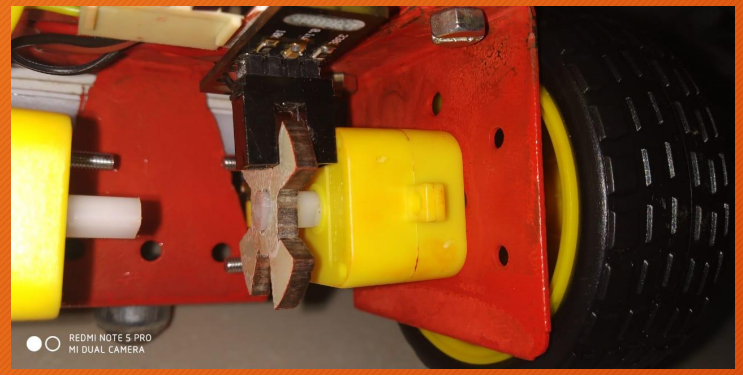
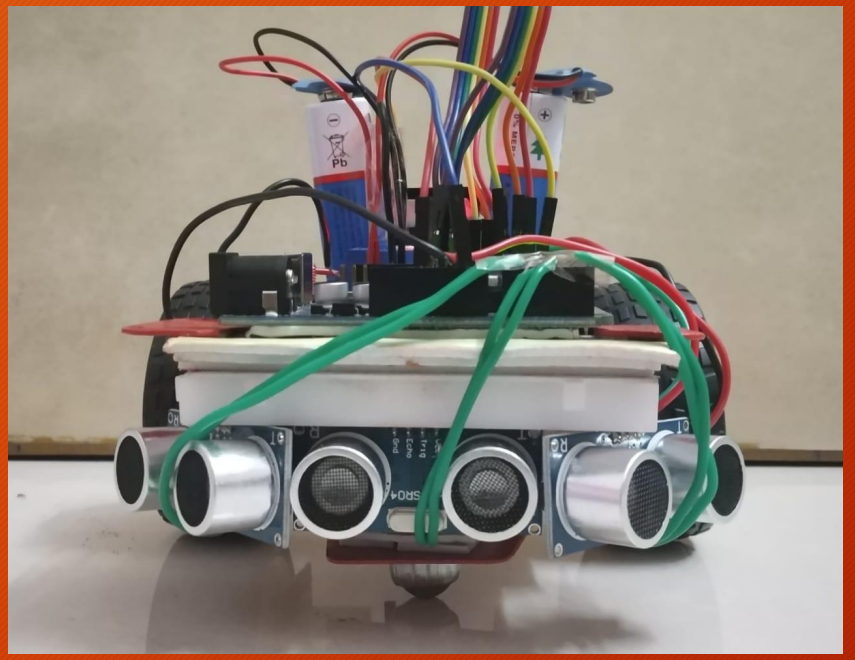
MICRO MOUSE MAZE

