HAW HAMBURG

HAUSARBEIT
Thien Phuc Tran

Mobile Robot Path Planning in Dynamic Unknown Environments using Particle Swarm Optimization

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Betreuung durch: Prof. Dr. Michael Köhler-Bußmeier
Eingereicht am: 31. Aug 2021

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Contents

Mobile robot path planning is one of the most essential problems in robotics. This work proposes a novel PSO-based method for mobile robot local path planning in dynamic unknown environments. Using this method, the robot is able to accelerate or decelerate in order to avoid collision more efficiently. In addition, the proposed method defines a fitness function that incorporates multiple optimality criteria as well as the risk of collision. This work also experimentally showed that the proposed method outperforms a D* Lite baseline and can work in a multi-robot path planning setting. The differences in the behavior of the path planner with different prioritized objectives are observed.

Keywords: Particle Swarm Optimization, Local Path Planning.

1 Introduction

Robotics has been gaining popularity over the past decades. Mobile robots are nowadays widely used in various fields such as manufacturing, agriculture, military, and space or underwater exploration. Most applications require robots to navigate around the environment and accomplish tasks like gathering data about the surroundings or transporting a package. The ability to navigate from one place to another is trivial to humans. On the contrary, robot path planning, which finds a collision-free trajectory from the initial position to the goal, is one of the major problems in robotics and it still remains an active research topic.

Path planning in dynamic environments is categorized as an NP-hard problem [3]. The complexity of the problem rises when there are multiple objectives to optimize [9]. A robot path planning algorithm, in general, can be classified as either global (offline) path planning or local (online) path planning [16]. Global path planning can generate the optimal path in complex environments but requires complete knowledge about the environment as well as static obstacles. Unfortunately, the environment is often partially known or unknown to the robot in most real-world applications; i.e., the robot does not have information about the obstacles until they appear within its sensor range. Algorithms for global path planning become inefficient in dynamic environments due to the high cost of replanning. In contrast to global path planning, local path planning can

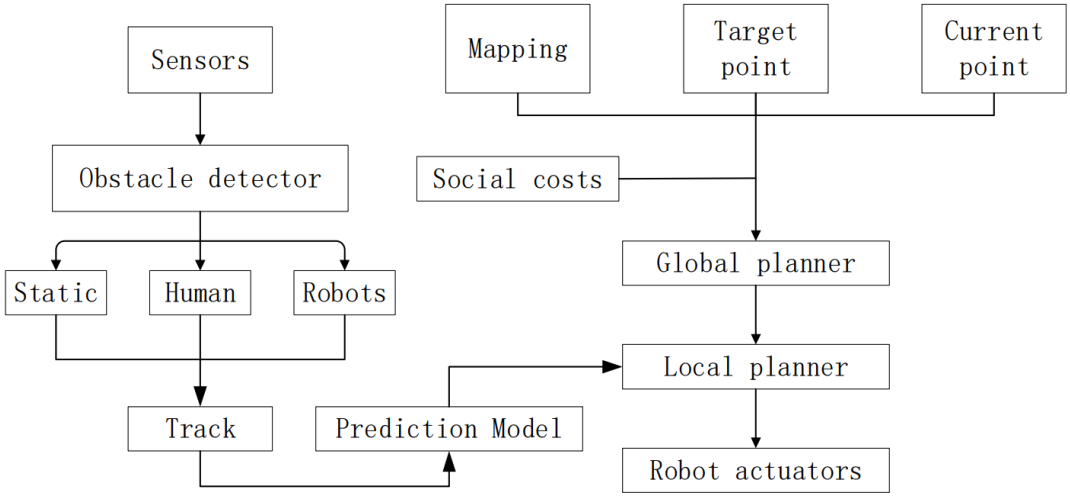


Figure 1: Overview of a robot navigation system [2].

deal with the dynamics and uncertainty of environments but may find suboptimal paths. Therefore, a robot navigation system often works in a hierarchical manner and consists of two major components: the global planner, which generates a global path (a set of waypoints) for the static map; i.e., without the dynamic obstacles, and a local planner, which guides the robot along the global path while taking the dynamics as well as motion constraints into consideration [2]. Figure 1 illustrates an example of such a system.

Roadmap methods for path planning such as Visibility Graph, Voronoi Diagram, Cell Decomposition methods generate a graph from the actual map. The generated graph can be used as the input for shortest path algorithms like Dijkstra, A* [15, 13]. However, these methods are not suitable for online path planning in dynamic or unknown environments, since the graph constantly needs to be re-generated and, thus, the path needs to be re-planned. D* Lite [12] was proposed to deal with dynamic environments more efficiently by reusing old information. In particular, D* Lite updates the graph only when changes are detected, and performs the search in the reverse direction, i.e., from the goal to the current position. Hence, the calculated distances remain unchanged when the robot moves. Because of this approach, D* Lite does not have significantly more advantages than A* when dealing with moving targets or obstacle-rich dynamic environments.

