JAISCR, 2019, Vol. 9, No. 4, pp. 267 – 282
10.2478/jaiscr-2019-0008

COLLISION-FREE AUTONOMOUS ROBOT NAVIGATION IN UNKNOWN ENVIRONMENTS UTILIZING PSO FOR PATH PLANNING

Evan Krell*, Alaa Sheta, Arun Prassanth Ramaswamy Balasubramanian, Scott A. King

Department of Computing Sciences, Texas A&M University-Corpus Christi,
Corpus Christi, TX 78412, USA
**E-mail: ekrell@islander.tamucc.edu*

Submitted: 17th June 2018; Accepted: 12th May 2019

Abstract

The autonomous navigation of robots in unknown environments is a challenge since it needs the integration of a several subsystems to implement different functionality. It needs drawing a map of the environment, robot map localization, motion planning or path following, implementing the path in real-world, and many others; all have to be implemented simultaneously. Thus, the development of autonomous robot navigation (ARN) problem is essential for the growth of the robotics field of research. In this paper, we present a simulation of a swarm intelligence method is known as Particle Swarm Optimization (PSO) to develop an ARN system that can navigate in an unknown environment, reaching a pre-defined goal and become collision-free. The proposed system is built such that each subsystem manipulates a specific task which integrated to achieve the robot mission. PSO is used to optimize the robot path by providing several waypoints that minimize the robot traveling distance. The *Gazebo* simulator was used to test the response of the system under various envirvector representing a solution to the optimization problem.onmental conditions. The proposed ARN system maintained robust navigation and avoided the obstacles in different unknown environments. vector representing a solution to the optimization problem.

Keywords: mobile robot, particle swarm optimization, path planning

1 Introduction

An autonomous robot is a physical agent that interacts with its environment to independently fulfill a mission [11]. It should be able to follow a path from a start to a goal without collision with obstacles in an unknown environment. Autonomous robots perform tasks in unstructured environments and choose their independent decisions as a function of the given goal, such as in package delivery, cleaning, autonomous car, surveillance, and search and rescue operations. The tendency is to provide robots with extra autonomy, which means that a robot can accomplish missions with little human assistance.

Ghosh et al. [14] compared two metaheuristic algorithms: bat algorithm and flower pollination algorithm (FPA). FPA was shown to perform better both in path distance and path following execution time. Unlike the vast majority of metaheuristic path planning research, real-time experiments were implemented using ARDUINO-based robots. In [8], the authors presented a real robot navigation system for path following implemented using GA and artificial bee colony algorithm. Experiments were done where the robot is allowed diagonal movements that come extremely close to the walls and corners. GAs shows better results in this case. In 2006, Chen and Li [9] modified the typical path planning formulation to emphasize smooth paths. They used a multi-objective fitness function considering both distance and smoothness as two objectives. Paths are generated as higher-order polynomials instead of a sequence of coordinate points. The experiments were done in a simulated 2D environment with static obstacles.

Sheta et. al. [4] demonstrate a system where GA path planning is applied to static environments. Using Euclidean distance, the shortest path to the goal is often touching a corner, which can be challenging to maneuver. Thus, the robot was restricted to row-wise and column-wise movements to avoid dangerous diagonals. This sacrifices optimality but ensures feasible paths that the robot can traverse. A similar approach [24] with restricted grid movement uses movement rules suited for multi-directional movement for complex environments, and the objective of the fitness function is to minimize the distance as well as the number of turns.

In [3], the authors presented a mobile robot navigation system that can deal with a hard task which requires avoiding both static and dynamic objects. A local map was initially made using a mounted ultrasonic sensor. This map offers the position of the nearest obstacles in the scene. An initial path is created for the robot navigation based on the initial local map. As soon as the robot starts its trajectory in real indoor environments with obstacles, the sensor continuously senses and updates the occupancy map.

A real robot system was shown in 2016 [1]. The author presented a hybrid algorithm of PSO and Gravitational Search Algorithm (GSA) was used, where the fitness function includes the fitness values of both algorithms. With the combined fitness, the authors aimed to take advantage of the exploitation of PSO and the exploration of GSA. The focus was on improving the solutions rather than handling complex environments, so planning used a global map of the static environment. Multiple robots were handled by a centralized solver.