MICRO-MOUSE MAZE SOLVER ROBOT



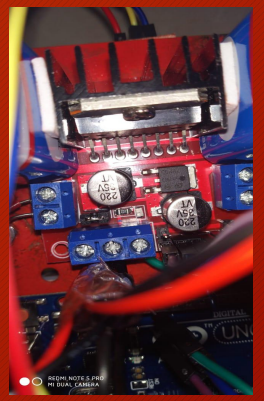
Opto-coupler Encoder

Ultrasonic Sensor



Micro-Mouse Maze Solver Robot

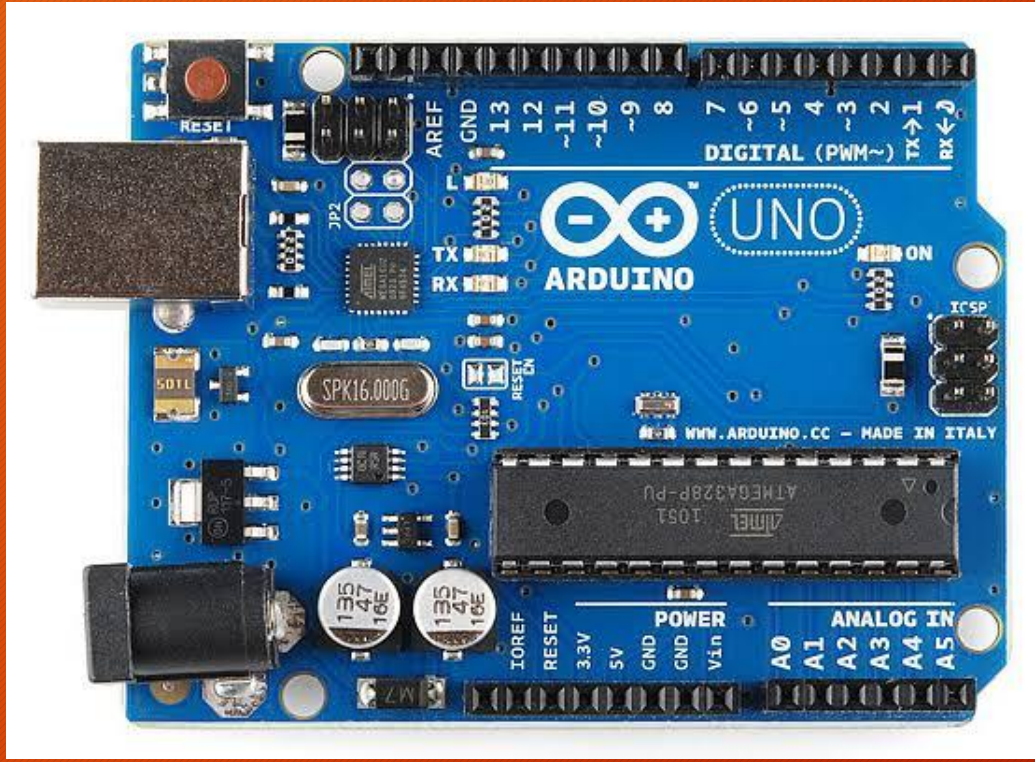
Arduino UNO



Motor Driver

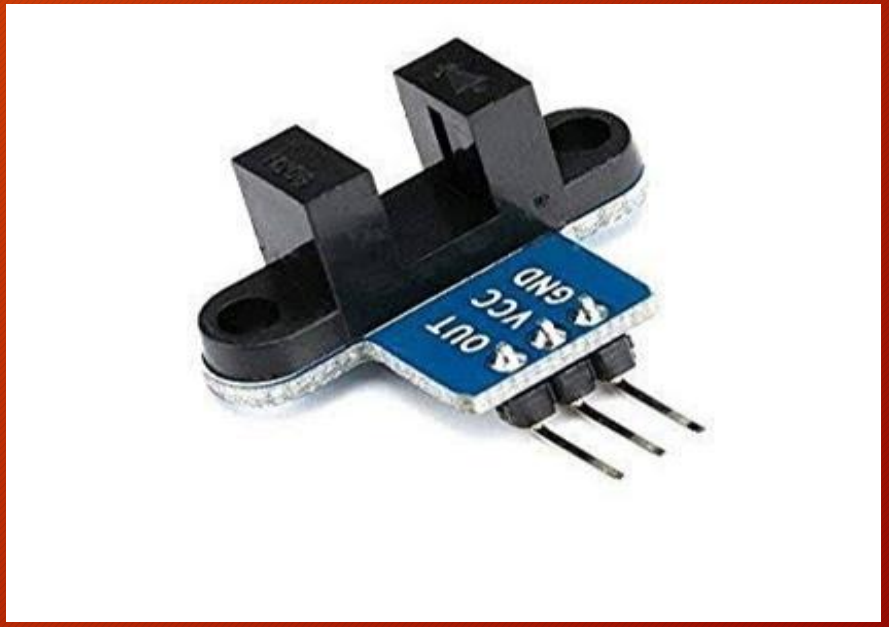
COMPONENTS USED:

ARDUINO UNO


RESET
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
JP2
TX
RX
ARDUINO
UNO
ON
ICSP
SPK16.000G
RESET
WWW.ARDUINO.CC – MADE IN ITALY
ATMEGA328P-PU
IOREF
RESET
3.3V
5V
GND
GND
Vin
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN

OPTO COUPLER ENCODER – LM393

A current is first applied to the Optocoupler, which makes the infrared LED emit a light that’s proportional to the current. When the light hits the photosensitive device, it switches on and starts to conduct a current as any ordinary transistor might. This is used in encoder.

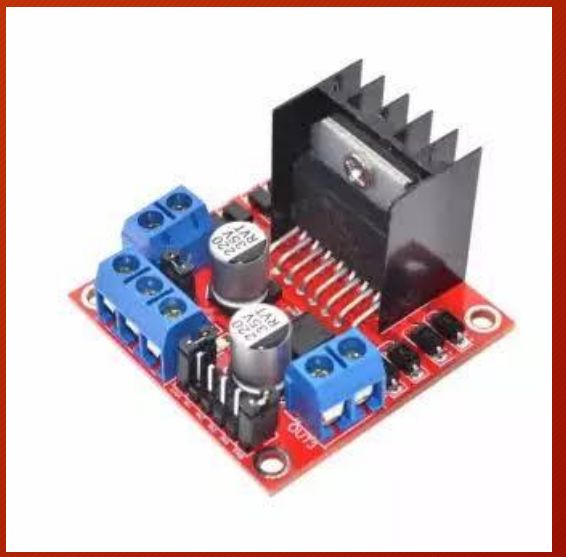


ULTRASONIC SENSOR – HC-SR04

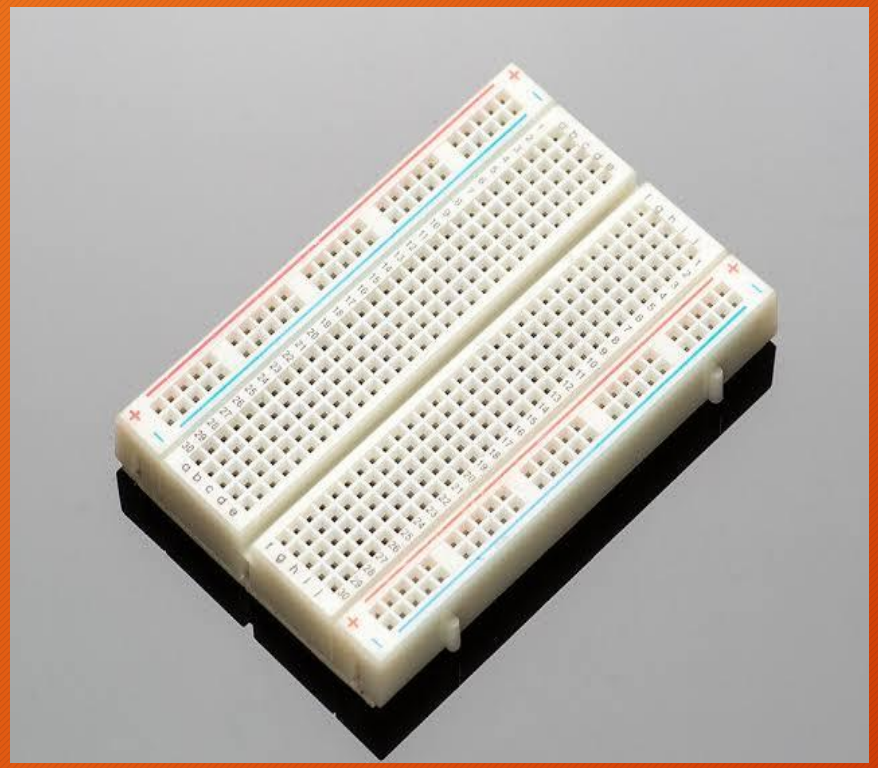
Ultrasonic sensors work by emitting sound waves at a frequency too high for humans to hear. They then wait for the sound to be reflected back, calculating distance based on the time required. This is similar to how radar measures the time it takes a radio wave to return after hitting an object. This is used to measure the distance travelled by the robot.

MOTOR DRIVER – L298N

A motor driver IC is an integrated circuit chip which is usually used to control motors in autonomous robots. Motor driver ICs act as an interface between microprocessors in robots and the motors in the robot. The most commonly used motor driver IC's are from the L293 series such as L293D, L293NE, etc. These ICs are designed to control 2 DC motors simultaneously. L293D consist of two H-bridge. H-bridge is the simplest circuit for controlling a low current rated motor. Motor Driver ICs are primarily used in autonomous robotics only. Also most microprocessors operate at low voltages and require a small amount of current to operate while the motors require a relatively higher voltages and current . Thus current cannot be supplied to the motors from the microprocessor. This is the primary need for the motor driver IC.



