TUTORIAL: COLONIZER

PROBLEM STATEMENT:

Build an image processing robot that has to find the shortest route in terms of distance for visiting every Planet(Circle) exactly once. Video feed of the arena will be provided through an overhead camera.

HOW TO APPROACH:

- An image processing code to detect the planets to traverse.
- An algorithm to find the shortest possible route.
- A code for IC to give commands to the bot.

MATERIALS:

- 2 Wheels
- 1 Castor Wheel
- Metal Chassis
- 2 DC motors
- 1 Development Board
- 1 Programmer (ISP/Arduino)
- Motor Driver (L293D)
- 4 LEDs
- Relimates

MECHANISMS:

1. **Autonomous bot**
    a. Locomotion
    b. Microcontroller
2. **Computer Vision**
    a. Color Detection
    b. Shape Detection
    c. Path Planning

AUTONOMOUS BOT:

Locomotion

The most popular robot locomotion is achieved through differential drives which allows the robot to move in all directions and turn as well. It consists of 2 drive wheels on a common axis, and each wheel can independently be being driven either forward or backwards.

Microcontroller

It is a control device which incorporates a microprocessor. It takes input from the device it is controlling and controls the device by sending signals to different components in the device. We can easily pre-program it to make the robot move as per instructions given.

The following code can be burnt on Arduino:

```
#define L_MOTOR_POSITIVE 9
#define L_MOTOR_NEGATIVE 10
#define L_MOTOR_ENABLE 3
#define R_MOTOR_POSITIVE 6
#define R_MOTOR_NEGATIVE 11
#define R_MOTOR_ENABLE 5
#define LED_PIN 13
#define MAX_SPEED_R 240
#define MAX_SPEED_L 180

/*
 * Movements:
 *      W := forward
 *      S := stop
 *      D := right
 *      A := left
 *      R := extreme right
 *      L := extreme left
 *      X := reverse
 */

void setup() {
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(9600);
}

int incomingByte = Serial.read();

void loop() {
  if (Serial.available() > 0) {
```

```
incomingByte = Serial.read();

Serial.print("I received: ");
Serial.println(incomingByte);
if (incomingByte == 'A') {
  digitalWrite(L_MOTOR_POSITIVE, LOW);
  digitalWrite(L_MOTOR_NEGATIVE, LOW);

  digitalWrite(R_MOTOR_POSITIVE, HIGH);
  digitalWrite(R_MOTOR_NEGATIVE, LOW);

  analogWrite(L_MOTOR_ENABLE, MAX_SPEED_L);
  analogWrite(R_MOTOR_ENABLE, MAX_SPEED_R);
}
else if (incomingByte == 'D') {
  digitalWrite(L_MOTOR_POSITIVE, HIGH);
  digitalWrite(L_MOTOR_NEGATIVE, LOW);

  digitalWrite(R_MOTOR_POSITIVE, LOW);
  digitalWrite(R_MOTOR_NEGATIVE, LOW);

  analogWrite(L_MOTOR_ENABLE, MAX_SPEED_L);
  analogWrite(R_MOTOR_ENABLE, MAX_SPEED_R);
}
else if (incomingByte == 'W') {
  digitalWrite(L_MOTOR_POSITIVE, HIGH);
  digitalWrite(L_MOTOR_NEGATIVE, LOW);

  digitalWrite(R_MOTOR_POSITIVE, HIGH);
  digitalWrite(R_MOTOR_NEGATIVE, LOW);

  analogWrite(L_MOTOR_ENABLE, MAX_SPEED_L);
  analogWrite(R_MOTOR_ENABLE, MAX_SPEED_R);
}
else if (incomingByte == 'S') {
  digitalWrite(L_MOTOR_POSITIVE, LOW);
  digitalWrite(L_MOTOR_NEGATIVE, LOW);

  digitalWrite(R_MOTOR_POSITIVE, LOW);
  digitalWrite(R_MOTOR_NEGATIVE, LOW);

  analogWrite(L_MOTOR_ENABLE, MAX_SPEED_L);
  analogWrite(R_MOTOR_ENABLE, MAX_SPEED_R);
}

else if (incomingByte == 'B') {
```

```
        digitalWrite(13, HIGH);
        delay(1000);
        digitalWrite(13, LOW);
      }
    }
}
```

COMPUTER VISION :

Color Detection:

For lighting LEDs corresponding to the color of the planet we need to first detect the color of planet. Have separate windows for each color so that colors in these particular range correspond to the color of the LEDs.

Shape Detection :

We need to identify obstacles(asteroids) that are triangular and planets that are circular in shape. A good approach to solve this problem would be to segment out the shapes from the image that is taken by the camera. A method called Hough circles could be used to figure out the planets.

Path Planning:

The problem can be modelled as weighted graph, such that planets depict the graphs vertices and the path are graph's edges. It is a minimization problem starting and finishing at specified vertices after having visited each other vertex exactly once. There are several methods to deduce the optimum path for traversal, one of which is A* ("A star").

A* (**A star**) : It is best-known form of Best First search.
The secret to its success is that it combines the pieces of information that Dijkstra's algorithm uses (favoring vertices that are close to the starting point) *and* information that Greedy Best-First-Search uses (favoring vertices that are close to the goal).

A* balances the two as it moves from the starting point to the goal. Each time through the main loop, it examines the vertex n that has the lowest f(n) = g(n) + h(n).

g(n) : the *exact cost* of the path from the starting point to any vertex n.
h(n) : represents the heuristic *estimated cost* from vertex n to the goal.
f(n) : estimated total cost of path through n to goal. It is implemented using priority queue by increasing f(n).

pseudo code:

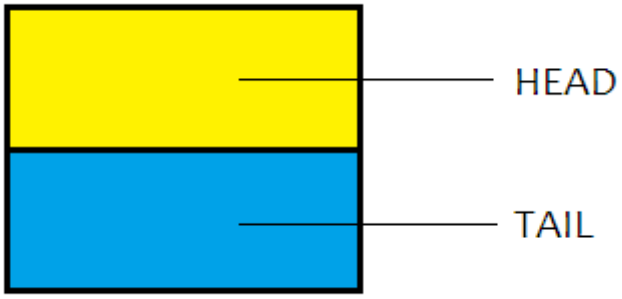
```
initialize the open list
```

```
initialize the closed list
put the starting node on the open list (you can leave its f at zero)

while the open list is not empty
      find the node with the least f on the open list, call it "q"
      pop q off the open list
      generate q's 8 successors and set their parents to q
      for each successor
            if successor is the goal, stop the search
      successor.g = q.g + distance between successor and q
      successor.h = distance from goal to successor
      successor.f = successor.g + successor.h

      if a node with the same position as successor is in the OPEN
list which/ has a lower f than successor, skip this successor
      if a node with the same position as successor is in the CLOSED
list which has a lower f than successor, skip this successor
      otherwise, add the node to the open list
      end
      push q on the closed list
end
```

You can also visit the following links for other approaches,
http://web.mit.edu/eranki/www/tutorials/search/
http://students.ceid.upatras.gr/~papagel/project/tspprobl.htm
https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/

Tracking the bot:

A marker can be used to track the position of the bot throughout the run.
The marker will contain 2 rectangles of different RGB values.



Scan the marker to find the center of the marker and the two rectangles which can be differentiated by color. The center of the marker will determine the position of the bot and the line joining the center of the two rectangles will provide its direction.