Dynamic obstacles were also addressed by Raja and Pugazhenthi [21]. Instead of reacting to unexpected obstacles, dynamic obstacle movement is incorporated into the path planning problem. This necessitates that the future trajectories of the obstacles are known. Also, each obstacle's shape is simplified by representing it with a circle. A similar approach was considered in [19] where dynamic obstacles move with probability functions. All obstacles were also presented as circles. In [18], a hybrid planning scheme was presented. In this study, PSO was used to generate the overall path, while probabilistic roadmaps (PRM) was used to refine the path around obstacles. A real-world robot demonstrated the effectiveness of the proposed planning schema.

Dynamic obstacles were addressed in 2017 by replanning whenever collision risks were detected [7]. The robot had a global view of the environment, but with uncertainty in the obstacle positions. The high cost of planning limits its effectiveness for reactive planning. To overcome this, a vectorized implementation of PSO was implemented that was demonstrated to be significantly more efficient by avoiding the loops used in the typical evaluation of the fitness function. Multiple agents were handled in the decentralized system by considering other agents as dynamic obstacles.

The goal of this research is two-fold.

- **Map-building problem:** Navigating through an environment required a map. The map can be used to locate both the start and goal positions. It can also be used to design a path between the robot's position and the goal position. Unfortunately, in many cases, this map does not exist early before navigation. Consequently, the robot must create its particular map in the unknown unstructured environment. Therefore, we must be able to both locate the robot and make the map at the same time. We plan to solve this problem via the use of LIDAR sensor. The simulated robot has *odometer* for self-localization and a *laser scanner* for detecting the range from the robot to obstacles within the scanner's radius. As the robot moves, the laser sensor scans the environment. The created map needs to be sufficient to avoid navigation errors.
- **Path Planning in unknown environments:** This problem is significant to solve because it disturbs the optimality of the path length and extensiveness of navigation task (i.e., approaching

the goal position when a solution exists). We present a method to select the optimal path based on a set of waypoints selected by PSO. These waypoints are optimized such that they minimize the distance the robot has to travel from a start to end point.

The system proposed here falls in between the 2D simulated experiments and those with real robots. A 3D physics-based environment is used for simulation. This means that the generated paths must be feasible given the physical dimensions and dynamics of the robot. Even systems using real robots often do so with a given map. The simulated robot here has initially no knowledge of the static environment and builds a map using a laser sensor while exploring before the path planning process. Emphasis is on ensuring that the robot can operate safely in complex indoor rooms with narrow passages and concave shapes.

This paper is organized as follows. In Section 2, PSO is described. Section 3 presents an overview of the proposed autonomous robot navigation system. Section 4 describes the experimental environment configuration, followed by experimental results in Section 5. Finally, conclusions and future work are given in Section 6.

2 Particle Swarm Optimization

PSO is a biologically-inspired metaheuristic optimization technique inspired by colony behavior in nature [2]. We use a PSO library written by Kyriakos Kentzoglanakis [16]. The library allows a programmer to write a fitness function that will be optimized by using the PSO algorithm.

PSO implements a single group of social particles that use their collective information to find the best resources. The environment that the particles explore is the search space of the problem, and a fitness function measures the desirability of a point. Particles are described by a position and a velocity which are adjusted over the iterations. These properties can be improved in two possible ways, one of which is by observing the best of the entire population. The best location so far is broadcast to all members. The second means of learning is keeping track of its history. To promote exploring the space and avoiding premature convergence, the particles are influenced by the global best, personal best and some randomization. In each iteration, the particles should move closer to the optimal solution.

The process is complete when the particles converge or reach a time or iteration limit. Particles exchange information with their neighborhood particles to memorize their best collective position. Using these properties, each particle updates its position. Equations 1 and 2 are used for updating the i_{th} particle in the k_{th} iteration, described by velocity V_i^k and position X_i^k. Each particle is an n-dimensional vector representing a solution to the optimization problem.

$$\begin{aligned} V_i^k = & wV_i^{k-1} + c_1R_1(B_i^{k-1} - X_i^{k-1}) \\ & + c_2R_2(G_i^{k-1} - X_i^{k-1}). \end{aligned} \tag{1}$$

$$X_i^k = X_i^{k-1} + V_i^k, \tag{2}$$

where:

- V_i^k: n-dimensional velocity i at iteration k
- X_i^k: n-dimensional position i at iteration k
- w: inertia weight
- c_1, c_2: cognitive and social coefficients
- R_1, R_2: n-dimensional random numbers
- B_i^k: local best position at iteration k
- G_i^k: global best position reached by the neighborhood of particle i at iteration k

