Global Path Planning for Mobile robots using Particle Swarm Optimization

Overview

Autonomous mobile robots are used in the environment where many human beings are working, cooperating with robots. In these environments, the collision-free path planning is one of the major problems to realize autonomous mobile robots. Since there are many stationary or moving obstacles in these environments, autonomous mobile robots should plan their own path that can avoid not only stationary obstacles but also moving ones. The main problem in robot path planning is to find a motion trajectory from a starting position to a goal position regarding to some optimization criteria such as shortest distance or minimum time. Path Planning is one of the most vital issues in the navigation of mobile robot, which means to find out an optimized collision free path from the start state to the goal state according to some performance merits. It can be classified into two categories: global path planning with all the information of the robot's known environment and local path planning in a partly or totally unknown environment. In global navigation methods cost of environmental change, especially in dynamic environment is very high, because supplying a new map is difficult. Local path planning methods can detect the unknown environment, and it does not need entire environment model. In this work, a method of global navigation based on particle swarm optimization technique is proposed.

There have been many algorithms for global path planning, such as artificial potential field, visibility graph, and cell decomposition etc. PSO is a heuristic search technique that is inspired by the behavior of bird flocks. The relative simplicity, the fast convergence and the population-based feature of PSO have made it a considerable viable alternative for solving the robot path planning problem. PSO has been used for target search applications, such as landmine detection, fire fighting, and military surveillance, and are an effective technique for swarm robotic search problems.

Exploration of Path Planning algorithms

Path Planning algorithms previously explored were,

- Dijkstra's algorithm
- A-star, D-star
- Probability roadmaps,
- Rapidly exploring random trees(RRT) and RRT-star.

All these algorithms were suited for a navigation of a single bot scenario, where the sensory information are used to determine the costmap and find out the path.

Evolutionary algorithms

These algorithms are based on the evolution of the species. Evolutionary algorithms are characterized by the existence of a population of individuals exposed to environmental pressure, which leads to natural selection, i.e. the survival of the fittest, and in turn the increase of the average fitness of the population. Fitness is the measure of the degree of adaptation of an organism to its environment; the bigger the fitness is, the more the organism is fit and adapted to the environment. In general, evolutionary algorithms focus only on a subset of mechanisms defined over the biological evolutionary process.

Explored evolutionary algorithms were:

- Ant Colony Optimization
- Particle Swarm Optimization
- Genetic algorithms
- Artificial Bee Colony
- Cuckoo Search

Among these techniques, we selected to continue working on Ant Colony Optimizer and Particle Swarm Optimization, though we continued with Particle Swarm optimization for its simplicity and faster convergence over ACO.

ACO and PSO Algorithm

Pseudo code for Particle Swarm Optimization

```
Input: ProblemSize, Population_size
Output: P_g_best
Population ← ∅
P_g_best ← ∅
For (i = 1 To Population_size)
  P_velocity ← RandomVelocity()
  P_position ← RandomPosition(Population_size)
  P_p_best ← P_position
  If (Cost(P_p_best) ≤ Cost(P_g_best))
    P_g_best ← P_p_best
  End
End
While (¬StopCondition())
  For (P ∈ Population)
    P_velocity ← UpdateVelocity(P_velocity, P_g_best, P_p_best)
    P_position ← UpdatePosition(P_position, P_velocity)
    If (Cost(P_position) ≤ Cost(P_p_best))
      P_p_best ← P_position
      If (Cost(P_p_best) ≤ Cost(P_g_best))
        P_g_best ← P_p_best
      End
    End
  End
End
Return (P_g_best)
```

Where,
a particle's velocity is updated using:

$$v_i(t+1) = v_i(t) + \left(c_1 \times rand() \times \left(p_i^{best} - p_i(t)\right)\right) + \left(c_2 \times rand() \times \left(p_{gbest} - p_i(t)\right)\right)$$

where $v_i(t+1)$ is the new velocity for the i^{th} particle, c_1 and c_2 are the weighting coefficients for the personal best and global best positions respectively, $p_i(t)$ is the i^{th} particle's position at time t, p_i^{best} is the i^{th} particle's best known position, and p_{gbest} is the best position known to the swarm. The $rand()$ function generate a uniformly random variable $\in [0, 1]$. Variants on this update equation consider best positions within a particles local neighborhood at time t.

