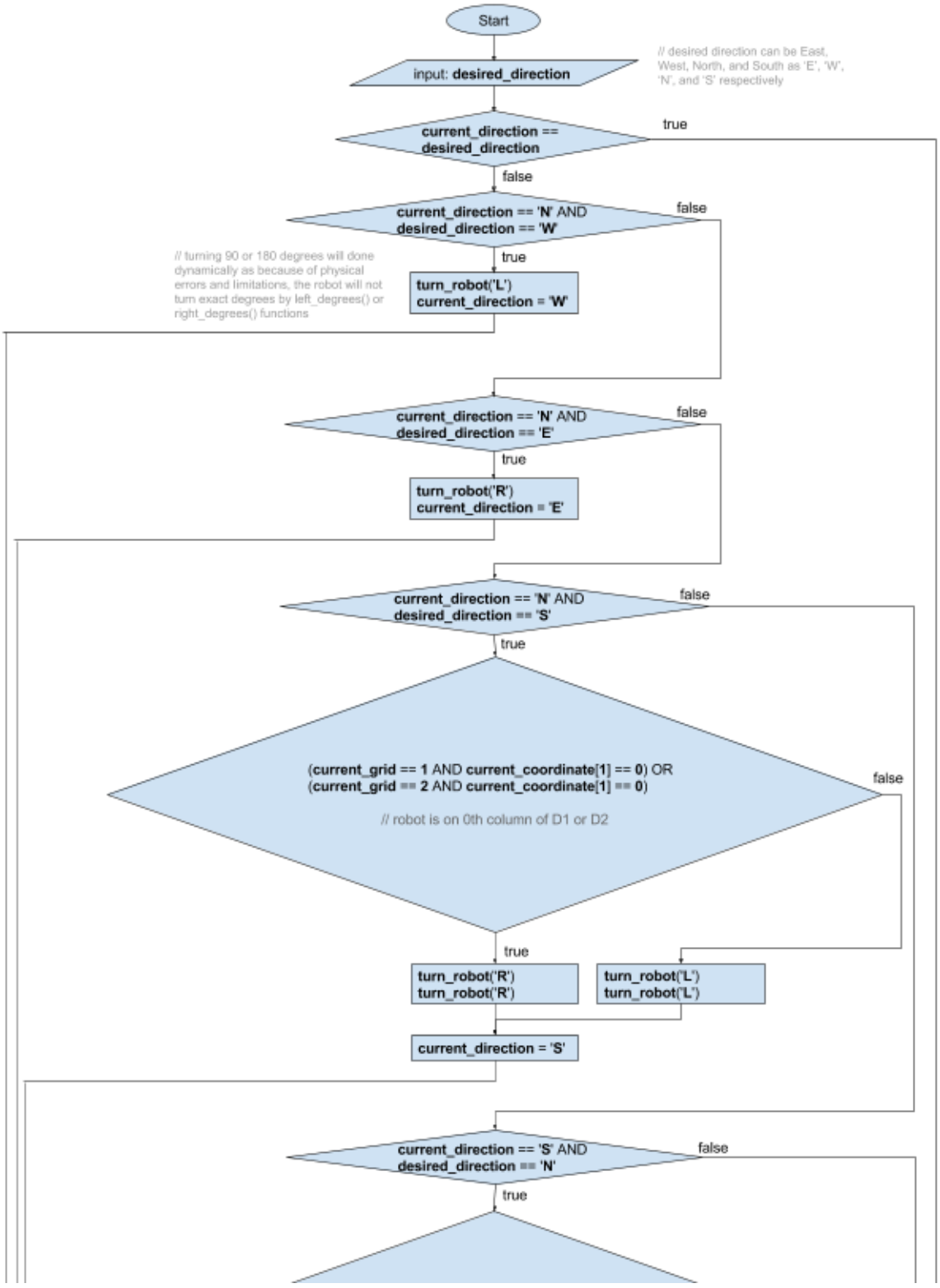


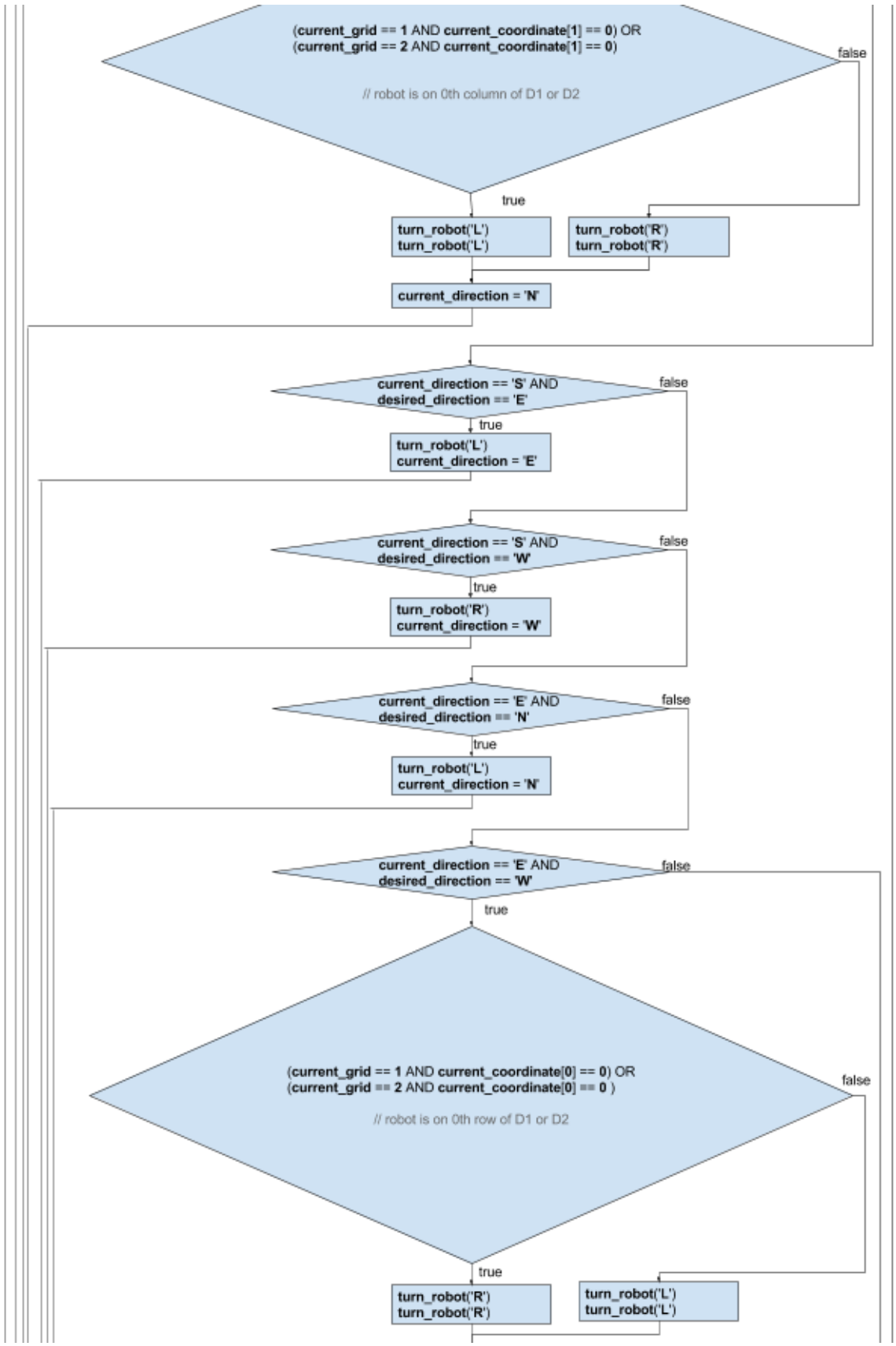
**Algorithm**: **turn_robot**

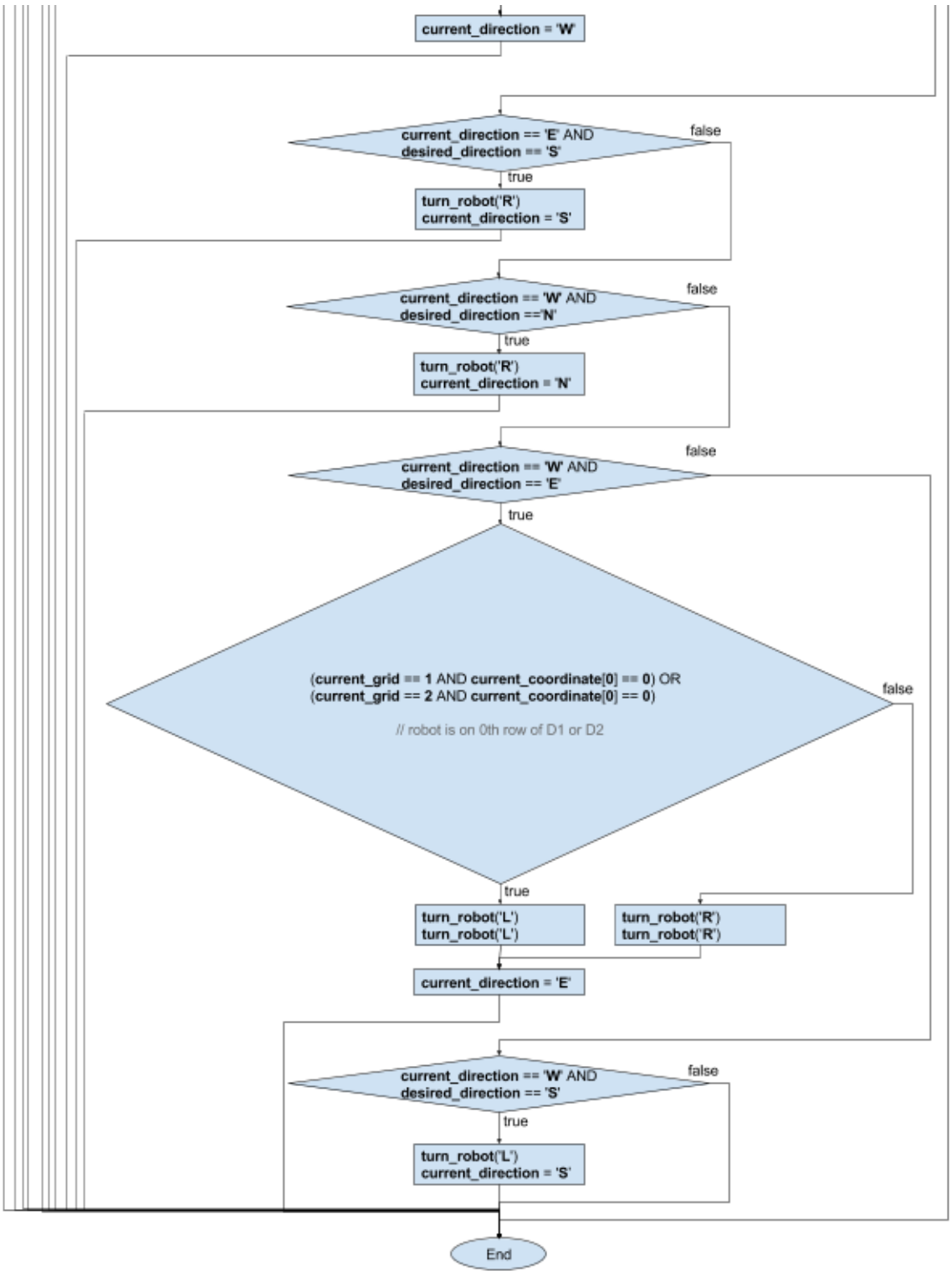
1. **Step 1**:
    a. Input **direction**
    *(here direction refers to left or right as 'L' and 'R' respectively)*
2. **Step 2**:
    a. **if** (**direction** == '**L**'), then do as following:
        i. Hard rotate the robot left for 75 degrees
        *(to skip current black line and number within cell if present)*
        ii. **if** (Center white line sensor is on white), do as following:
            1. Hard rotate the robot left for 5 degrees
            2. Go to **Step 2**, **a**, **ii**.
        iii. **else**, do as following:
            1. Go to **Step 3**
    b. **else**, do as following:
        i. Hard rotate the robot right for 75 degrees
        *(to skip current black line and number within cell if present)*
        ii. **if** (Center white line sensor is on white), do as following:
            1. Hard rotate the robot right for 5 degrees
            2. Go to **Step 2**, **b**, **ii**.
        iii. **else**, do as following:
            1. Go to **Step 3**
3. **Step 3**:
    a. End of algorithm **turn_robot**

- Function **change_direction** (character **desired_direction**)
    - **Details**:
      **desired_direction** is a character type variable. The values can be ‘E’, ‘W’, ‘N’, or ‘S’ for Ease, West, North, and South respectively. This function changes direction of the robot.

**Flowchart**: **change_direction**




(current_grid == 1 AND current_coordinate[1] == 0) OR
(current_grid == 2 AND current_coordinate[1] == 0)
// robot is on 0th column of D1 or D2
false
true
turn_robot('L')
turn_robot('L')
turn_robot('R')
turn_robot('R')
current_direction = 'N'
current_direction == 'S' AND
desired_direction == 'E'
false
true
turn_robot('L')
current_direction = 'E'
current_direction == 'S' AND
desired_direction == 'W'
false
true
turn_robot('R')
current_direction = 'W'
current_direction == 'E' AND
desired_direction == 'N'
false
true
turn_robot('L')
current_direction = 'N'
current_direction == 'E' AND
desired_direction == 'W'
false
true
(current_grid == 1 AND current_coordinate[0] == 0) OR
(current_grid == 2 AND current_coordinate[0] == 0 )
// robot is on 0th row of D1 or D2
false
true
turn_robot('R')
turn_robot('R')
turn_robot('L')
turn_robot('L')




**Algorithm**: change_direction

1. **Step 1:**
    a. Input **desired_direction** *(can be 'E', 'W', 'N', or 'S')*
2. **Step 2:**
    a. **if** (**current_direction** == **desired_direction**), do as following:

i. Go to **Step 15**

3. **Step 3**:
   a. **if** (**current_direction** == '**N**' AND **desired_direction** == '**W**'), then do as following:
      i. **turn_robot**('**L**') *// Turn left 90 degrees*
      ii. **current_direction** = '**W**'
      iii. Go to **Step 15**
   b. **else**, do as following:
      i. Go to **Step 4**
4. **Step 4**:
   a. **if** (**current_direction** == '**N**' AND **desired_direction** == '**E**'), then do as following:
      i. **turn_robot**('**R**') *// Turn right 90 degrees*
      ii. **current_direction** = '**E**'
      iii. Go to **Step 15**
   b. **else**, do as following:
      i. Go to **Step 5**
5. **Step 5**:
   a. **if** (**current_direction** == '**N**' AND **desired_direction** == '**S**'), then do as following:
      i. **if** ((**current_grid** == **1** AND **current_coordinate**[**1**] == **0**)
      OR (**current_grid** == **2** AND **current_coordinate**[**1**] == **0**)
      ), then do as following:
         *// Turn right 180 degrees*
         1. **turn_robot**('**R**') *// Turn right 90 degrees*