Potential Field methods [10] construct attractive and repulsive forces to form a force field (gradient) that gradually drives the robot away from the obstacles and towards the goal. The nature of potential field methods, however, leads to the problem of getting stuck

in local minima; i.e., the robot is trapped between obstacles and cannot find a way out. The problem becomes worse in obstacle-rich environments [22].

In contrast to classical methods that generate trajectories by strongly depending on prior knowledge, metaheuristic approaches generate a population of possible trajectories and improve them in each iteration [1]. These methods are designed to find good solutions for complex optimization problems. The solutions provided by metaheuristics methods are not guaranteed to be optimal. However, suboptimality is sufficient in most applications of mobile robot path planning, especially local path planning. Metaheuristic methods have been gaining popularity in the mobile robot navigation field of study thanks to their simplicity and effectiveness. Over the past decade, various metaheuristic optimization-based path planning methods were proposed, such as Genetic Algorithm (GA) [14], Cuckoo Search (CS) [17], and Particle Swarm Optimization (PSO) [24]. In [1], Ab Wahab et al. experimentally found that PSO with constriction factor performs better than other metaheuristic algorithms regarding path planning in dynamic unknown environments.

This work proposes a PSO-based local path planning method for mobile robots in dynamic unknown environments that takes the ability of robots to change movement speed into consideration. The proposed method also defined a fitness function for selecting a path that reduces the need for path re-planning, and thus, leads to performance improvement.

2 Related work

Dewang et al. [5] proposed a path planning method using Adaptive PSO. The inertia weight is varied based on iteration number. This results in exploration behavior in the early stage since the inertia weight is large. The exploration behavior is then slowly adjusted into an exploitation behavior, as the inertia weight decreases.

Girija and Joshi [6] proposed a hybrid method that combines PSO and Artificial Potential Field (APF) to improve convergence speed in obstacle-rich environments.

Hao et al. [7] use polar coordinates instead of Cartesian coordinates to express the turning angles. Phung and Ha [19] proposed Spherical Vector-based PSO that also uses polar coordinates to express the turning and climbing angles of UAV.

However, most path planning approaches attempt to find optimal paths only for the current state of the environment, i.e. dynamic obstacles and static obstacles do not make any difference to the robot. Moreover, the fact that the robot's movement speed can be controlled during navigation is neglected. In many cases, the robot can avoid obstacles just by accelerating, decelerating, or staying still for a short time. The method proposed in this work considers the ability to control movement speed in order to avoid collision more efficiently. In addition, the fitness of a path is evaluated partially based on the moving direction and movement speed of obstacles, so that future collision risk is included in the fitness value. Thus, the need for path re-planning can be reduced.

3 Particle Swarm Optimization

This section gives a brief overview of PSO and explains the common extensions that help to improve the convergence speed. Finally, Standard PSO-2011 (SPSO-2011) – one of the most popular variants of PSO – is introduced.

3.1 Canonical PSO

Inspired by bird flocks and fish schools, Kennedy and Eberhart [11] proposed in 1995 Particle Swarm Optimization, a population-based optimization technique, to solve non-linear functions. A particle in PSO, whose location represents a potential solution, moves in an n-dimensional problem space and searches for solutions while communicating with others. PSO creates a population of particles. Each particle is assigned an initial velocity, then the position of the particle is iteratively updated. In each iteration, the fitness function (or loss function) determines the fitness value of the particles. The particles, then, choose a new velocity based on their current velocity, their personal best position, and global best position. The velocity and position update can be expressed using the following equations:

$$
\begin{aligned}
v_i^{t+1} &= v_i^t + c_c r_c \left(p_i^t - x_i^t\right) + c_s r_s \left(g_i^t - x_i^t\right) \\
x_i^{t+1} &= x_i^t + v_i^{t+1}
\end{aligned}
$$

where x_i^t and v_i^{t+1} represent the position and velocity of the i-th particle at iteration t, respectively. p_i^t and g_i^t are the personal best and the global best position that are known to the i-th particle until the iteration t. c_c and c_s are acceleration coefficients that

control the influences of the global best (social component) and personal best (cognitive component). r_c and r_s are two numbers randomly drawn from a uniform distribution $U(0,1)$.

3.2 Inertia weight

Shi and Eberhart [20] proposed a modified version of PSO that uses inertia weight w. The inertia weight w controls the contribution of the previous velocity to the new one. Therefore, the equation to compute the new velocity for the i-th particle becomes:

$$v_i^{t+1} = wv_i^t + c_c r_c \left(p_i^t - x_i^t\right) + c_s r_s \left(g_i^t - x_i^t\right)$$

Inertia weight is a relevant parameter of PSO since it controls the trade-off between exploitation and exploration. This parameter can be adjusted after each iteration.