Algorithm 1: PSO Algorithm

```
1  Begin PSO
2    Randomly initialize the position and
     velocity of the particles: $X_i^0, V_i^0$
3    While (Not terminating condition) do
4        For i = 1 to number of particles
5          Evaluate the fitness:= $f(X_i^{k-1})$
6          Update $B_i^k$ and $G_i^k$
7          Update velocity $V_i^{k-1}$
8          Update position $X_i^{k-1}$
9        Next for
10       Update inertia weight $w$
11   End while
12 end PSO
```

Swarm size is the number of particles, each of which has its own personal best solution. c_1 and c_2 are constant coefficients that weight the influence of the local and global search, respectively. The inertia weight w is used to balance the trade-off between convergence and exploration [5]. Initially, local search is emphasized by setting a w to w_{max}. The value is linearly decreasing with each PSO iteration to eventually reach w_{min} that emphasizes global search to promote convergence. The PSO algorithm is given in Algorithm 1.

3 Proposed Robot System

In this Section, we present the high level architecture of the proposed ARN system. The system consists of a several phases/components each of which is responsible for implementing a specific function. The integration of these set of functions produce a layout of the system (See Figure 1).

3.1 Mapping Unknown Environment Using LIDAR

It is assumed that the environment is initially unknown and needs to be mapped before path planning. The simulated robot has an *odometer* for self-localization and a *laser scanner* (LIDAR) for detecting the range from the robot to obstacles within the scanner's radius. As the robot moves, the laser sensor scans the environment. The readings continually update an initially empty occupancy grid. The mobile robot system has to fully explore the space to cover the unmapped areas autonomously.

The TURTLEBOT robot is allowed to move freely within the environment and generates a map while avoiding obstacles. If the laser scanner detects an obstacle ahead, the TURTLEBOT is programmed to turn away in a random direction and moves forward. The goal of this strategy of exploration, random walk, is to avoid the problem of making the robot move in circles which is common with wall-following approaches. Though not prioritizing efficiency, it manages to generate maps without any *apriori* knowledge of the environment and with a very low computational footprint. This strategy is suitable for indoor environments that do not experience frequent changes, because the additional time of exploration, compared to slightly more sophisticated exploration schemes such as Closest Frontier is dwarfed by the duration during which the map remains accurate.

As was shown, the random walk is an effective for multi-robot exploration. A single added robot is likely to speed up exploration significantly, whereas robots using the same less naive method tend to make the same decisions which results in much exploration overlap. The trend is toward multiple robots operating in the same environment [12]. That random walk approaches the efficiency of more informed algorithms as the number of robots increases is demonstrated by Damer et al. [26].

3.2 Path Planning Using PSO

In this Section, we shall provide an overview of how we shall utilize PSO as a search algorithm to find the shortest path from a given start to the target point in the unknown environment. To generate this optimal path, PSO is used to pick the best multiple waypoint path that minimizes the distance the robot travels.

Path planning is the process of generating a sequence of action directives so that the robot can reach a target destination. Path planning has remained a challenge, despite numerous proposed techniques, since it is NP-hard. Metaheuristic algorithms have been demonstrated to drastically reduce the search space when finding the optimal path [20, 22].

The proposed solution path is presented as a sequence of waypoints, as shown in Figure 2. The path creation shall depend on the number of waypoints selected by the designer of the system. Too many waypoints might help PSO to find this optimal path but at the same time might slow the convergence process to such an optimal solution. Thus, our goal is to set up a balance between the two goals. We should always remember that our goal is to reduce the distance the robot travels in the environment. The proposed PSO representation shall be a sequence of waypoints P, demonstrated in Equation 3.