BATTERY, WIRES AND BREADBOARD



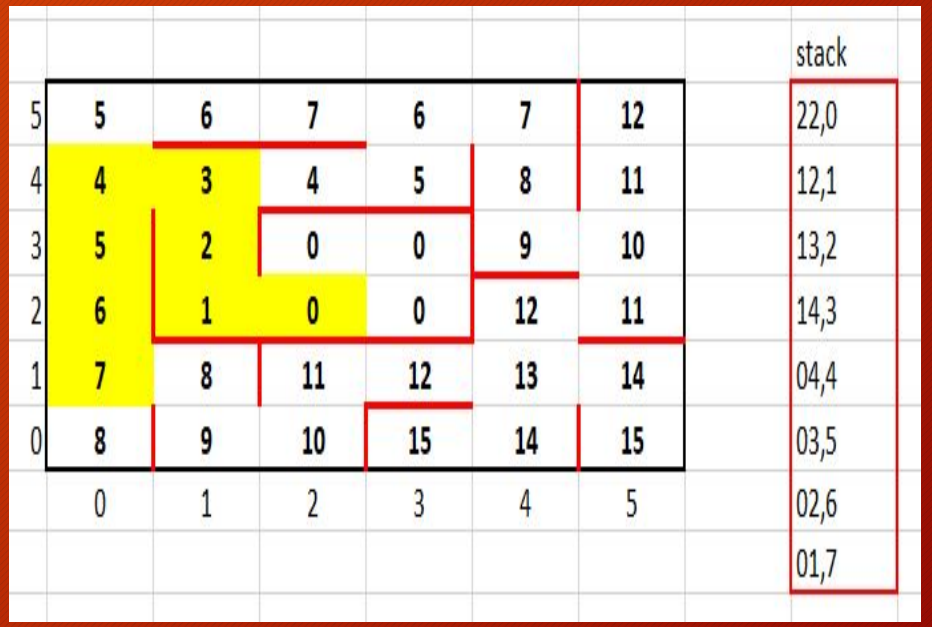
CHASSIS, WHEELS, CASTER WHEEL AND DC-MOTORS

MAZE SOLVING ALGORITHM

-Modified Flood Fill Algorithm

ALGORITHM DESCRIPTION:

- The Flood Fill algorithm works by assigning value for all nodes in the maze, where these values indicate the steps from any cell to the destination node.
- Once the micro-mouse reaches a particular node it updates its flood value to 1 + minimum flood value of the open neighbour node, propagates the flood values and moves to the node which has the smallest flood value.
- The Modified Flood Fill Algorithm is the further extension of the Flood Fill algorithm where the flood value of the present node is updated only if required to do so else continues its motion in a greater pace.
- Here in our Algorithm, after propagating the flood values to all the adjacent neighbouring nodes, the flood value of the present node is checked. Not every node travelled by the micro-mouse robot but only the nodes which is 1 less than the flood value of the previously stored node's flood value, is pushed to the dynamically allocated stack for keeping track of the preferred path to destination cell.
- In this way only one final exact path to the destination node will be stored which solves the algorithm to solve the entire maze by the micro-mouse maze efficiently.



							stack
5	5	6	7	6	7	12	22,0
4	4	3	4	5	8	11	12,1
3	5	2	0	0	9	10	13,2
2	6	1	0	0	12	11	14,3
1	7	8	11	12	13	14	04,4
0	8	9	10	15	14	15	03,5
	0	1	2	3	4	5	02,6
							01,7

Propagation And Storing Of Flood values:

5	5	6	7	6	7	12
4	4	3	4	5	8	11
3	5	2	0	0	9	10
2	6	1	0	0	12	11
1	7	8	11	12	13	14
0	**8**	9	10	15	14	15
	0	1	2	3	4	5

5	4	3	2	2	3	4
4	3	2	1	1	2	3
3	2	1	0	0	1	2
2	2	1	0	0	1	2
1	3	2	1	1	2	3
0	4	3	2	2	3	4
	0	1	2	3	4	5

5	4	3	2	2	3	4
4	3	2	1	1	2	3
3	2	1	0	0	1	2
2	3	1	0	0	1	2
1	3	2	1	1	2	3
0	4	3	2	2	3	4
	0	1	2	3	4	5

5	4	3	2	2	3	4
4	3	2	1	1	2	3
3	4	1	0	0	1	2
2	4	1	0	0	1	2
1	3	2	1	1	2	3
0	4	3	2	2	3	4
	0	1	2	3	4	5

5	4	3	2	2	3	4
4	3	2	1	1	2	3
3	4	1	0	0	1	2
2	4	1	0	0	1	2
1	3	2	1	1	2	3
0	4	3	2	2	3	4
	0	1	2	3	4	5

5	4	3	2	2	3	4
4	3	2	1	1	2	3
3	4	1	0	0	1	2
2	4	1	0	0	1	2
1	3	2	1	1	2	3
0	4	3	2	2	3	4
	0	1	2	3	4	5

5	4	4	3	2	3	4
4	3	2	1	1	2	3
3	4	1	0	0	1	2
2	4	1	0	0	1	2
1	3	2	1	1	2	3
0	4	3	2	2	3	4
	0	1	2	3	4	5

5	4	4	3	2	3	4
4	3	2	1	1	2	3
3	4	1	0	0	1	2
2	4	1	0	0	1	2
1	3	2	1	1	2	3
0	4	3	2	2	3	4
	0	1	2	3	4	5

5	4	4	3	2	3	4
4	3	2	1	1	2	3
3	4	1	0	0	1	2
2	4	1	0	0	1	2
1	3	2	1	1	2	3
0	4	3	2	2	3	4
	0	1	2	3	4	5

5	4	4	3	2	3	5
4	3	2	1	1	4	4
3	4	1	0	0	4	3
2	4	1	0	0	1	2
1	3	2	1	1	2	3
0	4	3	2	2	3	4
	0	1	2	3	4	5

5	4	4	3	2	3	5
4	3	2	1	1	4	4
3	4	1	0	0	4	3
2	4	1	0	0	1	2
1	3	2	1	1	2	3
0	4	3	2	2	3	4
	0	1	2	3	4	5

5	4	4	3	2	3	7
4	3	2	1	1	4	6
3	4	1	0	0	5	5
2	4	1	0	0	3	4
1	3	2	1	1	2	3
0	4	3	2	2	3	4
	0	1	2	3	4	5

stack
01,3

5	4	4	3	2	3	5
4	3	2	1	1	4	4
3	4	1	0	0	4	3
2	4	1	0	0	1	2
1	3	2	1	1	2	3
0	4	3	2	2	3	4
	0	1	2	3	4	5