3.3 Constriction factor

In canonical PSO, the particles' inertia may cause the buildup of velocity, making them move towards infinity instead of converging to a solution. A trivial workaround is to implement an upper bound v_{max} for the velocity. However, Clerc and Kennedy [4] stated that v_{max} is not necessary. A proper constriction factor can prevent explosion and improve the convergence speed. The constriction factor χ can be computed using a parameter φ.

$$\begin{aligned}
\chi &= \frac{2}{\varphi - 2 + \sqrt{\varphi^2 - 4\varphi}} \\
v_i^{t+1} &= \chi \left(wv_i^t + c_c r_c \left(p_i^t - x_i^t\right) + c_s r_s \left(g_i^t - x_i^t\right)\right) \\
\varphi &= c_c + c_s
\end{aligned}$$

It was also shown by Clerc and Kennedy [4] that $\varphi > 4$ will result in quick and guaranteed convergence. Most implementations of PSO with constriction factor use equal values for c_c and c_s for the sake of simplicity. In the early version of Standard PSO [4], the values $\varphi = 4.1$ and $c_c = c_s = \frac{\varphi}{2} = 2.05$ were suggested. Since the effect of constriction factor is similar to that of inertia weight, the inertia w can be replaced by χ. Let $\phi = \frac{\varphi}{2}$, the

above equations can be simplified to:

$$\chi = \frac{1}{\phi - 1 + \sqrt{\phi^2 - 2\phi}}$$
$$v_i^{t+1} = wv_i^t + cr_c\left(p_i^t - x_i^t\right) + cr_s\left(g_i^t - x_i^t\right)$$
$$c = \chi\phi$$
$$w = \chi$$

In addition, as mentioned in the introduction, Ab Wahab et al. [1] experimentally showed that PSO with constriction factor outperformed other metaheuristic methods with respect to path planning in unknown environments.

3.4 Standard PSO-2011

The equations to update velocity above are applied dimension by dimension, causing the distribution of all next possible positions (DNPP) to lie in two hyperrectangles that are parallel to the axes, which also makes PSO biased [18, 21]. The Standard PSO-2011 (SPSO-2011) [23] proposed a new method to calculate the next velocity. The DNPP in SPSO-2011 resembles a hypersphere, which is rotation invariant.

SPSO-2011 defines a center of gravity G_i between the current position, a point a bit beyond p_i, and a point a bit beyond g_i:

$$G_i = \frac{x_i + (x_i + c(p_i - x_i)) + (x_i + c(g_i - x_i))}{3} = x_i + c\frac{p_i + g_i - 2x_i}{3}$$

G_i is then the center of a hypersphere $\mathcal{H}_i\left(G_i^t, \|G_i^t - x_i^t\|\right)$ with radius $\|G_i - x_i\|$. Let h_i be a random point in the hypersphere $\mathcal{H}_i$, the velocity is updated using the following equation:

$$v_i^{t+1} = wv_i^t + h_i^t - x_i^t$$

4 Method

The path planning problem must be firstly transformed into an optimization problem, namely a minimization problem. This work defines a fitness function (cost function) that expresses optimality of the path regarding multiple perspectives such as feasibility,

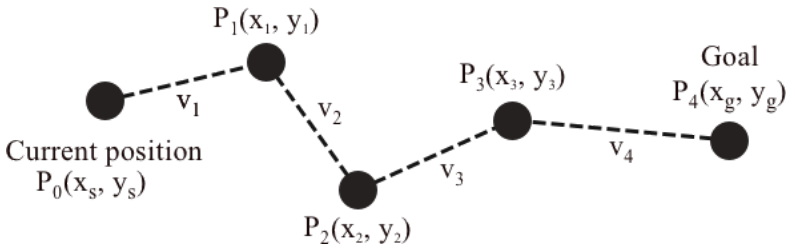


Figure 2: Illustration of a candidate solution with 3 intermediate waypoints.

safety, travel time, and travel distance. Then, SPSO-2011 is applied to minimize the fitness value. The path with the least fitness value will be chosen.

4.1 Problem formulation

A candidate path can be interpreted as a set of $N+2$ waypoints: the current position of the robot, N intermediate waypoints and the target. Since the current position and the target are fixed, the solution must contain only the coordinates of N waypoints.

There are $N+1$ segments between $N+2$ waypoints; the solution must contain $N+1$ movement speeds since the robot should be able to accelerate and decelerate while moving. Hence, the solution space for N intermediate waypoints in a 2D map is $3N+1$ -dimensional.