         2. **turn_robot**('**R**') *// Turn right 90 degrees*
      ii. **else**, do as following:
         *// Turn left 180 degrees*
         1. **turn_robot**('**L**') *// Turn left 90 degrees*
         2. **turn_robot**('**L**') *// Turn left 90 degrees*
      iii. **current_direction** = '**S**'
      iv. Go to **Step 15**
   b. **else**, do as following:
      i. Go to last **Step 6**
6. **Step 6**:
   a. **if** (**current_direction** == '**S**' AND **desired_direction** == '**N**'), then do as following:
      i. **if** ((**current_grid** == **1** AND **current_coordinate**[**1**] == **0**)
      OR (**current_grid** == **2** AND **current_coordinate**[**1**] == **0**)
      ), then do as following:
         *// Turn left 180 degrees*
         1. **turn_robot**('**L**') *// Turn left 90 degrees*
         2. **turn_robot**('**L**') *// Turn left 90 degrees*
      ii. **else**, do as following:
         *// Turn right 180 degrees*
         1. **turn_robot**('**R**') *// Turn right 90 degrees*
         2. **turn_robot**('**R**') *// Turn right 90 degrees*
      iii. **current_direction** = '**N**'
      iv. Go to **Step 15**
   b. **else**, do as following:
      i. Go to last **Step 7**
7. **Step 7**:

a. **if** (**current_direction** == '**S**' AND **desired_direction** == '**E**'), then do as following:
    i. **turn_robot**('**L**') *// Turn left 90 degrees*
    ii. **current_direction** = '**E**'
    iii. Go to **Step 15**
b. **else**, do as following:
    i. Go to **Step 8**

8. **Step 8**:
    a. **if** (**current_direction** == '**S**' AND **desired_direction** == '**W**'), then do as following:
        i. **turn_robot**('**R**') *// Turn right 90 degrees*
        ii. **current_direction** = '**W**'
        iii. Go to **Step 15**
    b. **else**, do as following:
        i. Go to **Step 9**
9. **Step 9**:
    a. **if** (**current_direction** == '**E**' AND **desired_direction** == '**N**'), then do as following:
        i. **turn_robot**('**L**') *// Turn left 90 degrees*
        ii. **current_direction** = '**N**'
        iii. Go to **Step 15**
    b. **else**, do as following:
        i. Go to **Step 10**
10. **Step 10**:
    a. **if** (**current_direction** == '**E**' AND **desired_direction** == '**W**'), then do as following:
        i. **if** ((**current_grid** == **1** AND **current_coordinate**[**0**] == **0**)
        OR (**current_grid** == **2** AND **current_coordinate**[**0**] == **0**)
        ), then do as following:
            *// Turn right 180 degrees*
            1. **turn_robot**('**R**') *// Turn right 90 degrees*
            2. **turn_robot**('**R**') *// Turn right 90 degrees*
        ii. **else**, do as following:
            *// Turn left 180 degrees*
            1. **turn_robot**('**L**') *// Turn left 90 degrees*
            2. **turn_robot**('**L**') *// Turn left 90 degrees*
        iii. **current_direction** = '**W**'
        iv. Go to **Step 15**
    b. **else**, do as following:
        i. Go to last **Step 11**
11. **Step 11**:
    a. **if** (**current_direction** == '**E**' AND **desired_direction** == '**S**'), then do as following:
        i. **turn_robot**('**R**') *// Turn right 90 degrees*
        ii. **current_direction** = '**S**'
        iii. Go to **Step 15**
    b. **else**, do as following:
        i. Go to **Step 12**
12. **Step 12**:
    a. **if** (**current_direction** == '**W**' AND **desired_direction** == '**N**'), then do as following:
        i. **turn_robot**('**R**') *// Turn right 90 degrees*
        ii. **current_direction** = '**N**'
        iii. Go to **Step 15**