5	4	4	3	2	3	5
4	3	2	1	1	4	4
3	4	1	0	0	4	3
2	4	1	0	0	1	2
1	3	2	1	1	2	3
0	4	3	2	2	3	4
	0	1	2	3	4	5

5	4	4	3	2	3	7
4	3	2	1	1	4	6
3	4	1	0	0	5	5
2	4	1	0	0	3	4
1	3	2	1	1	2	3
0	4	3	2	2	3	4
	0	1	2	3	4	5

5	4	4	3	2	3	7
4	3	2	1	1	4	6
3	4	1	0	0	5	5
2	4	1	0	0	3	4
1	3	2	1	1	2	3
0	4	3	2	4	3	4
	0	1	2	3	4	5

5	4	4	3	2	3	8
4	3	2	1	1	4	7
3	4	1	0	0	5	6
2	4	1	0	0	4	5
1	3	2	1	1	2	3
0	4	3	2	5	4	5
	0	1	2	3	4	5

5	4	4	3	2	3	8
4	3	2	1	1	4	7
3	4	1	0	0	5	6
2	4	1	0	0	8	7
1	3	2	5	6	7	8
0	4	3	4	9	8	9
	0	1	2	3	4	5

5	4	4	3	2	3	8
4	3	2	1	1	4	7
3	4	1	0	0	5	6
2	5	1	0	0	8	7
1	6	7	10	10	9	10
0	7	8	9	11	10	11
	0	1	2	3	4	5

5	4	3	2	2	3	8
4	3	2	2	1	4	7
3	4	1	0	0	5	6
2	5	1	0	0	8	7
1	6	7	10	10	9	10
0	7	8	9	11	10	11
	0	1	2	3	4	5

5	4	5	6	5	6	11
4	3	2	3	4	7	10
3	4	1	0	0	8	9
2	5	1	0	0	11	10
1	6	7	10	11	12	13
0	7	8	9	14	13	14
	0	1	2	3	4	5

5	5	6	7	6	7	12
4	4	3	4	5	8	11
3	5	2	0	0	9	10
2	6	1	0	0	12	11
1	7	8	11	12	13	14
0	8	9	10	15	14	15
	0	1	2	3	4	5

5	**5**	**6**	**7**	**6**	**7**	**12**
4	**4**	**3**	**4**	**5**	**8**	**11**
3	**5**	**2**	**0**	**0**	**9**	**10**
2	**6**	**1**	**0**	**0**	**12**	**11**
1	**7**	**8**	**11**	**12**	**13**	**14**
0	**8**	**9**	**10**	**15**	**14**	**15**
	0	1	2	3	4	5

stack

14,2
04,3
03,4
02,5
01,6

stack

22,0
12,1
13,2
14,3
04,4
03,5
02,6
01,7

MICRO-MOUSE MAZE SOLVING PROGRAM CODE

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
//SHAASTRA 2019 Micro-Mouse-Maze Solving Algorithm

#define SIZE 16    //Maze of 16 x 16

#define TRUE 1
#define FALSE 0

//PIN CONNECTIONS

//Motor Driver Connections

#define enR 10
#define Rf 9
#define Rb 8
#define enL 3
#define Lf 4
#define Lb 5

// Ultrasonic Connections
const int trig_right = A0;
const int echo_right = A1;
const int trig_front = A2;
const int echo_front = A3;
const int trig_left = A4;
const int echo_left = A5;

long duration_right, duration_front, duration_left;
int distance_right, distance_front, distance_left;

//Opto-coupler Encoder Connections
#define pin 2
int counter=0;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
//SHAASTRA 2019 Micro-Mouse-Maze Solving Algorithm

#define SIZE 16    //Maze of 16 x 16

#define TRUE 1
#define FALSE 0

//PIN CONNECTIONS

//Motor Driver Connections

#define enR 10
#define Rf 9
#define Rb 8
#define enL 3
#define Lf 4
#define Lb 5

// Ultrasonic Connections
const int trig_right = A0;
const int echo_right = A1;
const int trig_front = A2;
const int echo_front = A3;
const int trig_left = A4;
const int echo_left = A5;

long duration_right, duration_front, duration_left;
int distance_right, distance_front, distance_left;

//Opto-coupler Encoder Connections
#define pin 2
int counter=0;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Upload

Shaastra_Micro_Mouse_Maze_ppt_code

```
//Opto-coupler Encoder Connections
#define pin 2
int counter=0;

int State;
int LastState;


// Referance Constants
#define MAP_IJ this_maze->mapp[i][j]
#define MAP this_maze->mapp

#define FLOODVAL this_node->floodval
#define ROW this_node->row
#define COL this_node->column
#define VISITED this_node->visited
#define LEFT this_node->left
#define RIGHT this_node->right
#define UP this_node->up
#define DOWN this_node->down

#define STACKSIZE 256
int top;


//Start of maze
#define START_X 0
#define START_Y 0
#define LARGEVAL 256

// Directions
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Upload

Shaastra_Micro_Mouse_Maze_ppt_code

```
// Directions
#define NORTH 0
#define EAST 1
#define SOUTH 2
#define WEST 3

//STRUCTURE DEFINITIONS

typedef struct Node {


  short floodval;
  short row;
  short column;
  short visited;

  //pointers to neighbouring node
  struct Node * left;
  struct Node * right;
  struct Node * up;
  struct Node * down;

} Node;


typedef struct Maze {

  Node * mapp [SIZE][SIZE];

} Maze;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
typedef struct Stack {

  short top;
  Node * the_stack [STACKSIZE];

} Stack;

//FUNCTION PROTOTYPES

//Turning Functions
void turn_right();
void turn_left();
void U_turn();

//Ultrasonic Sensing Functions
int ultrasonic_sense_right();
int ultrasonic_sense_front();
int ultrasonic_sense_left();

//Orientation Defining Functions
void motor_turn_dir(int curr_dir , int next_dir);
void move_dir (Maze * this_maze, short * i, short * j, short * dir);

// Node Functions
Node * new_Node (const short i, const short j);
void delete_Node (Node ** npp);
void set_wall (Node * this_node, const short dir);
void set_value (Node * this_node, const short value);
void set_visited (Node * this_node);
void visit_Node (Maze * this_maze, Stack * this_stack, short i, short j, short & flag, short * dir);