Let $P_{wi} = (x_{wi}, y_{wi})$ denote the i-th waypoint in a candidate path. $P_{w0} = (x_s, y_s) = (x_{w0}, y_{w0})$ is the current position of the robot, $P_{w,N+1} = (x_g, y_g) = (x_{w,N+1}, y_{w,N+1})$ is the target position. $s_i = \|\overrightarrow{P_{w,i-1}P_{wi}}\|$ denotes the length of the i-th segment. The robot has the movement speed $v_i \in (0, v_{max}]$. For the sake of simplicity, this work abuses the notation of vector slightly by using it only for expressing directional movements. The position X of a particle (a candidate solution) can be expressed as follows:

$$X = (x_{w1}, y_{w1}, x_{w2}, y_{w2}, , ..., x_{wN}, y_{wN}, v_1, v_2, ..., v_{N+1})$$

The fitness function in this work is defined as a weighted sum of multiple criteria,

$$L(X) = \sum_k w_k L_k(X); \quad k \in \{\text{distance, time, feasibility, safety}\}$$

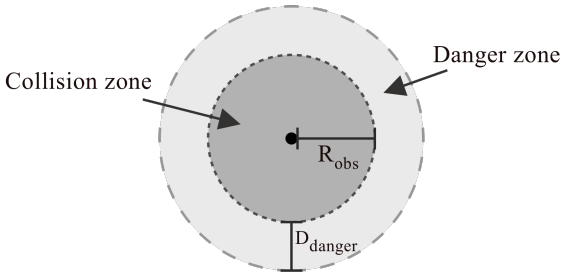


Figure 3: Illustration of an obstacle and the zones.

namely travel distance, travel time, safety and feasibility – which will be introduced in the next sub-sections.

4.2 Travel distance & travel time

The movement speed of the robot is assumed to be constant in most path planning methods; the travel time, hence, can be neglected. However, this work aims to develop the ability of the robot to accelerate or decelerate to avoid collision more efficiently. Therefore, shorter travel distance does not necessarily guarantee shorter travel time.

The travel distance is simply the total length of $N+1$ segments:

$$L_{distance}(X) = \sum_{i=1}^{N+1} \|\overrightarrow{P_{w,i-1}P_{wi}}\| = \sum_{i=1}^{N+1} s_i$$

Because the velocity remains constant within a segment, the total travel time can be calculated using the following equation:

$$L_{time}(X) = \sum_{i=1}^{N+1} \frac{\|\overrightarrow{P_{w,i-1}P_{wi}}\|}{v_i} = \sum_{i=1}^{N+1} \frac{s_i}{v_i}$$

4.3 Safety & feasibility

An obstacle in a 2D map is defined as a circle with radius R_{obs}. The approach of enclosing a polygonal obstacle by a circle is utilized in various local path planning algorithms. This approach helps to simplify the collision prediction and works efficiently in most mobile

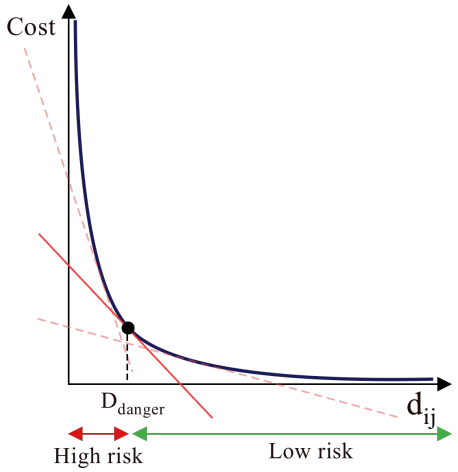


Figure 4: Illustration of the safety cost function.

robot navigation applications. Likewise, the robot is also enclosed by a circle of radius S.

This work assumes that the movement of obstacles is linear, i.e., the velocity is constant. However, sensors are often imperfect and the velocity of obstacles may change at any time in real-world applications. Thus, a danger distance D_{danger} is defined to keep the robot from the obstacles at a distance, which makes sure the robot has enough time to react upon unexpected dynamics (see Figure 3).

Assuming a set of M obstacles are detected by the sensors, let $P_{oj} = (x_{oj}, y_{oj})$ denote an obstacle j at the coordinate (x_{oj}, y_{oj}) with a velocity $\overrightarrow{v_{oi}} = (\Delta x_{oj}, \Delta y_{oj})$. The robot moves with a velocity of $\overrightarrow{v_{ri}} = v_i \cdot \frac{\overrightarrow{p_{i-1}p_i}}{\|\overrightarrow{p_{i-1}p_i}\|}$ within the i-th segment. The distance between the robot and the obstacle can be easily calculated after shifting the origin of the coordinate system to the robot's position. After shifting the origin, the position of the obstacle is $P'_{oj} = P_{oj} - P_{wi}$ and the velocity vector becomes $\overrightarrow{v'_{oj}} = \overrightarrow{v_{oj}} - \overrightarrow{v_{ri}}$. As the robot moves from the waypoint $P_{w,i-1}$ to the next waypoint P_{wi} in $t_i = \frac{s_i}{v_i}$ time units, the relative position of the obstacle also moves from P'_{oj} to $P'_{oj} + t_i \cdot \overrightarrow{v'_{oi}}$. The mininum distance d_{ij} between the obstacle j and the robot during the i-th segment is simply the distance between the origin $O = (0, 0)$ and the line segment $\left[P'_{oj}, P'_{oj} + t_i \cdot \overrightarrow{v'_{oi}}\right]$ minus their sizes.