b. **else**, do as following:
    i. Go to **Step 13**

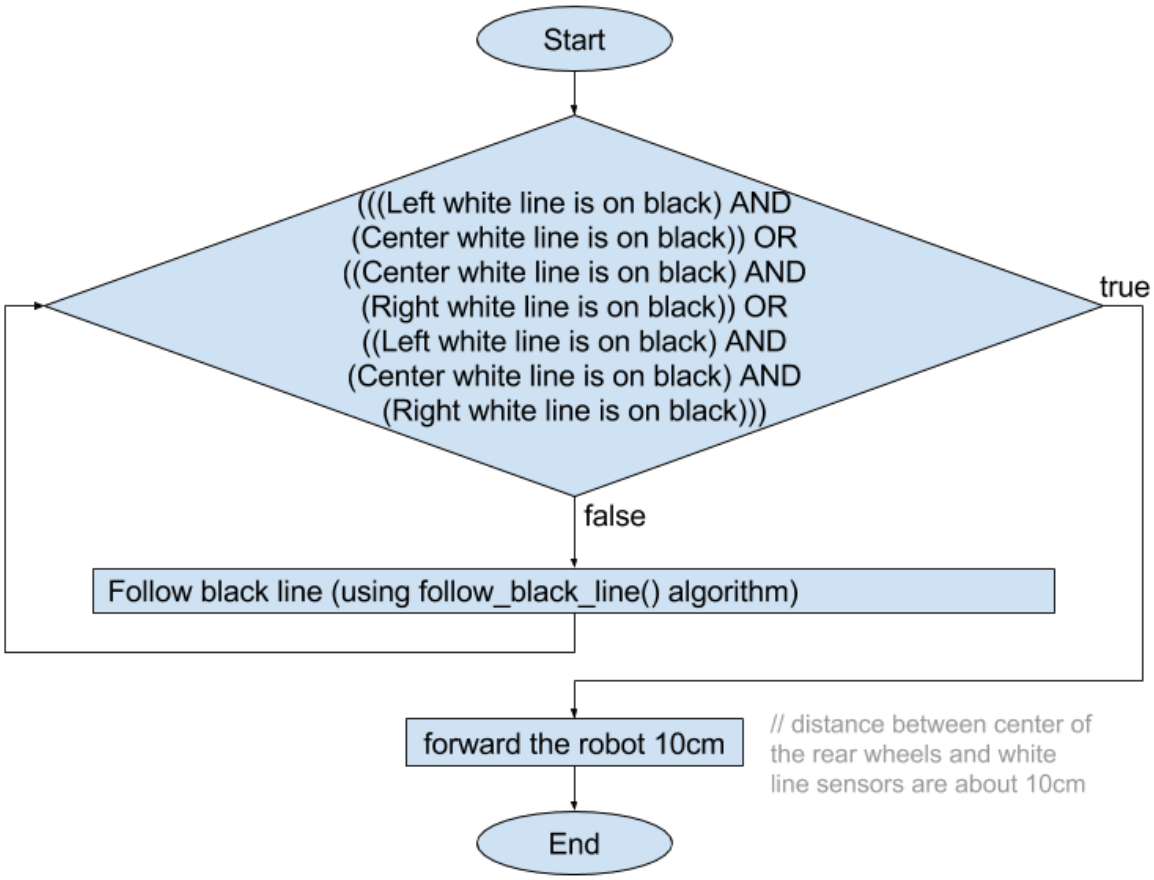
13. **Step 13**:
    a. **if** (**current_direction** == '**W**' AND **desired_direction** == '**E**'), then do as following:
        i. **if** ((**current_grid** == **1** AND **current_coordinate**[**0**] == **0**)
        OR (**current_grid** == **2** AND **current_coordinate**[**0**] == **0**)
        ), then do as following:
            *// Turn left 180 degrees*
            1. **turn_robot**('**L**') *// Turn left 90 degrees*
            2. **turn_robot**('**L**') *// Turn left 90 degrees*
        ii. **else**, do as following:
            *// Turn right 180 degrees*
            1. **turn_robot**('**R**') *// Turn right 90 degrees*
            2. **turn_robot**('**R**') *// Turn right 90 degrees*
        iii. **current_direction** = '**E**'
        iv. Go to **Step 15**
    b. **else**, do as following:
        i. Go to last **Step 14**

14. **Step 14**:
    a. **if** (**current_direction** == '**W**' AND **desired_direction** == '**S**'), then do as following:
        i. **turn_robot**('**L**') *// Turn left 90 degrees*
        ii. **current_direction** = '**S**'
        iii. Go to **Step 15**
    b. **else**, do as following:
        i. Go to **Step 15**

15. **Step 15**:
    a. End of algorithm **change_direction**

- Function **move_one_cell** ()
    - **Details**:
      This function moves the robot by 1 cell i.e. distance of 20 cm dynamically by detecting 3x3 cm black squares.

**Flowchart**: **move_one_cell**

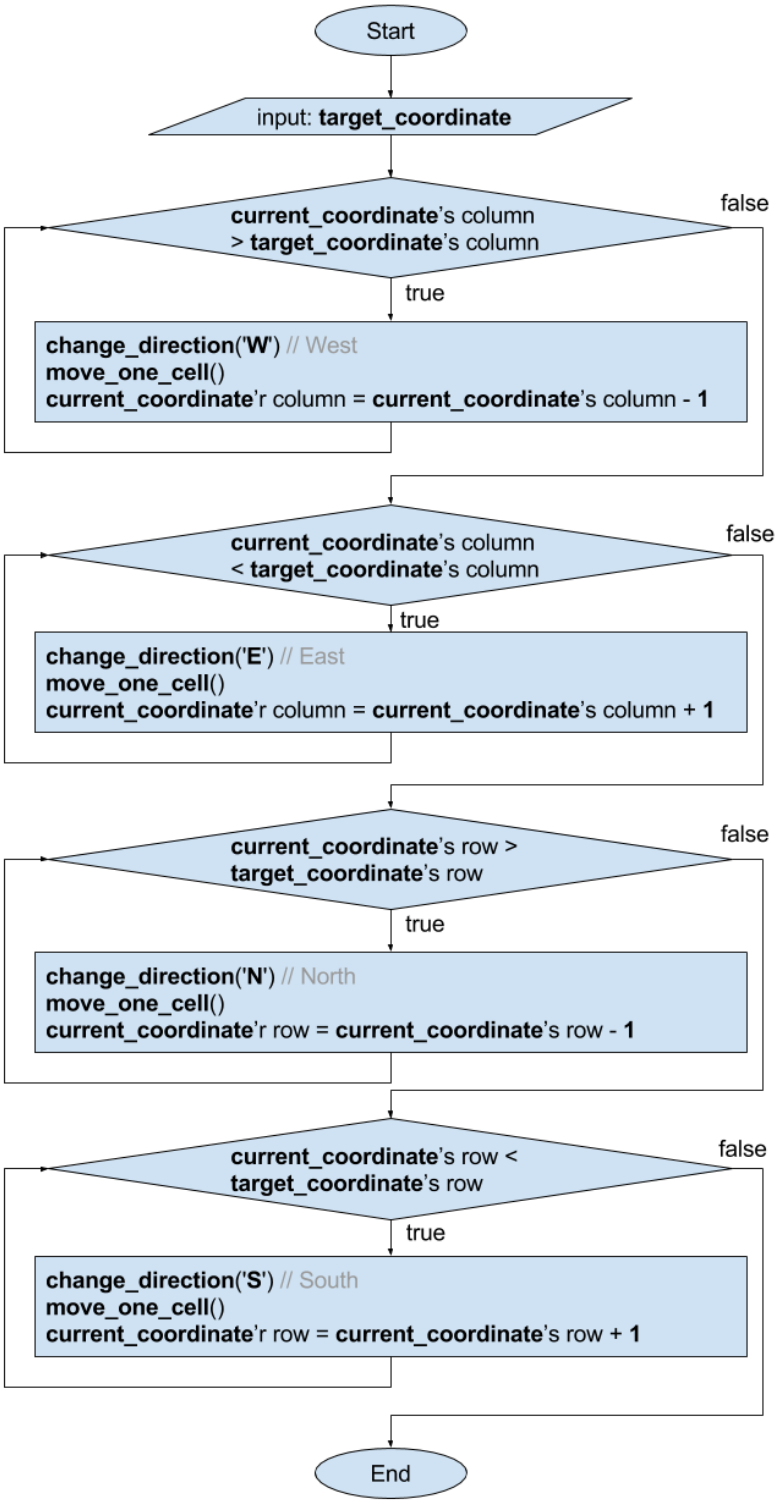



**Algorithm**: **move_one_cell**

1. **Step 1**:
    a. **if** ((left white line sensor is on black AND center white line sensor is on black) OR (center white line sensor is on black AND right white line sensor is on black) OR (left white line sensor is on black AND center white line sensor is on black AND right white line sensor is on black)), then do as following:
        i. follow_black_line(**Left_white_line**, **Center_white_line**, **Right_white_line**)
        *(Left_white_line, Center_white_line, Right_white_line are current readings from three white line sensors)*
        ii. Go to **Step 1**, **a**
    b. **else**, do as following:
        i. forward the robot 10 cm *(10 cm is the distance between center of the rear wheels and white line sensors)*
2. **Step 2**:
    a. End of algorithm **move_one_cell**

- Function **go_to_coordinate** (int array **coordinate**)
    - **Details**:
      **coordinate** is an integer array containing 2 elements i.e. row and column index. This function moves the robot to a given co-ordinate in the **current_grid**.