// Floodfill algorithm Functions
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

New

Shaastra_Micro_Mouse_Maze_ppt_code

```
// Floodfill algorithm Functions
short get_smallest_neighbor (Node * this_node);
short get_smallest_neighbor_dir (Node * this_node);
short floodval_check(Node * this_node) ;
void update_floodval (Node * this_node);
void push_open_neighbors (Node * this_node, Stack * this_stack);
void propagate_floodvalue(Node * this_node, Stack * this_stack);
void flood_fill (Node * this_node, Stack * this_stack,  short & reflood_flag);

//Travelled Path Run
void store_path(Node * this_node, Stack * this_stack);
void path_to_travel(Stack * this_stack1 , Stack * this_stack2);
void follow_the_travelled_path(Maze * this_maze, Stack * this_stack, short * i, short * j, short *dir);

// Maze Functions
Maze * new_Maze ();
void delete_Maze (Maze ** mpp);


// Stack Functions
Stack * new_Stack();
void delete_Stack (Stack ** spp);
int is_empty_Stack (Stack * this_stack);
void push (Stack * this_stack, Node * this_node);
void pop (Stack * this_stack, Node ** temp);

// MAZE FUNCTIONS

//Maze Constructor
Maze * new_Maze () {

  Maze * this_maze;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Upload

Shaastra_Micro_Mouse_Maze_ppt_code

```
// MAZE FUNCTIONS

//Maze Constructor
Maze * new_Maze () {

  Maze * this_maze;
  short i, j;

  this_maze = (Maze *) malloc(sizeof(Maze));

  for (i = 0; i < SIZE; ++i)
    for (j = 0; j < SIZE; ++j)
      MAP_IJ = new_Node (i, j);

  //setting the neighbors pointers
  for (i = 0; i < SIZE; i++)
    for (j = 0; j < SIZE; j++) {
      MAP_IJ->up = (i == SIZE-1) ? NULL : (this_maze->mapp[i+1][j]);
      MAP_IJ->right = (j == SIZE-1) ? NULL : (this_maze->mapp[i][j+1]);
      MAP_IJ->left = (j == 0) ? NULL : (this_maze->mapp[i][j-1]);
      MAP_IJ->down = (i == 0) ? NULL : (this_maze->mapp[i-1][j]);
    }

  return this_maze;
}

// Maze Destructor
void delete_Maze (Maze ** mpp) {

  short i, j;

  for (i = 0; i < SIZE; i++)
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Upload

Shaastra_Micro_Mouse_Maze_ppt_code

```
// Maze Destructor
void delete_Maze (Maze ** mpp) {

  short i, j;

  for (i = 0; i < SIZE; i++)
    for (j = 0; j < SIZE; j++)
      delete_Node (&((*mpp)->mapp[i][j]));

  free(*mpp);
  *mpp = 0;
}

//STACK FUNCTIONS

// Stack Constructor
Stack * new_Stack()
{

  Stack * this_stack = (Stack *) malloc(sizeof(Stack));
  this_stack->top = -1;
  return this_stack;
}

// Stack Destructor
void delete_Stack (Stack ** spp)
{

  if (spp == 0 || *spp == 0)
  {
    return;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
// Stack Destructor
void delete_Stack (Stack ** spp)
{

  if (spp == 0 || *spp == 0)
  {
    return;
  }
  free(*spp);
  *spp = 0;

}


// Checks if this_stack is empty
int is_empty_Stack (Stack * this_stack) {

  if (this_stack->top == -1)
    return 1;
  else return 0;
}

// Pushes an element to the top of this_stack
void push (Stack * this_stack, Node * this_node) {

  short index;
  this_stack->top += 1;
  index = this_stack->top;
  this_stack->the_stack[index] = this_node;


}
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Upload

Shaastra_Micro_Mouse_Maze_ppt_code

```
// Pops the top element of this_stack
void pop (Stack * this_stack, Node ** temp)
{
  short index;
  index = this_stack->top;
  *temp = this_stack->the_stack[index];
  this_stack->top -= 1;

}
//NODE FUNCTIONS

//Node Constructor
Node * new_Node (const short i, const short j)
{
  Node * this_node;
  this_node = (Node *) malloc(sizeof(Node));

  ROW = i;
  COL = j;
  VISITED = FALSE;

  if (i <= 7 && j <= 7)
    FLOODVAL = (7 - i) + (7- j) ;

  else if (i <= 7 && j >= 8)
    FLOODVAL = (7 - i) + (j - 8) ;

  else if (i >= 8 && j <= 7)
    FLOODVAL = (i - 8) + (7 - j) ;

  else
    FLOODVAL = (i - 8) + (j - 8) ;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
//Node Constructor
Node * new_Node (const short i, const short j)
{
  Node * this_node;
  this_node = (Node *) malloc(sizeof(Node));

  ROW = i;
  COL = j;
  VISITED = FALSE;

  if (i <= 7 && j <= 7)
    FLOODVAL = (7 - i) + (7- j) ;

  else if (i <= 7 && j >= 8)
    FLOODVAL = (7 - i) + (j - 8) ;

  else if (i >= 8 && j <= 7)
    FLOODVAL = (i - 8) + (7 - j) ;

  else
    FLOODVAL = (i - 8) + (j - 8) ;

  return this_node;
}



// Node Destructor
void delete_Node (Node ** npp) {

  free (*npp);
  *npp = 0;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
// Function for setting this node's floodval to a specific value
void set_value (Node * this_node, const short value) {
  FLOODVAL = value;
}

// Function for setting this node's VISITED to a specific value
void set_visited (Node * this_node) {
  VISITED = TRUE;
}

// Function for setting the walls of this node
void set_wall (Node * this_node, const short dir) {

  switch (dir) {

    case NORTH :
      if (ROW != SIZE-1) {
        UP = NULL;
      } break;

    case SOUTH :
      if (ROW != 0) {
        DOWN = NULL;
      } break;

    case EAST :
      if (COL != SIZE - 1) {
        RIGHT = NULL;
      } break;

    case WEST :
      if (COL != 0) {
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
    case EAST :
      if (COL != SIZE - 1) {
        RIGHT = NULL;
      } break;

    case WEST :
      if (COL != 0) {
        LEFT = NULL;
      } break;
  }
}
//FLOOD FILL ALGORITHM FUNCTIONS