The feasibility cost is infinity when the robot collides with any obstacle along the path, and 0 otherwise:

$$L_{feasibility}(X) = \sum_{i=1}^{N+1} \sum_{j=1}^{M} C_{ij}$$

$$C_{ij} = \begin{cases} \infty, & d_{ij} < 0. \\ 0, & \text{otherwise.} \end{cases}$$

$$d_{ij} = d(O, \left[P'_{oj}, P'_{oj} + t_i \cdot \overrightarrow{v'_{oi}}\right]) - R_{obs,i} - S$$

$$P'_{oj} = P_{oj} - P_{wi}$$

$$\overrightarrow{v'_{oj}} = \overrightarrow{v_{oj}} - \overrightarrow{v_{ri}}$$

Finally, the safety cost aims to keep the robot from the obstacle by a safe distance D_{danger}:

$$L_{safety}(X) = \sum_{i=1}^{N+1} \sum_{j=1}^{M} \frac{D_{danger}^2}{d_{ij}}$$

The function $f(x) = \frac{D_{danger}^2}{x}$ has the derivative $f'(x) = -\frac{D_{danger}^2}{x^2}$, which returns the value -1 when the distance is equal to D_{danger}. The closer the robot gets to the obstacle, the higher the risk of collision, and the steeper the slope of the function. As a result, the cost increases drastically when the robot approaches the danger zone and keeps moving towards the obstacle (see Figure 4).

4.4 Polar coordinates

Some robot types can only turn at a max angle φ_{max}. This work utilizes polar coordinates to natively express the turning angle of the robot. A candidate solution is expressed as follows:

$$X = (\varphi_{w1}, r_{w1}, \varphi_{w2}, r_{w2}, , ..., \varphi_{wN}, r_{wN}, v_1, v_2, ..., v_{N+1}),$$

where φ_{wi} and r_{wi} describe the turning angle and the segment length for the robot to move forward, respectively. This approach easily takes the motion constraints of the robot into consideration by setting the lower bound and upper bound for φ_i in the search space, namely $\varphi_i \in [-\varphi_{max}, \varphi_{max}]$. Moreover, the utilization of polar coordinates for path planning reportedly improves the convergence speed and the path quality [8, 19].

In order to use the previously defined cost function, the polar coordinates must be converted back to Cartesian coordinates. Let ϕ_r denote the current angle between the robot's

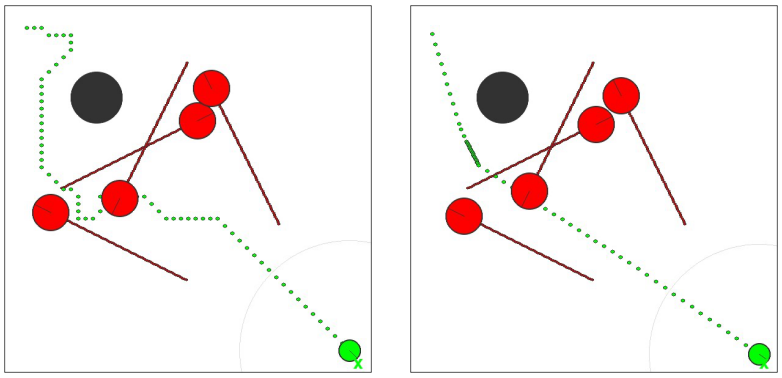
velocity and the x-axis at the beginning of the i-th segment, the waypoint P_{wi} can be calculated as follows:

$$\overrightarrow{\Delta p_i} = (cos\varphi_i, sin\varphi_i)$$
$$(\overrightarrow{\Delta p_i'})^T = \begin{bmatrix} cos\phi_r & -sin\phi_r \\ sin\phi_r & cos\phi_r \end{bmatrix} \overrightarrow{\Delta p_i}^T$$
$$P_{wi} = P_{w,i-1} + \overrightarrow{\Delta p_i'}$$

This approach, however, does not take the last turning angle φ_{N+1} into consideration. Therefore, φ_{N+1} must be calculated using Cartesian coordinates and punished by infinite cost in the case of $\varphi_{N+1} > \varphi_{max}$.

5 Experiments

A simulation is built in this work to validate the performance of the proposed method. The method is tested with different optimality criteria in order to observe the differences in movement behavior. In addition, the Cartesian coordinate-variant is compared to the polar coordinate-variant in order to confirm the effectiveness of using polar coordinates. In the simulation, the dynamic obstacles are marked in red, and the static obstacles are marked in black. The robot is the green circle, and the thin gray circle is the robot's sensor range. The short black line inside the circle depicts the facing direction. After each time step, the robot and the obstacles leave a small dot at their current position. This helps to visualize the traveled path as well as the movement speed of the robot. The robot's goal is the green **X** symbol.