**Flowchart**: **go_to_coordinate**

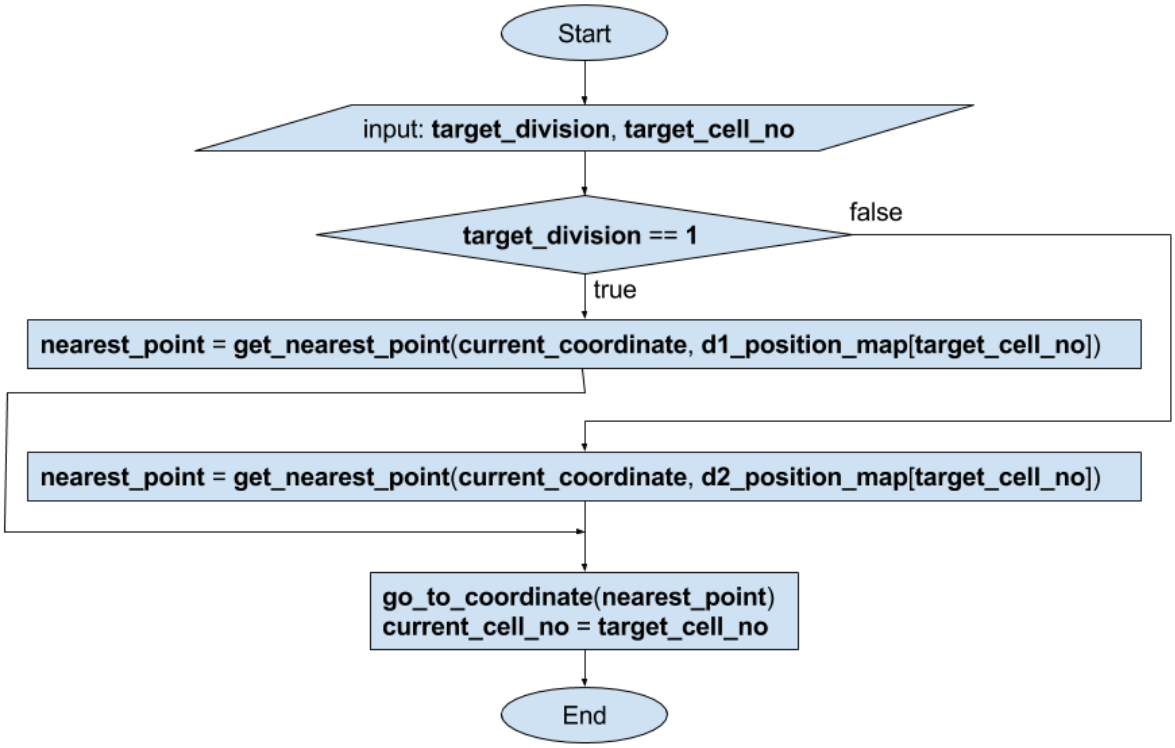


**Algorithm**: go_to_coordinate

1. **Step 1:**
    a. Taking input **target_coordinate**
    *(target_coordinate is an array of two elements i.e. row and column number)*
2. **Step 2:**
    a. **if** (**current_coordinate**'s column > **target_coordinate**'s column), then do as following:
        i. **change_direction**('**W**') *(W = West)*
        ii. **move_one_cell**()
        iii. **current_coordinate**'s column = **current_coordinate**'s column – **1**
        iv. Go to **Step 2**
    b. **else**, do as following:
        i. Go to **Step 3**
3. **Step 3:**
    a. **if** (**current_coordinate**'s column < **target_coordinate**'s column), then do as following:
        i. **change_direction**('**E**') *(E = East)*
        ii. **move_one_cell**()
        iii. **current_coordinate**'s column = **current_coordinate**'s column + **1**
        iv. Go to **Step 3**
    b. **else**, do as following:
        i. Go to **Step 4**
4. **Step 4:**
    a. **if** (**current_coordinate**'s row > **target_coordinate**'s row), then do as following:
        i. **change_direction**('**N**') *(N = North)*
        ii. **move_one_cell**()
        iii. **current_coordinate**'s row = **current_coordinate**'s row – **1**
        iv. Go to **Step 4**
    b. **else**, do as following:
        i. Go to **Step 5**
5. **Step 5:**
    a. **if** (**current_coordinate**'s row < **target_coordinate**'s row), then do as following:
        i. **change_direction**('**S**') *(S = South)*
        ii. **move_one_cell**()
        iii. **current_coordinate**'s row = **current_coordinate**'s row + **1**
        iv. Go to **Step 5**
    b. **else**, do as following:
        i. Go to **Step 6**
6. **Step 6:**
    a. End of algorithm **go_to_coordinate**

- Function **go_to_cell_no** (int target_division, int target_cell_no)
    - **Details**:
      **target_division** and **target_cell_no** are integer numbers. This function moves the robot to target cell's nearest point.

**Flowchart**: **go_to_cell_no**

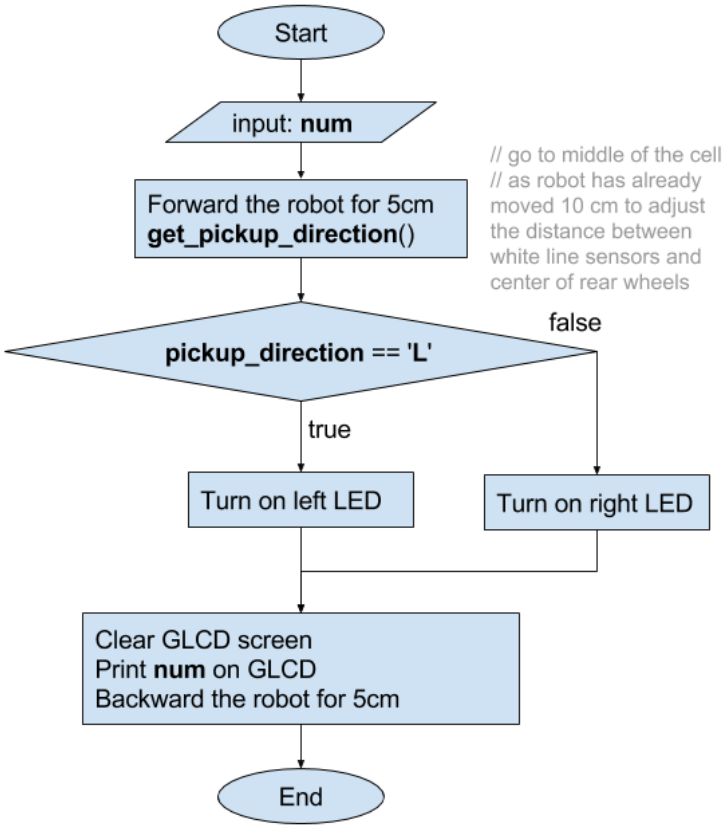



**Algorithm**: **go_to_cell_no**

1. **Step 1**:
    a. Taking input **target_division** and **target_cell_no**
2. **Step 2**:
    a. **if** (**target_division** == **1**), then do as following:
        i. **nearest_point** = **get_nearest_point**(**current_coordinate**, **d1_position_map**[**target_cell_no**])
    b. **else**, do as following:
        i. **nearest_point** = **get_nearest_point**(**current_coordinate**, **d2_position_map**[**target_cell_no**])
3. **Step 3**:
    a. **go_to_coordinate**(**nearest_point**)
    b. **current_cell_no** = **target_cell_no**
4. **Step 4**:
    a. End of algorithm **go_to_cell_no**

- Function **pickup** (int **num**)
    - **Details**:
      **num** is an integer numbers. This function picks up numbers i.e. operands from D1.