// function for obtaining this_node's smallest neighbor's floodval
short get_smallest_neighbor (Node * this_node) {

  short smallestneighbor = LARGEVAL;

  if (UP != NULL && (UP->down != NULL) && (UP->floodval) < smallestneighbor)
    smallestneighbor = UP->floodval;

  if (RIGHT != NULL && (RIGHT->left != NULL) && (RIGHT->floodval) < smallestneighbor)
    smallestneighbor = RIGHT->floodval;

  if (LEFT != NULL && (LEFT->right != NULL) && (LEFT->floodval) < smallestneighbor)
    smallestneighbor = LEFT->floodval;

  if (DOWN != NULL && (DOWN->up != NULL) && (DOWN->floodval) < smallestneighbor)
    smallestneighbor = DOWN->floodval;

  return smallestneighbor;
}
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
// function for obtaining this nodes's smallest neighbor's direction
short get_smallest_neighbor_dir (Node * this_node)
{

  short smallestval;
  smallestval = get_smallest_neighbor(this_node);

      //returning a direction means that path is reachable
      if ((UP != NULL) && (UP->floodval == smallestval) && (UP->visited == FALSE))
       return NORTH;
      else if ((RIGHT != NULL) && (RIGHT->floodval == smallestval) && (RIGHT->visited == FALSE))
        return EAST;
      else if ((LEFT != NULL) && (LEFT->floodval == smallestval) && (LEFT->visited == FALSE))
        return WEST;
      else if ((DOWN != NULL) && (DOWN->floodval == smallestval) && (DOWN->visited == FALSE))
        return SOUTH;

    if ((UP != NULL) && (UP->floodval == smallestval))
       return NORTH;
      else if ((RIGHT != NULL) && (RIGHT->floodval == smallestval))
        return EAST;
      else if ((LEFT != NULL) && (LEFT->floodval) == smallestval)
        return WEST;
      else //if ((DOWN != NULL) && (DOWN->floodval == smallestval))
        return SOUTH;

}

// function for modified flood_fill
short floodval_check(Node * this_node) {
  if (get_smallest_neighbor (this_node) + 1 == this_node->floodval)
    return TRUE;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
// function for modified flood_fill
short floodval_check(Node * this_node) {
  if (get_smallest_neighbor (this_node) + 1 == this_node->floodval)
    return TRUE;
  else
  return FALSE;
}

//Floodfill Update Function
void update_floodval (Node * this_node)
{
  this_node->floodval = get_smallest_neighbor (this_node) + 1;
}

// pushes the open neighboring cells of this node to the stack
void push_open_neighbors (Node * this_node, Stack * this_stack)
{

  if (UP != NULL && UP->down != NULL)
    push (this_stack, UP);

  if (LEFT != NULL && LEFT->right != NULL)
    push (this_stack, LEFT);

  if (RIGHT != NULL && RIGHT->left != NULL)
    push (this_stack, RIGHT);

  if (DOWN != NULL && DOWN->up != NULL)
    push (this_stack, DOWN);

}
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
// main function for propagating the flood values

void propagate_floodvalue(Node * this_node, Stack * this_stack){

    Node * temp;
    short check;

    push_open_neighbors(this_node, this_stack);
    while(!is_empty_Stack(this_stack)) {
      check = floodval_check(this_stack->the_stack[top]);
      if (!check)
  {
    update_floodval(this_stack->the_stack[top]);
    propagate_floodvalue(this_stack->the_stack[top],this_stack);
  }
      else
      {
        pop (this_stack, &temp);
      }
 }
}

void flood_fill (Node * this_node, Stack * this_stack, short & reflood_flag) {

  short check;

  if (!reflood_flag)
    if (ROW == SIZE / 2 || ROW == SIZE / 2 - 1)
        if (COL == SIZE / 2 || COL == SIZE / 2 - 1)
        {
          reflood_flag = TRUE;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
void flood_fill (Node * this_node, Stack * this_stack, short & reflood_flag) {

  short check;

  if (!reflood_flag)
    if (ROW == SIZE / 2 || ROW == SIZE / 2 - 1)
        if (COL == SIZE / 2 || COL == SIZE / 2 - 1)
        {
          reflood_flag = TRUE;
          return;
        }

   check = floodval_check (this_node);

  if (!check)
  {
    update_floodval(this_node);
    push (this_stack, this_node);
    propagate_floodvalue(this_node,this_stack);
  }
}

//TRAVELLED PATH RUN ALGORITHM

//Storing only required elements
void store_path(Node * this_node, Stack * this_stack)
{
  if(ROW==0 && COL==0)
     push(this_stack,this_node);
  else if(FLOODVAL== this_stack->the_stack[top]->floodval - 1)
     push (this_stack,this_node);
}
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
//Final path to run from start to destination
void path_to_travel(Stack * this_stack1 , Stack * this_stack2)
{
 Node * temp;
 while(!is_empty_Stack(this_stack1))
 {
  push (this_stack2, this_stack1->the_stack[top]);
  pop (this_stack1, &temp);
 }
}

//ORIENTATION SPECIFICATION FUNCTION

//Motor turn definition
void motor_turn_dir(int curr_dir , int next_dir)
  {
    if(next_dir==curr_dir+1 || next_dir==curr_dir-3)
    {
      turn_right();
    }
    else if(next_dir==curr_dir-1 || next_dir==curr_dir+3)
    {
      turn_left();
    }
    else if(next_dir==curr_dir+2 || next_dir==curr_dir-2)
    {
      U_turn();
    }

  }
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
//Move to specified direction and update the current node value
void move_dir (Maze * this_maze, short * i, short * j, short * dir) {


  Node * this_node;
  short next_dir;

  this_node = this_maze->mapp[(*i)][(*j)];
  next_dir = get_smallest_neighbor_dir(this_node);

  int curr_dir = (*dir);
  motor_turn_dir(curr_dir , next_dir);

  if (next_dir == NORTH)
    (*i) = (*i) + 1;
  else if (next_dir == EAST)
    (*j) = (*j) + 1;
  else if (next_dir == WEST)
    (*j) = (*j) - 1;
  else if (next_dir == SOUTH)
    (*i) = (*i) - 1;