A particle's position is updated using:

$$p_i(t+1) = p_i(t) + v_i(t)$$

Pseudo code for Ant Colony Optimization

- Initialize the population of solutions xi,j
- Evaluate the population
- cycle=1
- Repeat
- Produce new solutions (food source positions) υi,j in the neighborhood of xi,j for the Employed bees using the formula υi,j = xi,j + Φij(xi,j - xk,j) (k is a solution in the Neighborhood of i, Φ is a random number in the range [-1,1])and evaluate them
- Apply the greedy selection process between xi and υi
- Calculate the probability values Pi for the solutions xi by means of their fitness values using the equation (1)

$$P_i = \frac{fit_i}{\sum_{i=1}^{SN} fit_i} \qquad (1)$$

In order to calculate the fitness values of solutions we employed the following equation (eq. 2):

$$fit_i = \begin{cases} \frac{1}{1+f_i} & if\ f_i \geq 0 \\ 1 + abs(f_i) & if\ f_i < 0 \end{cases} \qquad (2)$$

Normalize Pi values into [0,1]

- Produce the new solutions (new positions) υi for the onlookers from the solutions xi, selected depending on Pi, and evaluate them
- Apply the greedy selection process for the onlookers between xi and υi
- Determine the abandoned solution (source), if exists, and replace it with a new randomly produced solution xi for the scout using the equation (3)
  - xij=minj + rand(0,1)*(maxj - minj) (3)
- Memorize the best food source position (solution) achieved so far
- cycle=cycle+1
- until cycle= Maximum Cycle Number (MCN)

PSO Implementation (2D holonomic case)

The initial implementation was made in a 2D search space to verify the working PSO algorithm. Here, a point robot with no dimension of its own is considered. This is a holonomic case, and thus the dependencies on orientation was nullified. The path generation from a given set of start and goal position was observed in the presence of various circular obstacles in the search area. The start and end cells were user given variables. The user can also choose the number of obstacles and their radius. The code generates a set of obstacles randomly placed in the entire search space. Certain coefficients such as Inertial weight coefficient, individual and social coefficients were chosen via trial and error method to determine the possible set of parameters which gives the best results. The entire program was prepared in Cpp and the data gathered was stored in a csv file. This was used to plot the data using python's matplotlib utility, to visualize the particles movement and the path generated.

Since our objective in path planning situation is to get the shortest path between the start and goal, we've taken the distance between two points as the fitness function. The requirement is to minimize this function.

Modifications applied following various State-of-the-art, Optimization improvements

The improvements in the path-planning algorithm implemented were as follows:

- Implemented Shortest Path Planning PSO (For better convergence)

  (Applies a slight perturbation to the Global best particles so that it is not stuck in local minima. The particles are updated by the given velocity formula)

double alpha = 0.1, beta = 0.2; // arbitrarily generated constants

double r3 = generate_random();

global_best_pos.velx = global_best_pos.x - particle_states[i].x + (alpha * global_best_pos.velx) + (beta * r3);

global_best_pos.vely = global_best_pos.y - particle_states[i].y + (alpha * global_best_pos.vely) + (beta * r3);

particle_states[i].x = particle_states[i].x + global_best_pos.velx;

particle_states[i].y = particle_states[i].y + global_best_pos.vely;

- Implemented "Intelligent Vehicle Global Path Planning Based on Improved Particle Swarm Optimization" (For better convergence, applying variable inertial weight)

  double c = 2.3;

  double wmax = 0.9, wmin = 0.4

  w = wmin + (wmax - wmin) * exp(-pow(((c * iter_ctr) / no_of_iters), 2.0));

- From “A Multi-Objective PSO-based Algorithm for Robot Path Planning”

  Refining fitness function and improving Path generation using two different fitness function.

  First OF: find shortest path from each newly generated particle to goal

  double spath = distance(particle_states[i], target)

  Second OF: find the smoothest path: the angle between the two hypothetical lines connecting the goal point to the robot’s two successive positions in each iteration, i.e. gbest[i] and gbest[i−1]

  double num = ((particle_states[i].x – target.x) * (global_pos[iter_ctr-1].x – target.x)) +

  ((particle_states[i].y – target.y) * (global_pos[iter_ctr-1].y – target.y))

  double den = sqrt(pow((particle_states[i].x – target.x), 2.0) + pow((particle_states[i].y – target.y), 2.0)) *

  sqrt(pow((global_pos[iter_ctr-1].x – target.x), 2.0) + pow((global_pos[iter_ctr-1].y– target.y), 2.0))

  double smooth = acos(num / den)

  double alpha1 = 1, alpha2 = 0.25

  double total_fitness = alpha1 * spath + alpha2 * smooth

  (Need to minimize the total fitness)

Obstacle Avoidance

For obstacle avoidance, we have used relocation method instead of penalty method for obstacle avoidance. In penalty method, we remove the particles falling inside an obstacle, or the path connecting new and old point passes through an obstacle. However, in relocation method, particles’ positions are relocated, so no particles are lost.