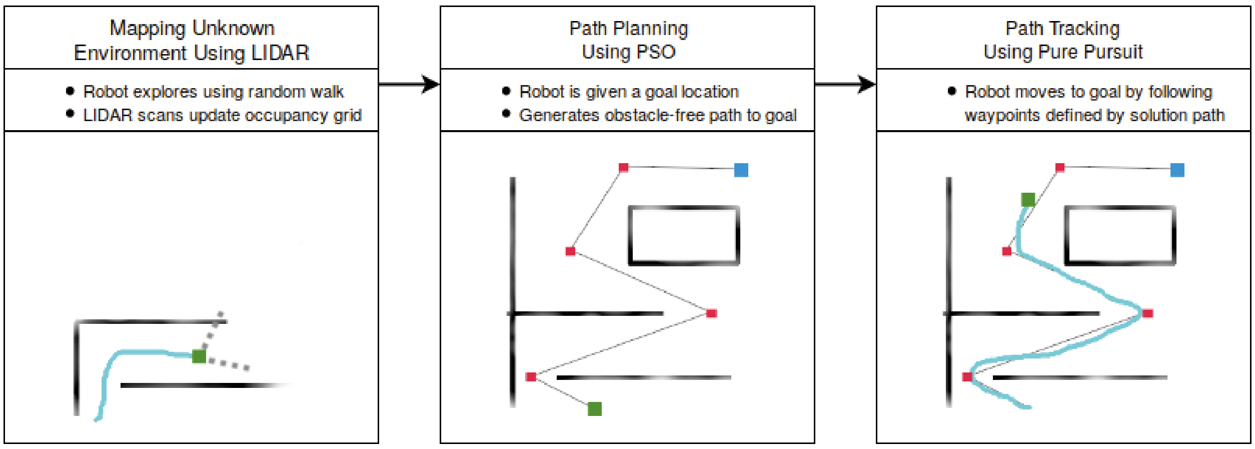

Mapping Unknown Environment Using LIDAR
Robot explores using random walk
LIDAR scans update occupancy grid
Path Planning Using PSO
Robot is given a goal location
Generates obstacle-free path to goal
Path Tracking Using Pure Pursuit
Robot moves to goal by following waypoints defined by solution path

Figure 1. Stages of the proposed autonomous robot navigation system

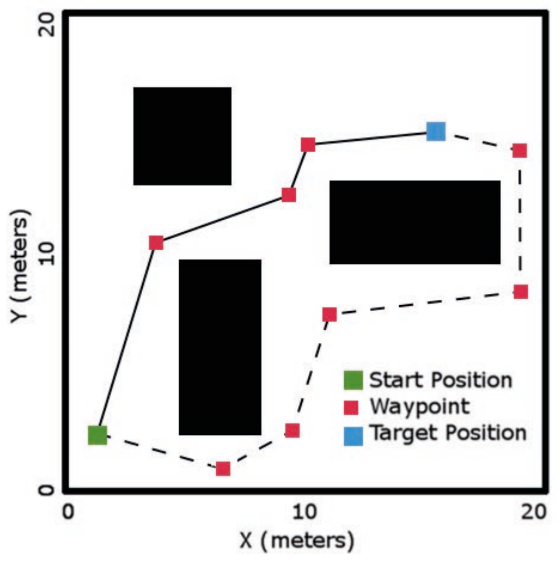


Figure 2. Examples of two paths based a sequences of waypoints between the start and goal locations. The dotted path is less efficient since it is longer in distance.

$$P = [(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)]. \qquad (3)$$

The proposed fitness function to evaluate the quality of a candidate solution is the Euclidean distance, as given in Equation 4). The overall path distance, $Path_D$, is the summation of these distances (See Equation 5).

$$D(x_i,y_i,x_j,y_j) = \sqrt{(x_i-x_j)^2+(y_i-y_j)^2}, \qquad (4)$$

$$P_D = \sum_{i=1}^{n-1} D(<x_i,y_i>,<x_{i+1},y_{i+1}>). \qquad (5)$$

Collisions along the path are calculated by the summation of the number of obstacles between each adjacent waypoints, called $Path_O$ (See Equation 6). This is a summation of the number of occupied cells in the occupancy grid between adjacent waypoint. β represents Bresenham's line algorithm, which is used to count occupied cells in a straight line between two points [25]. This algorithm was originally designed for drawing lines in a bitmap image, but can be generalized to get a list of every cell that lies between the points. Each cell's occupancy is checked to calculate the total number of occupied cells.

$$P_O = \sum_{i=1}^{n-1} \beta(x_i,y_i,x_{i+1},y_{i+1}). \qquad (6)$$

The adopted fitness function F used to minimize the path from a start to target locations using PSO is provided in Equation 7.