  (*dir) = next_dir;
}

//Follow the Travelled path in Final Run
void follow_the_travelled_path(Maze * this_maze, Stack * this_stack, short * i, short * j, short * dir)
{
  Node * this_node;
  Node * temp;
  short face_dir;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
//Follow the Travelled path in Final Run
void follow_the_travelled_path(Maze * this_maze, Stack * this_stack, short * i, short * j, short * dir)
{
  Node * this_node;
  Node * temp;
  short face_dir;
  face_dir = (*dir);
  while(!is_empty_Stack(this_stack))
  {
    this_node = this_maze->mapp[(*i)][(*j)];
  int prev_i, prev_j;
  prev_i=ROW;
  prev_j=COL;

  pop (this_stack, &temp);
  if (this_stack->the_stack[top]->row == prev_i + 1)
  {
    motor_turn_dir(face_dir , NORTH);
    (*i) = (*i) + 1;
    *dir = NORTH;
  }
  else if(this_stack->the_stack[top]->column == prev_j + 1)
  {
    motor_turn_dir(face_dir , EAST);
    (*j) = (*j) + 1;
    *dir = EAST;
  }
  else if(this_stack->the_stack[top]->column == prev_j - 1)
  {
    motor_turn_dir(face_dir , WEST);
    (*j) = (*j) - 1;
    *dir = WEST;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
else if(this_stack->the_stack[top]->column == prev_j - 1)
{
  motor_turn_dir(face_dir , WEST);
  (*j) = (*j) - 1;
  *dir = WEST;

}
else if(this_stack->the_stack[top]->row == prev_i - 1)
{
  motor_turn_dir(face_dir , SOUTH);
  (*i) = (*i) - 1;
  *dir = SOUTH;
}

  counter=0;

  analogWrite(enR,255);
  analogWrite(enL,255);
  digitalWrite(Rf,HIGH);
  digitalWrite(Rb,LOW);
  digitalWrite(Lf, HIGH);
  digitalWrite(Lb, LOW);


  State=digitalRead(pin);
 if(State!=LastState)
 {
    counter++;
 }
 LastState=State;

 if(counter==7)
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
    State=digitalRead(pin);
   if(State!=LastState)
   {
      counter++;
   }
   LastState=State;

   if(counter==7)
   {

    digitalWrite(Rf,LOW);
    digitalWrite(Rb,LOW);
    digitalWrite(Lf,LOW);
    digitalWrite(Lb,LOW);

    delay(1000);
   }

  }

}
//Functions That get executed when the mouse has visited the node
void visit_Node (Maze * this_maze, Stack * this_stack, short i, short j, short & flag, short * dir, Stack * store_stack) {

  Node * this_node;
  int northwall, eastwall, southwall, westwall;

  this_node = this_maze->mapp[i][j];

short face_dir;
face_dir = (*dir);
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
//Functions That get executed when the mouse has visited the node
void visit_Node (Maze * this_maze, Stack * this_stack, short i, short j, short & flag, short * dir, Stack * store_stack) {

  Node * this_node;
  int northwall, eastwall, southwall, westwall;

  this_node = this_maze->mapp[i][j];

short face_dir;
face_dir = (*dir);

   distance_right = ultrasonic_sense_right();
   distance_front = ultrasonic_sense_front();
   distance_left = ultrasonic_sense_left();

   if(face_dir == NORTH)
   {
     if(distance_right<=6)
      eastwall=1;
     if(distance_front<=6)
      northwall=1;
     if(distance_left<=6)
      westwall=1;
   }

   else if(face_dir == EAST)
   {
     if(distance_right<=6)
      southwall=1;
     if(distance_front<=6)
      eastwall=1;
     if(distance_left<=6)
```


```
  else if(face_dir == EAST)
  {
    if(distance_right<=6)
     southwall=1;
    if(distance_front<=6)
     eastwall=1;
    if(distance_left<=6)
     northwall=1;
  }

  else if(face_dir == WEST)
  {
    if(distance_right<=6)
     northwall=1;
    if(distance_front<=6)
     westwall=1;
    if(distance_left<=6)
     southwall=1;
  }

  else if(face_dir == SOUTH)
  {
    if(distance_right<=6)
     westwall=1;
    if(distance_front<=6)
     southwall=1;
    if(distance_left<=6)
     eastwall=1;
  }

//detects the walls and update its neighbours
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
//detects the walls and update its neighbours
  if (northwall) {
    set_wall(this_node, NORTH);
  }
  if (eastwall) {
    set_wall(this_node, EAST);
  }
  if (southwall) {
    set_wall(this_node, SOUTH);
  }
  if (westwall) {
    set_wall(this_node, WEST);
  }

   flood_fill (this_node, this_stack, flag);
   store_path(this_node, store_stack);
   set_visited (this_node);

}
//Motor Turn Functions
void turn_right()
{

    //take a right turn
  counter=0;
  analogWrite(enR,255);
  digitalWrite(Rf,LOW);
  digitalWrite(Rb,HIGH);
  digitalWrite(Lf,LOW);
  digitalWrite(Lb,LOW);

  State=digitalRead(pin);
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
//Motor Turn Functions
void turn_right()
{

    //take a right turn
  counter=0;
  analogWrite(enR,255);
  digitalWrite(Rf,LOW);
  digitalWrite(Rb,HIGH);
  digitalWrite(Lf,LOW);
  digitalWrite(Lb,LOW);

  State=digitalRead(pin);
   if(State!=LastState)
   {
      counter++;
  }
   LastState=State;

   if(counter==8)
   {
    digitalWrite(Rf,LOW);
    digitalWrite(Rb,LOW);
    delay(1000);
   }
}

void turn_left()
{

    //take a left turn
  counter=0;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Upload

Shaastra_Micro_Mouse_Maze_ppt_code

```
void turn_left()
{

    //take a left turn
  counter=0;
  analogWrite(enR,255);
  digitalWrite(Rf,HIGH);
  digitalWrite(Rb,LOW);
  digitalWrite(Lf,LOW);
  digitalWrite(Lb,LOW);

  State=digitalRead(pin);
   if(State!=LastState)
   {
      counter++;
  }
   LastState=State;

   if(counter==8)
   {
    digitalWrite(Rf,LOW);
    digitalWrite(Rb,LOW);
    delay(1000);
   }
}

void U_turn()
{


    //take a Uturn
  counter=0;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
void U_turn()
{


    //take a Uturn
  counter=0;
  analogWrite(enR,255);
  digitalWrite(Rf,HIGH);
  digitalWrite(Rb,LOW);
  digitalWrite(Lf,LOW);
  digitalWrite(Lb,LOW);

  State=digitalRead(pin);
   if(State!=LastState)
   {
      counter++;
  }
   LastState=State;

   if(counter==16)
   {
    digitalWrite(Rf,LOW);
    digitalWrite(Rb,LOW);
    delay(1000);
   }
}

//Ultrasonic Detection Functions
int ultrasonic_sense_right()
{
  int distance;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

New

Shaastra_Micro_Mouse_Maze_ppt_code