(a) D* Lite (b) Proposed method

Figure 5: Comparison between the paths generated by D* Lite and the proposed method.

The outcome of the proposed method is firstly compared to a baseline – a grid-based D* Lite implementation. Using D* Lite, the simulation took about 1m49s to complete the scenario in Figure 5. The robot traveled a total distance of 242.75m in 108s. Meanwhile, the simulation took solely **18s** to finish using the proposed method (3 intermediate waypoints, 30 particles, 1000 iterations). The robot traveled a total distance of **134.8m** in **66s**. It can be observed that the limitation of sensor range causes D* Lite to make redundant movements.

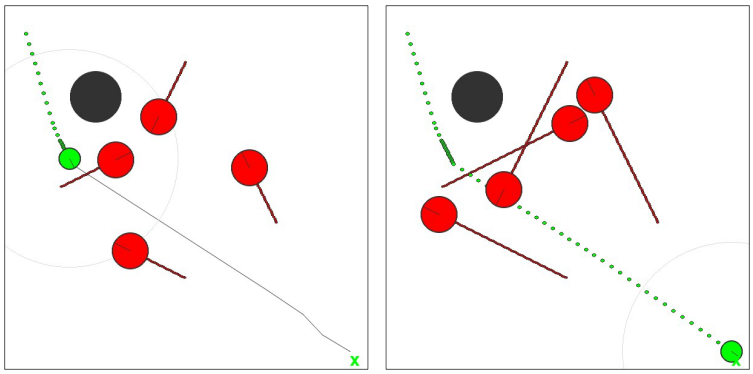
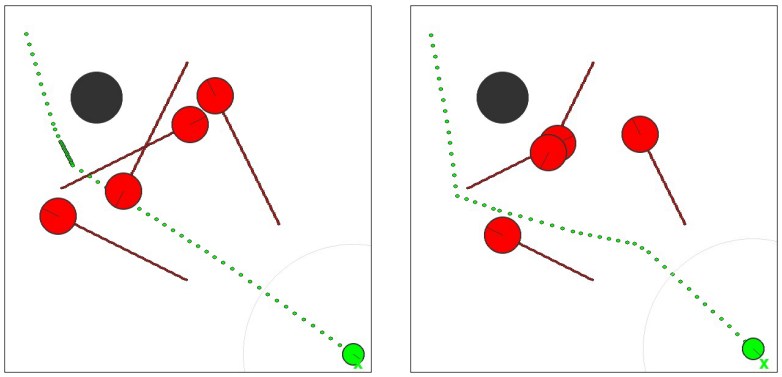


Figure 6: The robot can slow down to avoid the obstacle efficiently.



(a) Prioritizing travel distance (b) Prioritizing travel time

Figure 7: Comparison between the prioritized travel distance setting (a) and and the prioritized travel time settings (b).

As shown in Figure 6, the robot can slow down for a short time when it's behind an obstacle. After the obstacle moved out of its trajectory, the robot accelerates and gets to the target quickly.

The weight parameters $w_{distance}$ and w_{time} define the balance between the travel distance and the travel time. When $w_{distance} > w_{time}$, the robot prefers to slow down and wait

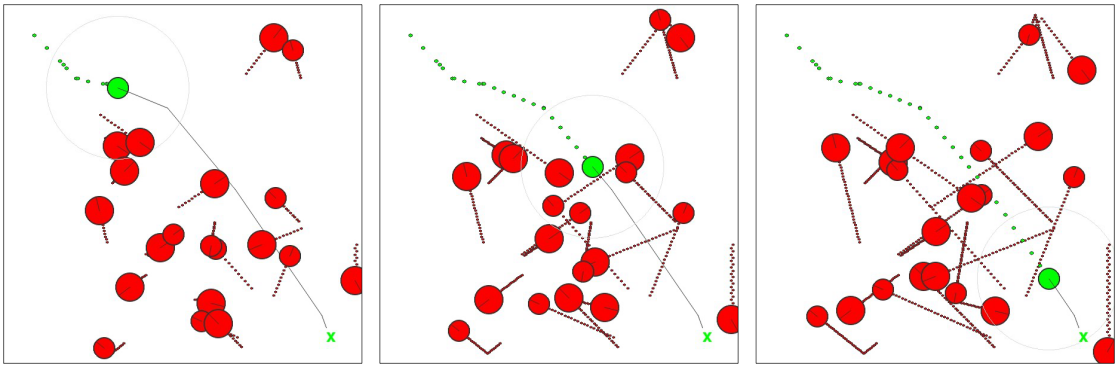
for the obstacle to move further. This results to shorter travel distance, but longer travel time (see Figure 7a). On the other hand, the robot prefers to accelerate and move around the obstacle when $w_{distance} < w_{time}$, resulting to longer travel distance but shorter travel time (see Figure 7b).