The algorithm part for relocation of particles’ positions is:

```
for i =1 to M, M is no. of obstacles do
    for j=1 to N, N is no. of particles do
        Find I, I is matrix of intersection points of
        obstacle and line connecting the new position
         of the particle to the robot's current position.
          while ( I ~= empty) do
             Find new position of the particle
              for k=1 to M do
                 Find I
                 if(I==empty )
                   Break
                 end if
              end for k
          end while
    end for j
end for i
```

Parameter values and Results

We have chosen a 2D square search space of 3000*3000-unit boundary. A wide range of parameters were used on a trial and error basis to determine the set of parameters giving the best results in terms of faster convergence, efficient obstacle avoidance, feasible path generation, avoiding local minima, preventing high overshoots, etc. The final parameters that balances a trade-off between all these criteria are the following:

- Starting coordinates taken: (1000, 1000)
- Goal coordinates taken: (-500, -1000)
- Number of obstacles taken: 7-10
- Swarm size: 100
- Number of max iterations: 200
- Inertial weight coefficients, wmin: 0.2, wmax: 0.7
- Local and Social coefficients, c1: 2, c2: 2
- Max allowed Velocity: width / 10
- Threshold to reach near goal to succeed: width/20
- Threshold to avoid near obstacles: radius of obstacle * 0.1

Simulation Results:

The above parameters along with the modifications implemented from various papers have resulted in faster convergence as can be seen from the console printed values for multiple runs.

Starting coordinates taken: (1000, 1000)

Goal coordinates taken: (-500, -1000)

```
Iteration :1.........
New Global Best fitness: 983.131
New Global Best Position: (232.027,-344.382)
Iteration :2.........
New Global Best fitness: 983.131
New Global Best Position: (232.027,-344.382)
Iteration :3.........
New Global Best fitness: 983.131
New Global Best Position: (232.027,-344.382)
Iteration :4.........
New Global Best fitness: 975.864
New Global Best Position: (189.749,-309.879)
Iteration :5.........
New Global Best fitness: 960.622
New Global Best Position: (116.245,-263.191)
Iteration :6.........
New Global Best fitness: 957.519
New Global Best Position: (179.034,-325.268)
Iteration :7.........
New Global Best fitness: 942.717
New Global Best Position: (100.002,-273.041)
Local Minima detected:
Iteration :8.........
New Global Best fitness: 639.705
New Global Best Position: (-101.486,-500.247)
Local Minima detected:
Iteration :9.........
New Global Best fitness: 364.861
New Global Best Position: (-284.023,-706.114)
Local Minima detected:
Iteration :10.........
New Global Best fitness: 187.207
New Global Best Position: (-403.306,-840.631)
Local Minima detected:
Iteration :11.........
New Global Best fitness: 48.2397
New Global Best Position: (-546.614,-1002.29)


Reached near goal. SUCCESS

Final Global Best Position: (-546.614,-1002.29)
Values written to csv file
Run Successful
```

Case: 1

```
Iteration :1.........
New Global Best fitness: 1339.96
New Global Best Position: (833.549,-872.195)
Iteration :2.........
New Global Best fitness: 627.126
New Global Best Position: (-1124.87,-989.451)
Iteration :3.........
New Global Best fitness: 471.484
New Global Best Position: (-967.636,-936.053)
Iteration :4.........
New Global Best fitness: 192.296
New Global Best Position: (-634.156,-863.526)
Iteration :5.........
New Global Best fitness: 178.587
New Global Best Position: (-518.146,-823.556)
Iteration :6.........
New Global Best fitness: 42.3198
New Global Best Position: (-505.271,-958.282)


Reached near goal. SUCCESS

Final Global Best Position: (-505.271,-958.282)
Values written to csv file
Run Successful
```

Case: 2

```
start or target in obstacle.....
Iteration :1.........
New Global Best fitness: 441.27
New Global Best Position: (-473.635,-1438.98)
Iteration :2.........
New Global Best fitness: 194.075
New Global Best Position: (-655.696,-887.296)
Iteration :3.........
New Global Best fitness: 55.4197
New Global Best Position: (-551.416,-1019.88)


Reached near goal. SUCCESS

Final Global Best Position: (-551.416,-1019.88)
Values written to csv file
Run Successful
```