```
//Ultrasonic Detection Functions
int ultrasonic_sense_right()
{
  int distance;
delay(500);
digitalWrite(trig_right, LOW);
delayMicroseconds(2);
digitalWrite(trig_right, HIGH);
delayMicroseconds(10);
digitalWrite(trig_right, LOW);
duration_right = pulseIn(echo_right, HIGH);
distance = (duration_right)*0.034/2;
  return distance;
}

int ultrasonic_sense_front()
{
  int distance;
  delay(500);
digitalWrite(trig_front, LOW);
delayMicroseconds(2);
digitalWrite(trig_front, HIGH);
delayMicroseconds(10);
digitalWrite(trig_front, LOW);
duration_front = pulseIn(echo_front, HIGH);
distance = (duration_front)*0.034/2;
  return distance;
}

int ultrasonic_sense_left()
{
  int distance;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
int ultrasonic_sense_left()
{
  int distance;
  delay(500);
digitalWrite(trig_left, LOW);
delayMicroseconds(2);
digitalWrite(trig_left, HIGH);
delayMicroseconds(10);
digitalWrite(trig_left, LOW);
duration_left = pulseIn(echo_left, HIGH);
distance = (duration_left)*0.034/2;
  return distance;
}

//Variable declarations
  Maze * my_maze;
  Stack * my_stack;
  Stack * store_path_stack;
  Stack * follow_travelled_path_stack;
  short found_dest;
  short face_dir;
  short x, y;
  Node * temp;

//This is setup function for arduino
void setup() {

  pinMode(enR, OUTPUT);
  pinMode(Rf, OUTPUT);
  pinMode(Rb, OUTPUT);
  pinMode(enL, OUTPUT);
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Verify

Shaastra_Micro_Mouse_Maze_ppt_code

```
//This is setup function for arduino
void setup() {

  pinMode(enR, OUTPUT);
  pinMode(Rf, OUTPUT);
  pinMode(Rb, OUTPUT);
  pinMode(enL, OUTPUT);
  pinMode(Lf, OUTPUT);
  pinMode(Lb, OUTPUT);

pinMode(trig_right, OUTPUT);
pinMode(echo_right, INPUT);
pinMode(trig_front, OUTPUT);
pinMode(echo_front, INPUT);
pinMode(trig_left, OUTPUT);
pinMode(echo_left, INPUT);



//Allocate memory
  my_maze = new_Maze();
  my_stack = new_Stack();
  store_path_stack = new_Stack();
  follow_travelled_path_stack = new_Stack();

  //intialise variables
  x = START_X;
  y = START_Y + 1;
  face_dir = NORTH;
  found_dest = FALSE;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
  //intialise variables
  x = START_X;
  y = START_Y + 1;
  face_dir = NORTH;
  found_dest = FALSE;

  pinMode(pin,INPUT);
    LastState=digitalRead(pin);

    analogWrite(enR,255);
    analogWrite(enL,255);
    digitalWrite(Rf,HIGH);
    digitalWrite(Rb,LOW);
    digitalWrite(Lf, HIGH);
    digitalWrite(Lb, LOW);


}


void loop() {

while(!found_dest)
{
  State=digitalRead(pin);
   if(State!=LastState)
   {
      counter++;
  }
   LastState=State;
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Upload

Shaastra_Micro_Mouse_Maze_ppt_code

```
void loop() {

while(!found_dest)
{
  State=digitalRead(pin);
   if(State!=LastState)
   {
      counter++;
  }
   LastState=State;

   if(counter==7)
   {

    digitalWrite(Rf,LOW);
    digitalWrite(Rb,LOW);
    digitalWrite(Lf,LOW);
    digitalWrite(Lb,LOW);

    delay(1000);
   }

   visit_Node(my_maze, my_stack, y, x, found_dest, face_dir, store_path_stack);
   move_dir(my_maze, &y, &x, &face_dir);

    counter=0;

    analogWrite(enR,255);
    analogWrite(enL,255);
    digitalWrite(Rf,HIGH);
    digitalWrite(Rb,LOW);
    digitalWrite(Lf, HIGH);
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
  counter=0;

  analogWrite(enR,255);
  analogWrite(enL,255);
  digitalWrite(Rf,HIGH);
  digitalWrite(Rb,LOW);
  digitalWrite(Lf, HIGH);
  digitalWrite(Lb, LOW);
}

  analogWrite(enR,255);
  analogWrite(enL,255);
  digitalWrite(Rf,LOW);
  digitalWrite(Rb,LOW);
  digitalWrite(Lf, LOW);
  digitalWrite(Lb, LOW);
 delay(120000);

  path_to_travel( store_path_stack , follow_travelled_path_stack );

  analogWrite(enR,255);
  analogWrite(enL,255);
  digitalWrite(Rf,HIGH);
  digitalWrite(Rb,LOW);
  digitalWrite(Lf, HIGH);
  digitalWrite(Lb, LOW);
 State=digitalRead(pin);
  if(State!=LastState)
  {
     counter++;
 }
```

Shaastra_Micro_Mouse_Maze_ppt_code | Arduino 1.8.7

File Edit Sketch Tools Help

Shaastra_Micro_Mouse_Maze_ppt_code

```
   analogWrite(enL,255);
   digitalWrite(Rf,HIGH);
   digitalWrite(Rb,LOW);
   digitalWrite(Lf, HIGH);
   digitalWrite(Lb, LOW);
 State=digitalRead(pin);
  if(State!=LastState)
  {
     counter++;
 }
  LastState=State;

  if(counter==7)
  {

   digitalWrite(Rf,LOW);
   digitalWrite(Rb,LOW);
   digitalWrite(Lf,LOW);
   digitalWrite(Lb,LOW);

   delay(1000);
  }
     x = START_X;
     y = START_Y + 1;

    face_dir = NORTH;
    follow_the_travelled_path(my_maze, follow_travelled_path_stack , y , x, face_dir);
      delete_Maze(&my_maze);
      delete_Stack (&my_stack);
      delete_Stack (&store_path_stack);
      delete_Stack (&follow_travelled_path_stack);
}
```

Thank You