Prioritized objective	Cartesian PSO				Polar PSO			
	Travel distance		Travel time		Travel distance		Travel time	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Travel distance	136.64	3.78	93.40	24.67	**132.40**	1.34	**72.80**	5.76
Travel time	153.02	17.29	**56.60**	6.69	**152.17**	6.84	56.80	3.56

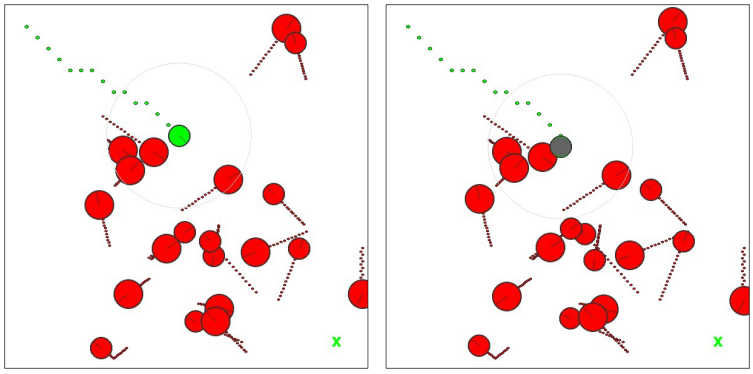
Table 1: Comparision between Cartesian PSO and Polar PSO with different prioritized objective.

Table 1 compares the variant that uses Cartesian coordinates (Cartesian PSO) to the variant with polar coordinates (Polar PSO). Polar PSO outperforms Cartesian PSO by a large margin in the prioritized travel distance setting. However, Cartesian PSO performs slightly better in the other setting. The author hypothesizes that the nature of the two search spaces causes them to behave differently. Cartesian PSO uses the coordinates of intermediate waypoints and tends to converge to big and sudden movements; while Polar PSO uses turning angles of the robot and segment lengths and tends to converge to smoother movements.

The proposed method is then compared to D* Lite in an obstacle-rich environment. The robot successfully avoided the obstacles and reached the target using the proposed method. D* Lite failed to foresee the position of the obstacle at the next time step and crashed into the obstacle (Figure 8), since such classical methods consider the current state of the environment as static.



(a) The robot can navigate through a crowd of obstacles.



(b) Using D* Lite, the robot failed to foresee the collision in the next time step.

Figure 8: Comparison between the proposed method and D* Lite in an obstacle-rich environment.

6 Conclusion

This work proposes a novel local path planning method that uses PSO for path optimization. The robot successfully avoids collision by decelerating and accelerating, resulting in shorter travel distance or travel time in comparison to other methods. In addition, collision risk is taken into consideration in order to construct better paths and avoid re-planning. It was experimentally showed that the proposed method outperforms D* Lite by a large margin in dynamic, obstacle-rich, and unknown environments.

Polar PSO outperforms Cartesian PSO significantly when prioritizing travel distance more than travel time. In the prioritized travel time setting, Cartesian PSO performs slightly better. The author hypothesizes that the differences in nature of the two search spaces cause the particles to search in different manners. Searching by directly changing the waypoint's coordinates may tend to converge to big and sudden movements

while modifying turning angles and segment lengths tend to converge to smoother movements.

Future work is to test the algorithm in real-world applications. Furthermore, the author aims to generalize the proposed method to solve the problem in a 3D space, so that it can incorporate elevation data and be utilized for UAV path planning.

Technical specifications

The implementation of the simulation in the experiments uses Python 3.6. The source code of the simulation can be found in https://git.haw-hamburg.de/acf530/pso-pathplanning. The experiments ran on a Macbook Pro with the following specifications:

Model: Macbook Pro 2017

Processor: 2.8 GHz Quad-Core Intel Core i7

Memory: 16 GiB 2133 MHz

OS: MacOS Big Sur 11.4

References

[1] Ab Wahab, Mohd N. ; Nefti-Meziani, Samia ; Atyabi, Adham: A comparative review on mobile robot path planning: Classical or meta-heuristic methods? In: *Annual Reviews in Control* (2020)

[2] Cai, Kuanqi ; Wang, Chaoqun ; Cheng, Jiyu ; De Silva, Clarence W. ; Meng, Max Q-H: Mobile robot path planning in dynamic environments: a survey. In: *arXiv preprint arXiv:2006.14195* (2020)

[3] Canny, John ; Reif, John: New lower bound techniques for robot motion planning problems. In: *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)* IEEE (Veranst.), 1987, S. 49–60

[4] Clerc, Maurice ; Kennedy, James: The particle swarm-explosion, stability, and convergence in a multidimensional complex space. In: *IEEE transactions on Evolutionary Computation* 6 (2002), Nr. 1, S. 58–73

[5] Dewang, Harshal S. ; Mohanty, Prases K. ; Kundu, Shubhasri: A Robust Path Planning For Mobile Robot Using Smart Particle Swarm Optimization. In: *Procedia Computer Science* 133 (2018), S. 290–297. – URL https://www.sciencedirect.com/science/article/pii/S1877050918309815. – International Conference on Robotics and Smart Manufacturing (RoSMa2018). – ISSN 1877-0509