Case: 3

```
start or target in obstacle.....
start or target in obstacle.....
Iteration :1.........
New Global Best fitness: 264.619
New Global Best Position: (-292.003,-1161.94)
Iteration :2.........
New Global Best fitness: 247.249
New Global Best Position: (-284.132,-1120.06)
Iteration :3.........
New Global Best fitness: 223.547
New Global Best Position: (-322.239,-1135.42)
Local Minima detected:
Iteration :4.........
New Global Best fitness: 164.529
New Global Best Position: (-436.869,-1151.81)
Iteration :5.........
New Global Best fitness: 78.3348
New Global Best Position: (-423.286,-1006.34)


Reached near goal. SUCCESS

Final Global Best Position: (-423.286,-1006.34)
Values written to csv file
Run Successful
```

Case: 4

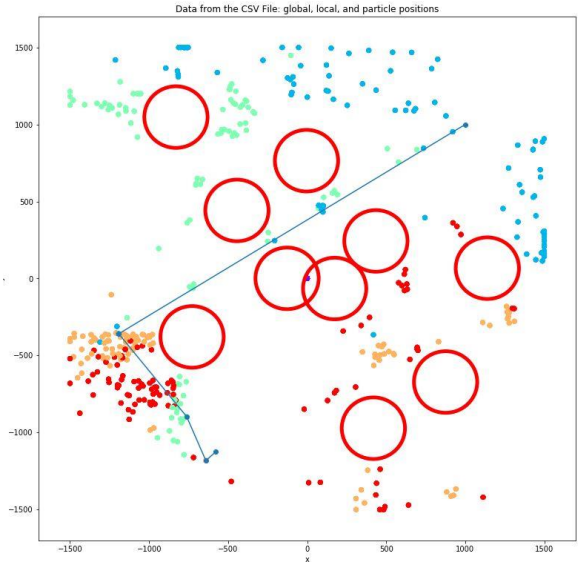
Graphical results::

Following graphs were plotted in python using the data gathered from the above.

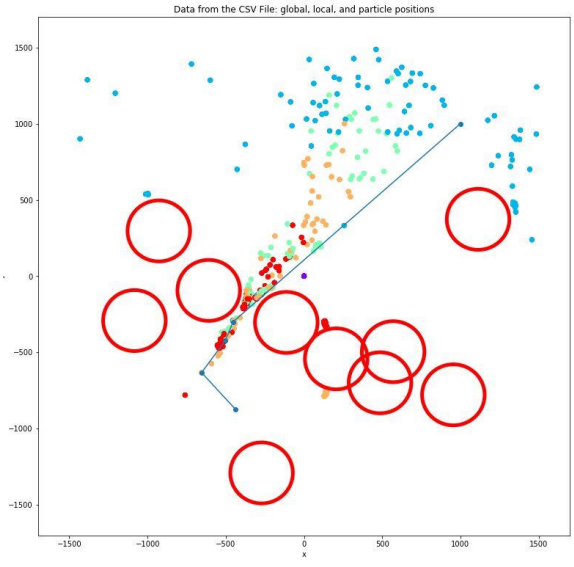
- Initial point: The point at the top right
- Final point: The point at the bottom left
- Obstacles: The red circles represent the obstacles in the area under consideration
- Final path: The blue line represents the final path from initial to final point.
- Color coding of particles: The colors of the particles are based on the iterations. With each iteration, the particles change color from blue (initial particles distribution) to red (final particles distribution)

Points to note:

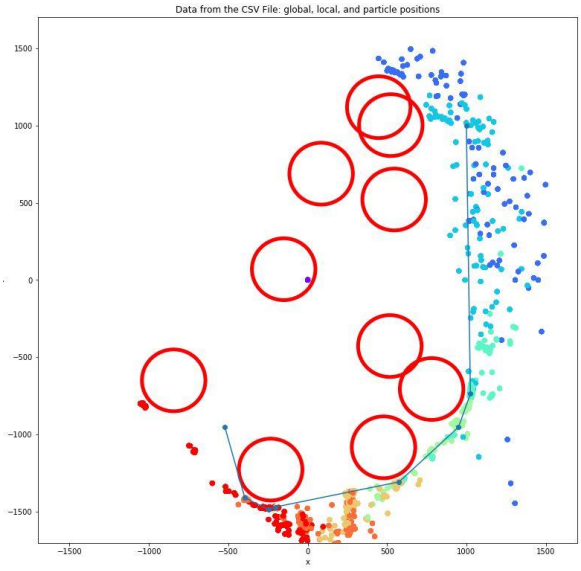
- As per the algorithm, none of the particles are placed inside the boundaries of the obstacles.
- Initial particles (blue in color) are randomly placed, thus are more scattered. The final particles (red in color) converge near the goal point.
- The algorithm finds out the least cost path even when the obstacles are very closely situated, thus proving the efficiency of the said method.