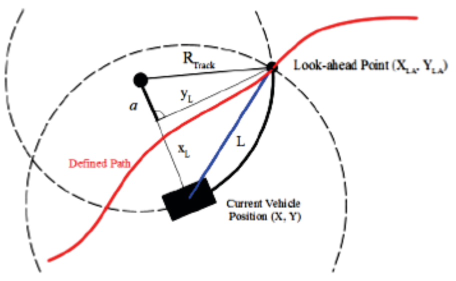
$$F_{path} = P_D + (P_O)^2. \qquad (7)$$

3.3 Path Tracking Using Pure Pursuit

Once the actual position of the robot and its target position are well-defined on the map, it is then essential to generate a trajectory between these two points. Numerous methods were presented in the literature and the selection is governed by each particular environment, the robot size, the payload sensors and the maximum obstacle sizes. In the circumstance of a dynamic environment with unpredictable movement of objects, which is typical for community areas, the most suitable solution is to use an adaptive route planning strategy such that we can avoid dynamic obstacles, while traveling to the target destination.

We defined path tracking as the process of determining the speed and steering settings of an autonomous robot to follow a particular path based on the waypoints generated by PSO. To create a smooth path for the robot such that the robot can avoid obstacles.

According to the proposed ARN system, we suggest the use of the look-ahead path smoothing algorithm known as the Pure Pursuit Path Tracking Algorithm [10]. Pure pursuit was first presented in 1969 [23]. The algorithm was initially intended as a method of geometrically find the curvature that will direct the robot to a chosen path point, termed the goal point. The implementation of the pure pursuit idea is illustrated in Figure 3.



R_{Track}
Look-ahead Point (X_{LA}, Y_{LA})
a
y_L
x_L
L
Defined Path
Current Vehicle Position (X, Y)

Figure 3. Description of the Pure Pursuit algorithm [6]

The algorithm calculates the angular velocity required to move the robot from its current position in space to locate a look-ahead point in front of the robot. We assume that the linear velocity of the robot is constant. The algorithm then moves the look-ahead point on the path according to the current position of the robot until the last point of the path. We can imagine that the robot is regularly chasing a point in front of it.

Assume the current robot position in (X,Y) in the environment is given. And, the constant look-ahead distance is L with the goal is to find the look-ahead point (X_{LA},Y_{LA}) on the path. The algorithm will compute the radius R_{Track} of the arc that connects the current location to the look-ahead distance point. Based on the shown two rectangular triangles given in Figure 3, we can formulate the following equations as given in [6]

$$x_L^2 + y_L^2 = L^2, \tag{8}$$
$$a_L^2 + y_L^2 = R_{Track}^2, \tag{9}$$
$$x_L + a = R_{Track}, \tag{10}$$

Substituting the value of a from Equation 10 in Equation 9 we get that

$$\begin{aligned}(R_{Track} - x_L)^2 + y_L^2 &= R_{Track}^2,\\ R_{Track}^2 - 2R^x_{TrackL} + x_L^2 + y_L^2 &= R_{Track}^2,\\ R_{Track}^2 - 2R^x_{TrackL} + L^2 &= R_{Track}^2.\end{aligned}$$

Finally, we conclude that

$$R_{Track}^2 = \frac{L^2}{2x_L}. \tag{11}$$

R_{Track} value governs the actual arc radius the robot has to follow with a reciprocal value $1/R_{Track}$.

4 Environmental Setup

We divide space into an occupancy grid, where cells are either free or blocked. The grid is binary with a *0* indicating the cell is empty or *1* that the cell is occupied. We used a grid size of $5cm^2$. We used a laser scanner to detect objects which has some uncertainty.

To reduce the risk of collisions inflating the size of an object can be used. The required inflation size depends on the robot, the grid scale, and the turning radius of the robot. The inflation should be large enough to avoid a collision, but not so large that nearby obstacles be joined into a single obstacle when a free and safe path exists between them. We used inflation of $10cm^2$ or two grid cells.

Our experiments are performed in two *Gazebo* environments called *Doorways* and *Open*. Both environments have a 20 $\times$ 20-meter surface with 3D obstacles and a defined target location. 2D coordinates in path planning refer to the ground plane of the space. The two proposed environmental setup are shown in Figures 4 and 5, respectively.