[6] Girija, S ; Joshi, Ashok: Fast Hybrid PSO-APF Algorithm for Path Planning in Obstacle Rich Environment. In: *IFAC-PapersOnLine* 52 (2019), Nr. 29, S. 25–30. – URL https://www.sciencedirect.com/science/article/pii/S2405896319325583. – 13th IFAC Workshop on Adaptive and Learning Control Systems ALCOS 2019. – ISSN 2405-8963

[7] Hao, Yanling ; Zu, Wei ; Zhao, Yuxin: Real-Time Obstacle Avoidance Method based on Polar Coordination Particle Swarm Optimization in Dynamic Environment. In: *2007 2nd IEEE Conference on Industrial Electronics and Applications*, 2007, S. 1612–1617

[8] Hao, Yanling ; Zu, Wei ; Zhao, Yuxin: Real-time obstacle avoidance method based on polar coordination particle swarm optimization in dynamic environment. In: *2007 2nd IEEE conference on industrial electronics and applications* IEEE (Veranst.), 2007, S. 1612–1617

[9] Hussein, Ahmed ; Mostafa, Heba ; Badrel-din, Mohamed ; Sultan, Osama ; Khamis, Alaa: Metaheuristic optimization approach to mobile robot path planning. In: *2012 international conference on engineering and technology (ICET)* IEEE (Veranst.), 2012, S. 1–6

[10] Hwang, Yong K. ; Ahuja, Narendra u. a.: A potential field approach to path planning. In: *IEEE Transactions on Robotics and Automation* 8 (1992), Nr. 1, S. 23–32

[11] Kennedy, James ; Eberhart, Russell: Particle swarm optimization. In: *Proceedings of ICNN'95-international conference on neural networks* Bd. 4 IEEE (Veranst.), 1995, S. 1942–1948

[12] Koenig, Sven ; Likhachev, Maxim: D^* lite. In: *Aaai/iaai* 15 (2002)

[13] Kwak, Jeonghoon ; Sung, Yunsick: Autonomous UAV Flight Control for GPS-Based Navigation. In: *IEEE Access* 6 (2018), S. 37947–37955

[14] Lamini, Chaymaa ; Benhlima, Said ; Elbekri, Ali: Genetic algorithm based approach for autonomous mobile robot path planning. In: *Procedia Computer Science* 127 (2018), S. 180–189

[15] Li, Jianqiang ; Deng, Genqiang ; Luo, Chengwen ; Lin, Qiuzhen ; Yan, Qiao ; Ming, Zhong: A Hybrid Path Planning Method in Unmanned Air/Ground Vehicle (UAV/UGV) Cooperative Systems. In: *IEEE Transactions on Vehicular Technology* 65 (2016), Nr. 12, S. 9585–9596

[16] Mohanty, Prases K. ; Parhi, Dayal R.: Controlling the motion of an autonomous mobile robot using various techniques: a review. In: *Journal of Advance Mechanical Engineering* 1 (2013), Nr. 1, S. 24–39

[17] Mohanty, Prases K. ; Parhi, Dayal R.: Optimal path planning for a mobile robot using cuckoo search algorithm. In: *Journal of Experimental & Theoretical Artificial Intelligence* 28 (2016), Nr. 1-2, S. 35–52

[18] Monson, Christopher K. ; Seppi, Kevin D.: Exposing origin-seeking bias in PSO. In: *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, 2005, S. 241–248

[19] Phung, Manh D. ; Ha, Quang P.: Safety-enhanced UAV path planning with spherical vector-based particle swarm optimization. In: *Applied Soft Computing* 107 (2021), S. 107376

[20] Shi, Yuhui ; Eberhart, Russell: A modified particle swarm optimizer. In: *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)* IEEE (Veranst.), 1998, S. 69–73

[21] Spears, William M. ; Green, Derek T. ; Spears, Diana F.: Biases in particle swarm optimization. In: *Innovations and developments of swarm intelligence applications*. IGI Global, 2012, S. 20–43

[22] Warren, C.W.: Multiple robot path coordination using artificial potential fields. In: *Proceedings., IEEE International Conference on Robotics and Automation*, 1990, S. 500–505 vol.1

[23] Zambrano-Bigiarini, Mauricio ; Clerc, Maurice ; Rojas, Rodrigo: Standard particle swarm optimisation 2011 at cec-2013: A baseline for future pso improvements. In: *2013 IEEE congress on evolutionary computation* IEEE (Veranst.), 2013, S. 2337–2344

[24] Zhang, Yong ; Gong, Dun-wei ; Zhang, Jian-hua: Robot path planning in uncertain environment using multi-objective particle swarm optimization. In: *Neurocomputing* 103 (2013), S. 172–185

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original