**Flowchart**: **pickup**

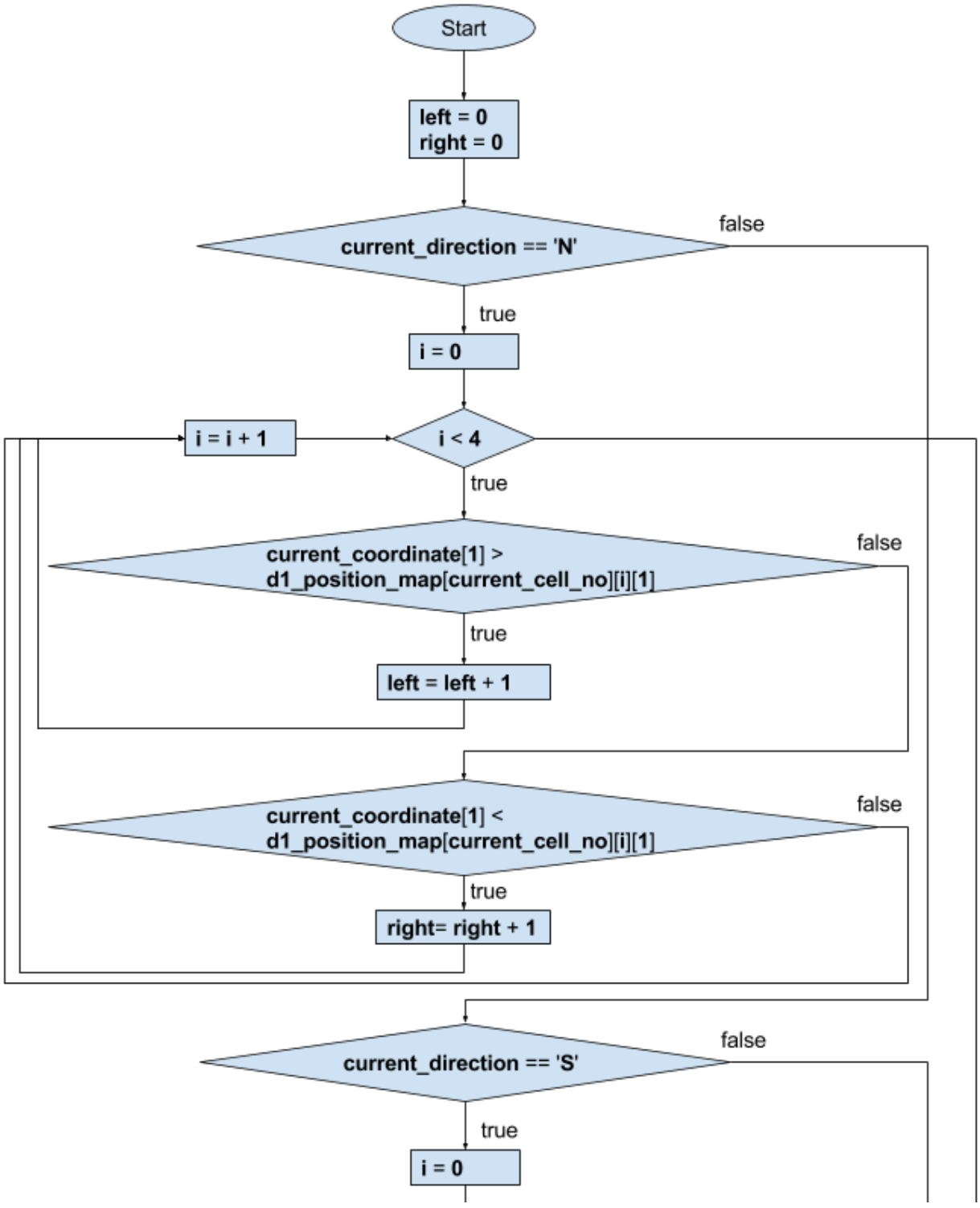


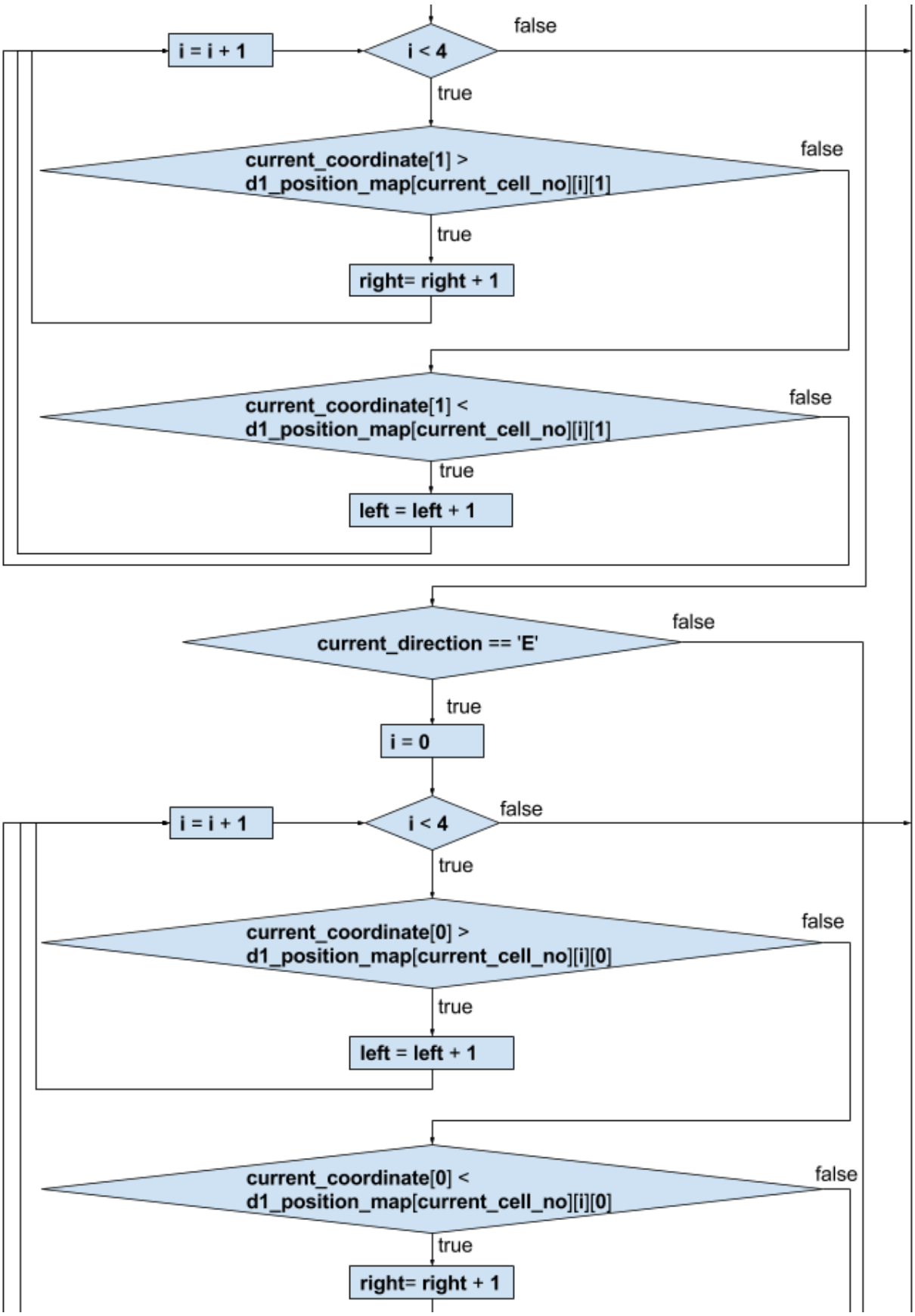

**Algorithm**: **pickup**

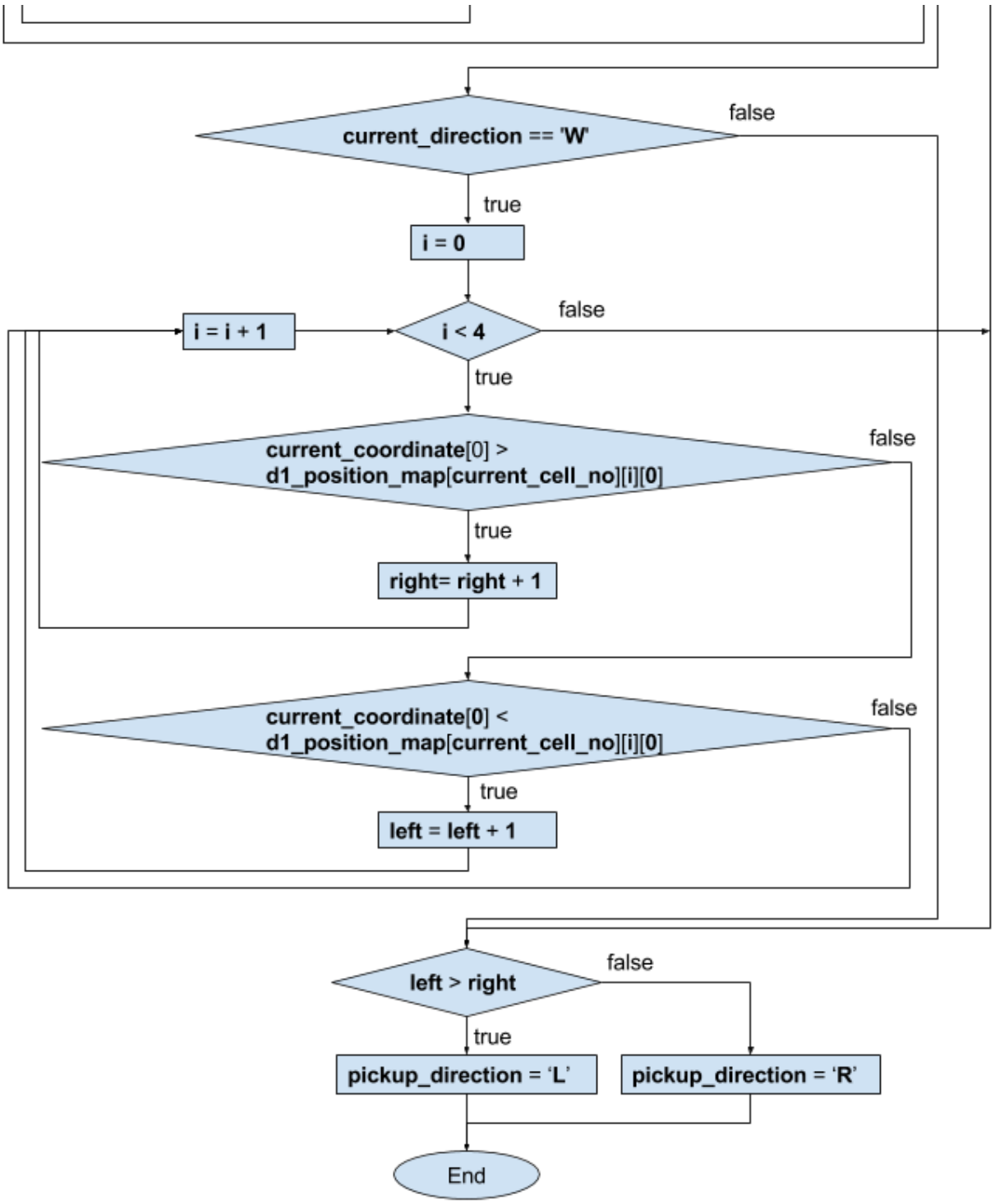
1. **Step 1**:
    a. Input **num** *(num is the value of the number that is to be picked)*
    b. Forward the robot for 5 cm (20/2 – 10/2)
    *(as robot has already moved 10 cm to adjust the distance between white line sensors and center of rear wheels)*
    c. **get_pickup_direction**()
2. **Step 2**:
    a. **if** (**pickup_direction** == '**L**'), then do as following:
        i. Turn on left LED
    b. **else**, do as following:
        i. Turn on right LED
3. **Step 3**:
    a. Clear GLCD screen
    b. Print **num** on GLCD
    c. Backward the robot for 5 cm
4. **Step 4**:
    a. End of algorithm **pickup**

- Function **get_pickup_direction** ()
    - **Details**:
      **direction** is a integer numbers. This function calculates pickup direction and updates global variable **pickup_direction** accordingly.