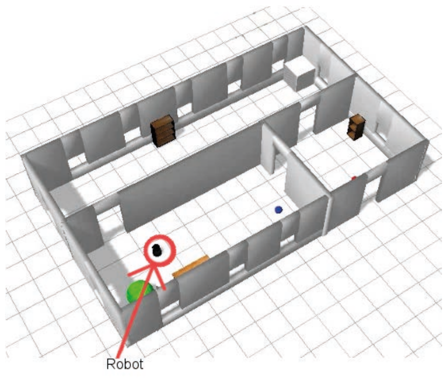


Figure 4. The *Doorways* environment

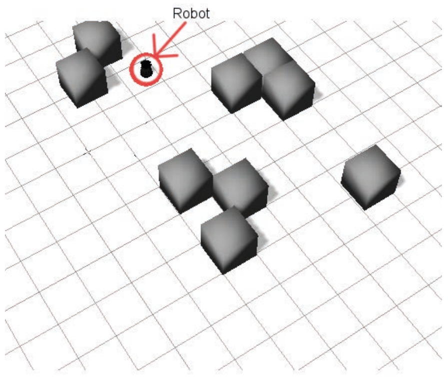


Figure 5. The *Open* environment

- The *Doorways* has two narrow connected hallways and passages between them. A collision-free path between hallways has to go around the wall. The straight path across the wall is a local optimum. If PSO lacks sufficient exploration, it may not find the openings and instead choose the infeasible solution across the wall.

- The *Open* setup has an open space with scattered obstacles. Whereas the feasible optimal solution found in the *Doorways* environment was tightly bounded by the narrow passages, there are many possibilities for feasible solutions in *Open*. The experiment goal is to see if, given a range of feasible solutions, PSO can evolve the optimal path with minimum distance.

Our proposed ARN system is tested using a TURTLEBOT robot in *Gazebo* 3D simulator [17]. *Gazebo* is a 3D robot simulation software package that elevates robot simulation closer to a real robot compared to a typical 2D approach [17]. It supports dynamic physical models, 3D rendering of the environment, multiple robots, and simulation of sensor noise. These features make *Gazebo* results much more realistic than those where the robot is merely a particle in a plane.

This robot can move forward or turn on its axis [13]. The ability to change heading in place is suited for cramped indoor environments. An image of the TURTLEBOT is shown in Figure 6.

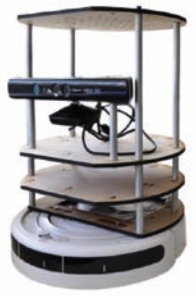


Figure 6. A TURTLEBOT with a laser scanner

5 Experimental Results

Experiments are performed to evaluate the navigation system. An experiment begins by placing the TURTLEBOT in any unobstructed location within a 3D *Gazebo* environment. Then, the TURTLEBOT begins the exploration and mapping phase. An initially empty map is populated with the location of obstacles in real time, based on laser scanner inputs. Exploration ends after 5 minutes. If the exploration stage is successful, then the occupancy map contains all obstacles exteriors since the laser scanner cannot see into the obstacle interior.

The environments in these experiments are assumed to be static, and the TURTLEBOT uses the maps for path planning. During the exploration phase, there is no target. The concept is to create a map that can be used in the future for path planning tasks.

In the path planning phase, the TURTLEBOT uses PSO to generate a solution path as a sequence of waypoints between the start and target locations in the environment. Moving a 3D blue ball inside *Gazebo* with the mouse allows a user to specify the target position, without having to type the coordinates. Path planning generates a solution path between the TURTLEBOT and blue ball. A feasible path is one where the robot can reach the target without collision and an optimal path is both feasible and minimal distance.

Finally, the solution path is used for the path tracking phase so that the TURTLEBOT will move to the target. Path tracking is done by traveling from waypoint to waypoint until reaching the target.

5.1 Map Generation Results

The robot was able to generate maps after the 5-minute exploration phase for both the *Doorways* and *Open* environments. The exploration phase resulted in the occupancy grids shown in Figure 7 and Figure 8. *Gazebo* simulates noisy sensors, so minor imperfection can be seen. The major details of both spaces appear to be successfully captured and suitable for path planning. The minor discrepancy between actual and observed location is expected to be handled by a 2-pixel inflation previously described.