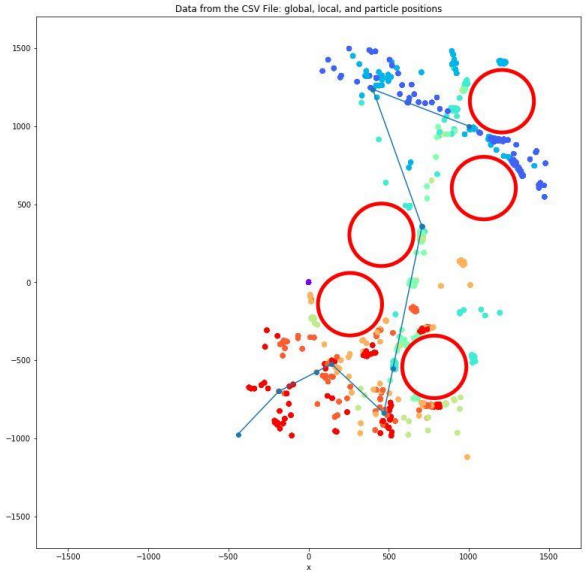
Case: 1



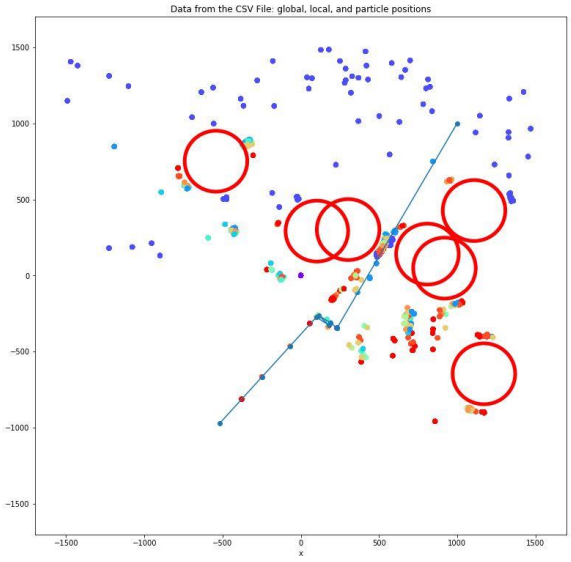
Case: 2



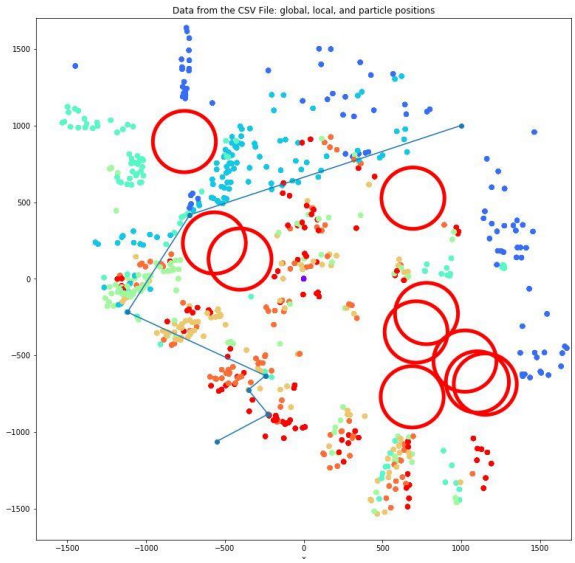
Case: 3



Case: 4



Case : 5



Case: 6

Inference

From the simulation results, we can verify that the implemented method is able to deal with the global planning requirements. The convergence is fast and takes very few iterations as can be observed from the console printed results. The obstacles were randomly placed in the environment and the algorithm avoided all of them. None of the generated particles is inside the obstacle boundaries. In few cases, the algorithm passed through narrow passages between bigger obstacles to reach the goal in shortest distance possible.

The method is easy to implement, the algorithm is very fast and works in cluttered and changing environments with moving obstacles. As demonstrated along this work, the method can perform in all types of environments. For now, the obstacles in the environment with a limited radius around it are detected. Other forms of obstacles can also be detected. It cannot be said with certainty that the path travelled by the robot to the global is optimum because the environment is dynamic and unknown. The method is flexible, that way we can change any parameters, or control the degree of importance of avoiding or moving toward the goal.