**Flowchart**: **get_pickup_direction**




i = i + 1
i < 4
false
true
current_coordinate[1] >
d1_position_map[current_cell_no][i][1]
false
true
right= right + 1
current_coordinate[1] <
d1_position_map[current_cell_no][i][1]
false
true
left = left + 1
current_direction == 'E'
false
true
i = 0
i = i + 1
i < 4
false
true
current_coordinate[0] >
d1_position_map[current_cell_no][i][0]
false
true
left = left + 1
current_coordinate[0] <
d1_position_map[current_cell_no][i][0]
false
true
right= right + 1


current_direction == 'W'
false
true
i = 0
i = i + 1
i < 4
false
true
current_coordinate[0] >
d1_position_map[current_cell_no][i][0]
false
true
right= right + 1
current_coordinate[0] <
d1_position_map[current_cell_no][i][0]
false
true
left = left + 1
left > right
false
true
pickup_direction = 'L'
pickup_direction = 'R'
End

**Algorithm**: get_pickup_direction

1. **Step 1**:
    a. Initialize **left** = **0**, **right** = **0**
2. **Step 2**:
    a. **if** (**current_direction** == '**N**'), then do as following:
        i. **i** = **0**
        ii. **if** (**i** < **4**), then do as following:
            1. **if** (**curren_coordinate**[**1**] > **d1_position_map**[**current_cell_no**][**i**][**1**]), then do as following:
                a. **left = left + 1**
            2. **else if** (**curren_coordinate**[**1**] < **d1_position_map**[**current_cell_no**][**i**][**1**]), then do as following:
                a. **right** = **right** + **1**
            3. **i** = **i** + **1**
            4. Go to **Step 2**, **a**. **ii**.
        iii. **else**, do as following:
            1. Go to **Step 6**
    b. **else**, do as following:
        i. Go to **Step 3**
3. **Step 3**:
    a. **else if** (**current_direction** == '**S**'), then do as following:
        i. **i** = **0**
        ii. **if** (**i** < **4**), then do as following:
            1. **if** (**curren_coordinate**[**1**] > **d1_position_map**[**current_cell_no**][**i**][**1**]), then do as following:
                a. **right** = **right** + **1**
            2. **else if** (**curren_coordinate**[**1**] < **d1_position_map**[**current_cell_no**][**i**][**1**]), then do as following:
                a. **left = left + 1**
            3. **i** = **i** + **1**
            4. Go to **Step 3**, **a**. **ii**.
        iii. **else**, do as following:
            1. Go to **Step 6**
    b. **else**, do as following:
        i. Go to **Step 4**
4. **Step 4**:
    a. **else if** (**current_direction** == '**E**'), then do as following:
        i. **i** = **0**
        ii. **if** (**i** < **4**), then do as following:
            1. **if** (**curren_coordinate**[**0**] > **d1_position_map**[**current_cell_no**][**i**][**0**]), then do as following:
                a. **left = left + 1**
            2. **else if** (**curren_coordinate**[**0**] < **d1_position_map**[**current_cell_no**][**i**][**0**]), then do as following:

a. **right** = **right** + **1**

3. **i** = **i** + **1**
4. Go to **Step 4**, **a**. **ii**.

iii. **else**, do as following:

1. Go to **Step 6**

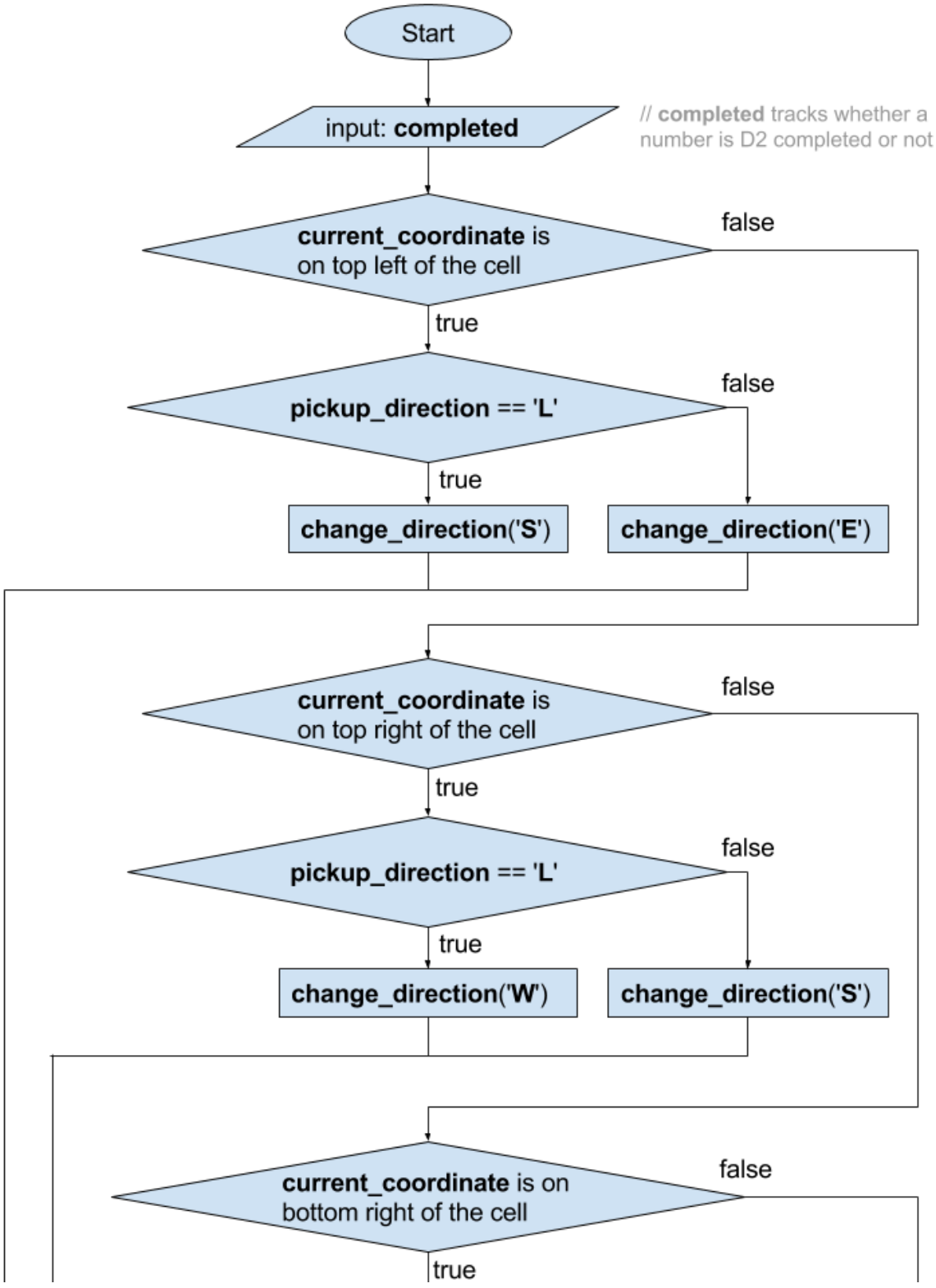
b. **else**, do as following:

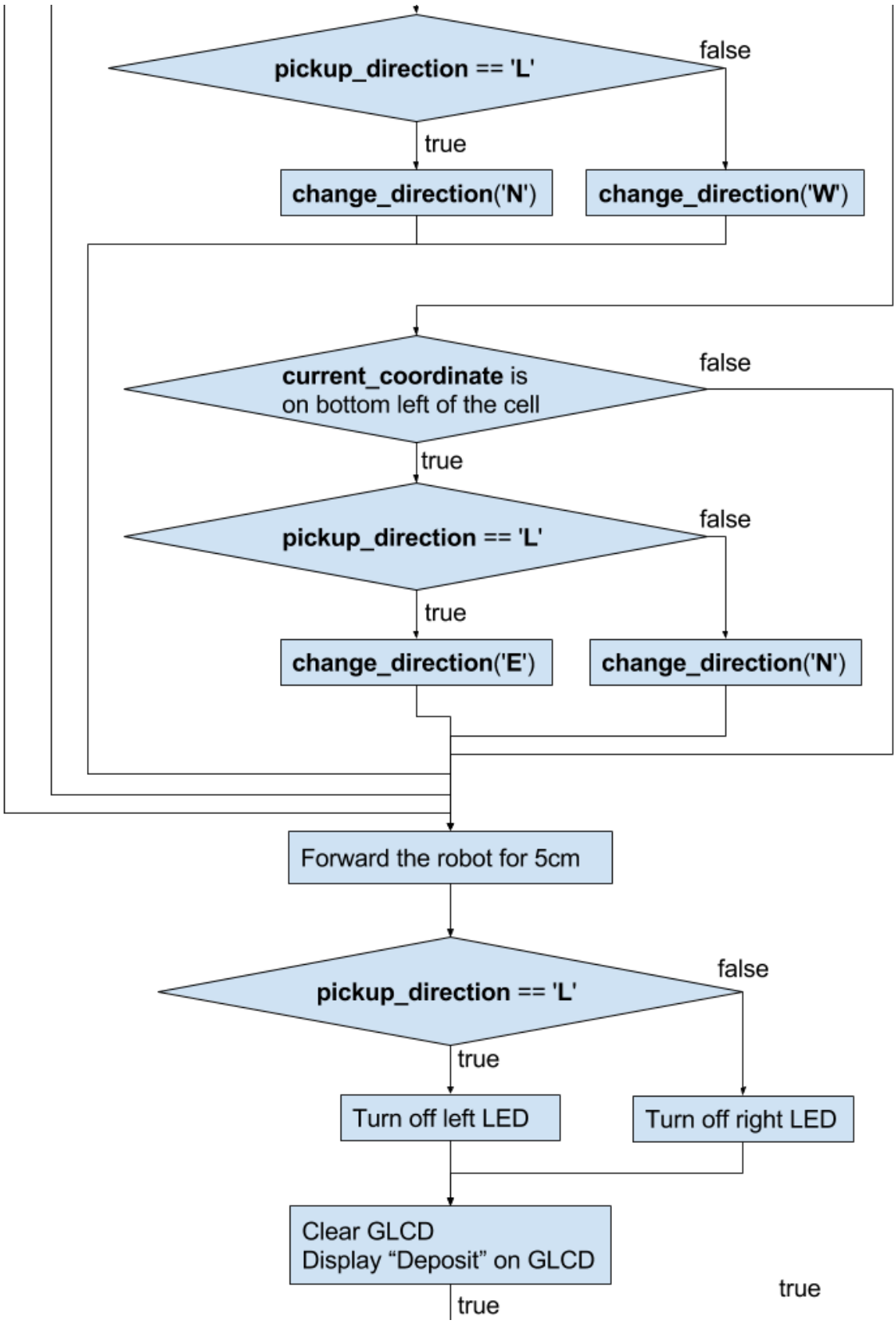
i. Go to **Step 5**

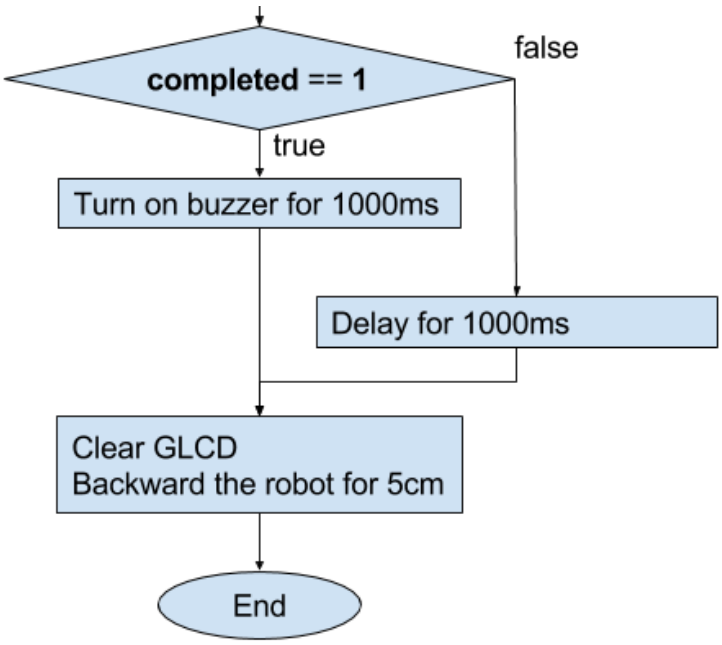
5. **Step 5**:
   a. **else if** (**current_direction** == '**W**'), then do as following:
      i. **i** = **0**
      ii. **if** (**i** < **4**), then do as following:
         1. **if** (**curren_coordinate**[**0**] > **d1_position_map**[**current_cell_no**][**i**][**0**]), then do as following:
            a. **right** = **right** + **1**
         2. **else if** (**curren_coordinate**[**0**] < **d1_position_map**[**current_cell_no**][**i**][**0**]), then do as following:
            a. **left = left** + **1**
         3. **i** = **i** + **1**
         4. Go to **Step 5**, **a**. **ii**.
      iii. **else**, do as following:
         1. Go to **Step 6**
   b. **else**, do as following:
      i. Go to **Step 6**
6. **Step 6**:
   a. **if** (**left** > **right**), then do as following:
      i. **pickup_direction = 'L'**
   b. **else**, do as following:
      i. **pickup_direction** = '**R**'
7. **Step 7**:
   a. End of algorithm **get_pickup_direction**

- Function **deposit** (int completed)
    - **Details**:
      **direction** is a integer numbers. This function deposits operands to D2.

**Flowchart**: **deposit**




pickup_direction == 'L'
false
true
change_direction('N')
change_direction('W')
current_coordinate is on bottom left of the cell
false
true
pickup_direction == 'L'
false
true
change_direction('E')
change_direction('N')
Forward the robot for 5cm
pickup_direction == 'L'
false
true
Turn off left LED
Turn off right LED
Clear GLCD
Display “Deposit” on GLCD
true
true




**Algorithm**: **deposit**

1. **Step 1**:
    a. Input **completed**
    *(completed tracks whether a number in D2 completed or not)*
2. **Step 2**:
    a. **if** (**current_coordinate** is on top left of the cell), then do as following:
        i. **if** (**pickup_direction** == '**L**'), then do as following:
            1. **change_direction**('**S**')
        ii. **else**, do as following:
            1. **change_direction**('**E**')
        iii. Go to **Step 6**
    b. **else**, do as following:
        i. Go to **Step 3**
3. **Step 3**:
    a. **if** (**current_coordinate** is on top right of the cell), then do as following:
        i. **if** (**pickup_direction** == '**L**'), then do as following:
            1. **change_direction**('**W**')
        ii. **else**, do as following:
            1. **change_direction**('**S**')
        iii. Go to **Step 6**
    b. **else**, do as following:
        i. Go to **Step 4**
4. **Step 4**:
    a. **if** (**current_coordinate** is on bottom right of the cell), then do as following:
        i. **if** (**pickup_direction** == '**L**'), then do as following:
            1. **change_direction**('**N**')
        ii. **else**, do as following:
            1. **change_direction**('**W**')
        iii. Go to **Step 6**

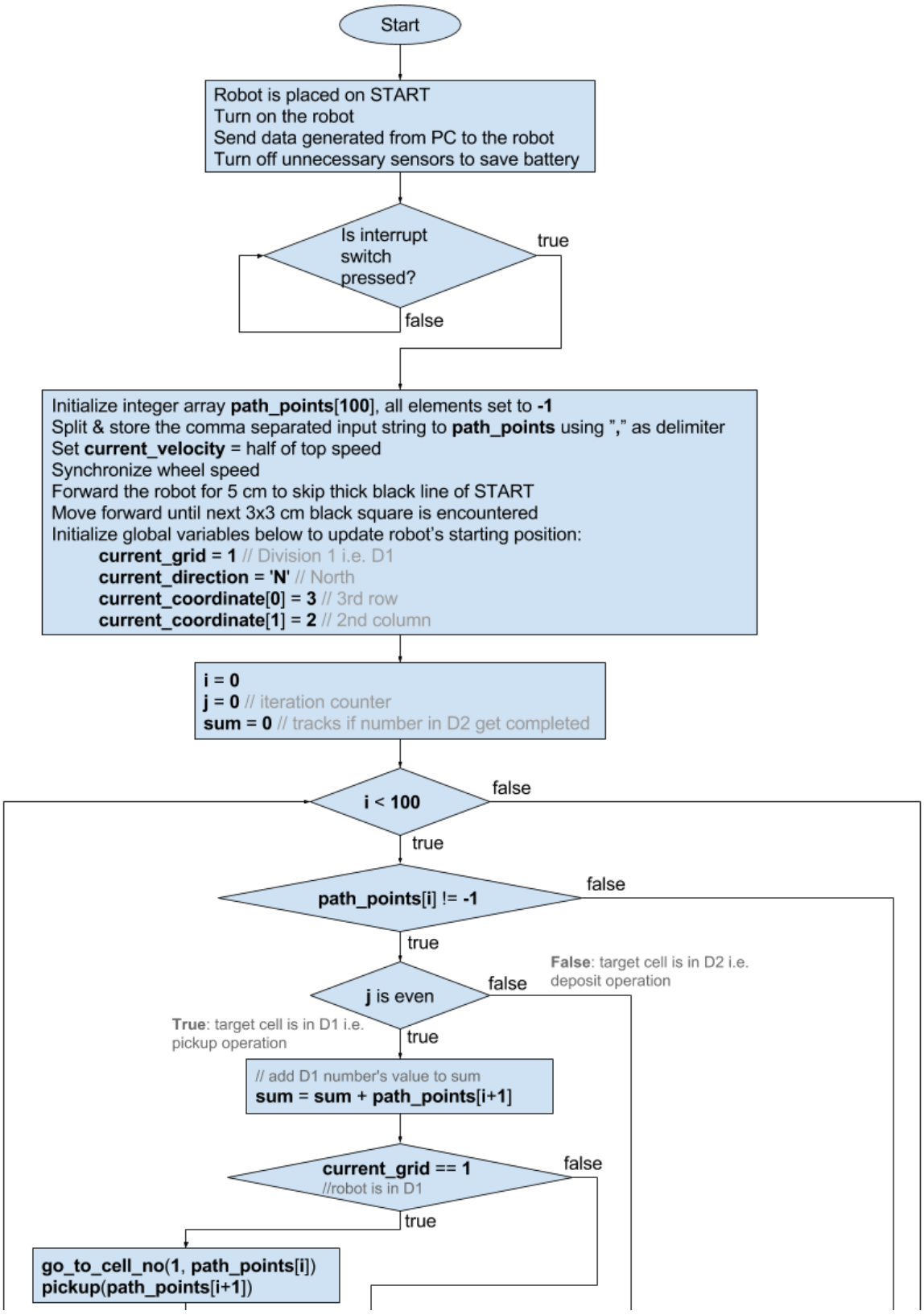
b. **else**, do as following:
    i. Go to **Step 5**

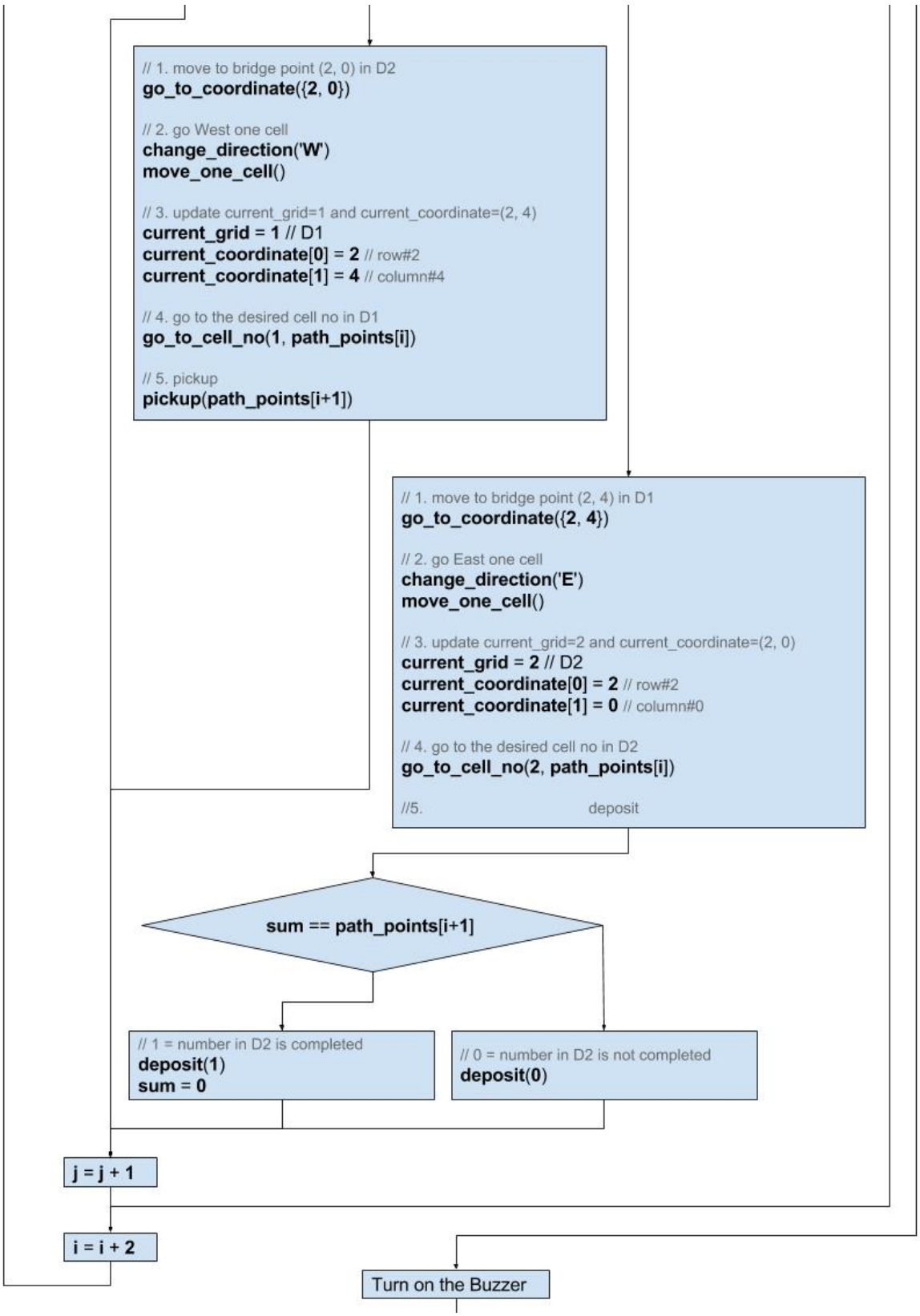
5. **Step 5**:
    a. **if** (**current_coordinate** is on bottom left of the cell), then do as following:
        i. **if** (**pickup_direction == 'L'**), then do as following:
            1. **change_direction**('**E**')
        ii. **else**, do as following:
            1. **change_direction**('**N**')
        iii. Go to **Step 6**
    b. **else**, do as following:
        i. Go to **Step 6**
6. **Step 6**:
    a. Forward the robot for 5 cm
    b. **if** (**pickup_direction == 'L'**), then do as following:
        i. Turn off left LED
    c. **else**, do as following:
        i. Turn off right LED
    d. Clear GLCD
    e. Display message "Deposit" on GLCD
    f. **if** (**completed** == **1**), then do as following:
        i. Turn on buzzer for 1000 ms
    g. **else**, do as following:
        i. Delay for 1000 ms
    h. Clear GLCD
    i. Backward the robot for 5 cm
7. **Step 7**:
    a. End of algorithm **deposit**

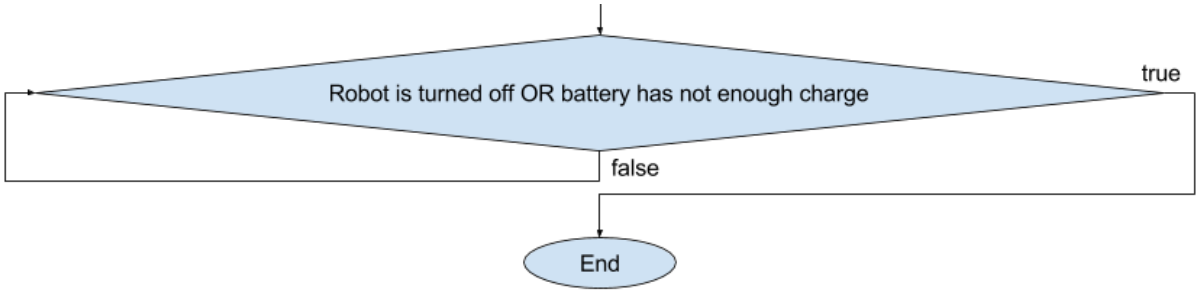
**Q-2 Draw a flowchart illustrating main function of your code.**

**Answer**:

- Flowchart for the **main** function:




// 1. move to bridge point (2, 0) in D2
go_to_coordinate({2, 0})
// 2. go West one cell
change_direction('W')
move_one_cell()
// 3. update current_grid=1 and current_coordinate=(2, 4)
current_grid = 1 // D1
current_coordinate[0] = 2 // row#2
current_coordinate[1] = 4 // column#4
// 4. go to the desired cell no in D1
go_to_cell_no(1, path_points[i])
// 5. pickup
pickup(path_points[i+1])
// 1. move to bridge point (2, 4) in D1
go_to_coordinate({2, 4})
// 2. go East one cell
change_direction('E')
move_one_cell()
// 3. update current_grid=2 and current_coordinate=(2, 0)
current_grid = 2 // D2
current_coordinate[0] = 2 // row#2
current_coordinate[1] = 0 // column#0
// 4. go to the desired cell no in D2
go_to_cell_no(2, path_points[i])
//5. deposit
sum == path_points[i+1]
// 1 = number in D2 is completed
deposit(1)
sum = 0
// 0 = number in D2 is not completed
deposit(0)
j = j + 1
i = i + 2
Turn on the Buzzer




- Algorithm for the **main** function:
  1. **Step 1**:
     a. Robot is placed on START
     b. Turn on the robot
     c. Send data generated from PC to the robot
     d. Turn off unnecessary sensors to save battery (like sharp IR, proximity sensors)
  2. **Step 2**:
     a. **if** (interrupt switch is pressed), then do as following:
        i. Go to **Step 3**
     b. **else**, do as following:
        i. Go to **Step 2**, **a.**
  3. **Step 3**:
     a. Initialize integer array **path_points**[**100**], all elements are set to **-1**
     b. Split and store comma separated input string to **path_points** using "," as delimeter
     c. Set **current_velocity** = half of top speed i.e. **127**
     d. Synchronize wheel speed
     e. Forward the robot for 5 cm to skip thick black line of START
     f. Move forward until next 3x3 cm black square is encountered
     g. Initialize global variables below to update robot's starting position:
        i. **current_grid** = **1** *(Division 1 i.e. D1)*
        ii. **current_direction** = '**N**' *(North)*
        iii. **current_coordinate**[**0**] = **3** *(3rd row)*
        iv. **current_coordinate**[**1**] = **2** *(2nd column)*
  4. **Step 4**:
     *a.* Initialize i = 0, j = 0 *(iteration counter)*
     b. Initialize sum = 0 *(tracks if number in D2 get completed)*
     c. **if** (**i** < **100**), then do as following:
        i. **if** (**path_points**[**i**] **!= -1**), then do as following:
           1. **if** (**j** is even), then do as following:
              *(True: target cell is in D1 i.e. pickup operation)*
              a. **sum = sum** + **path_points**[**i** + **1**] *(add D1 number's value to sum)*
              b. **if** (**current_grid == 1**), then do as following:
                 *(robot is in D1)*
                 i. **go_to_cell_no**(**1**, **path_points**[**i**])
                 ii. **pickup**(**path_points**[**i** + **1**])
              c. **else**, do as following:
                 *(1. move to bridge point (2, 0) in D2)*

i. **go_to_coordinate**({**2**, **0**})

*(2. go West one cell)*

ii. **change_direction**('**W**')

iii. **move_one_cell**()

*(3. update current_grid=1 & current_coordinate=(2, 4))*

iv. **current_grid** = **1** *(D1)*

v. **current_coordinate**[**0**] = **2** *(row#2)*

vi. **current_coordinate**[**1**] = **4** *(column#4)*

*(4. go to the desired cell no in D1)*

vii. **go_to_cell_no**(**1**, **path_points**[**i**])

*(5. pickup)*

viii. **pickup**(**path_points**[**i** +**1**])

d.

2. **else**, do as following:

*(False: target cell is in D2 i.e. deposit operation)*

*(1. move to bridge point (2, 4) in D1)*

a. **go_to_coordinate**({**2**, **4**})

*(2. go East one cell)*

b. **change_direction**('**E**')

c. **move_one_cell**()

*(3. update current_grid=2 & current_coordinate=(2, 0))*

d. **current_grid** = **2** *(D2)*

e. **current_coordinate**[**0**] = **2** *(row#2)*

f. **current_coordinate**[**1**] = **0** *(column#0)*

*(4. go to the desired cell no in D2)*

g. **go_to_cell_no**(**2**, **path_points**[**i**])

*(5. deposit)*

i. **if** (**sum** == **path_points**[**i** + **1**]), then do as following:
(1=number in D2 is completed)
    1. deposit(1)
    2. sum = 0

ii. **else**, do as following:
(0=number in D2 is not completed)
    1. deposit(0)

3. **j** = **j** + **1**

4. Go to **Step 4**, **c**, **iii.**

ii. **else**, do as following:
    1. Go to **Step 4**, **c**, **iii.**

iii. **i = i** + **2**

iv. Go to **Step 4, c.**

    d. **else**, do as following:
        i. Go to **Step 5**
5. **Step 5**:
    a. Turn on the buzzer
    b. **if** (Robot is turned off OR battery has not enough charge), then do as following:
        i. Go to **Step 6**
6. **Step 6**:
    a. End of algorithm **main**

## Challenges (10)

**What are the major challenges that you can anticipate in addressing this theme?**

**Answer:**

1. **Challenge 1**: Generate robot's traversal path, format the data, and send it to the robot over USB to Serial cable.
2. **Challenge 2**: Follow the 1cm thick black line with 3 white line sensors as the gap between two white line sensors is more than 1cm.
3. **Challenge 3**: Detect 3x3cm black squares and following 1cm black line until detecting the next 3x3cm black square.
4. **Challenge 4**: Mapping grids cell co-ordinates by row columns and updating robot's current position based on current row, column, division and direction.
5. **Challenge 5**: Synchronize two wheels and rotate the robot by 90 degrees perfectly.
6. **Challenge 6**: Finding the shortest path and get nearest co-ordinate of a cell (each cell has four co-ordinates i.e. corner points) to reach from current co-ordinate by calculating costs of each path.
7. **Challenge 7**: Get the pickup direction (left/right).
8. **Challenge 8**: Cross the bridge between two divisions i.e. D1 and D2.
9. **Challenge 9**: Rotate to robot in such direction that deposit side matches the pickup side.
10. **Challenge 10**: Debug some special exception cases.