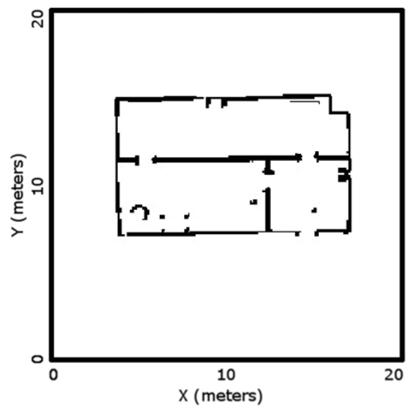


Figure 7. The *Doorways* environment: Occupancy grid generated by robot using laser scanner

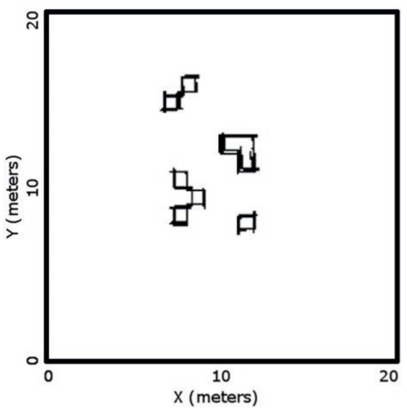


Figure 8. The *Open* environment: Occupancy grid generated by robot using laser scanner

5.2 Path Planning Results

In this Section, we describe the developed experimental results using PSO for robot path planning. PSO was used to evolve the possible set of waypoints path P from a start to target locations (see Equation 3). PSO used the Euclidean distance as a criteria to minimize. Using the maps generated in the first phase of the proposed ARN system, the TURTLEBOT used PSO to generate the shortest path from start to target.

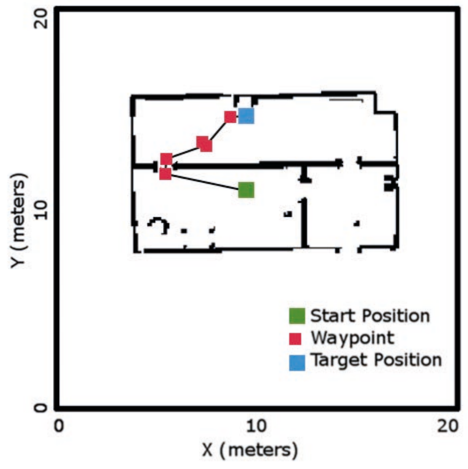
According to our simulation, the robot was observed to arrive at the target position successfully. PSO generated best paths for all four cases considered are shown in Figures 5.2.1 to Figure 5.2.1. The PSO parameters used for the experiments are given in Table 1.

Table 1. PSO parameters used

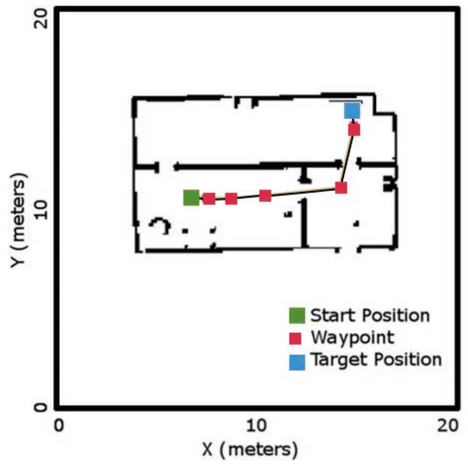
Parameter	Value
Number of iterations	1500
Swarm size	500
c_1	1.496
c_2	1.494
$w_m in$	0.3
$w_m ax$	0.7298

5.2.1 Performance with various number of waypoints:

Determining the optimal number of waypoints is an optimization problem. Here, it is clear that at least two waypoints are needed for the required number of turns. All the solution paths found are effective at distance minimizing and collision-avoidance.



(a) Experiment 1 PSO solution path

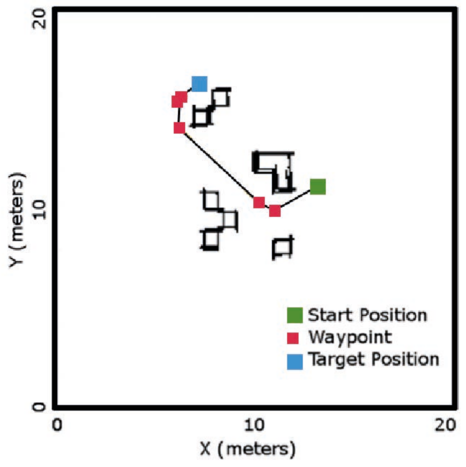


(b) Experiment 2 PSO solution path

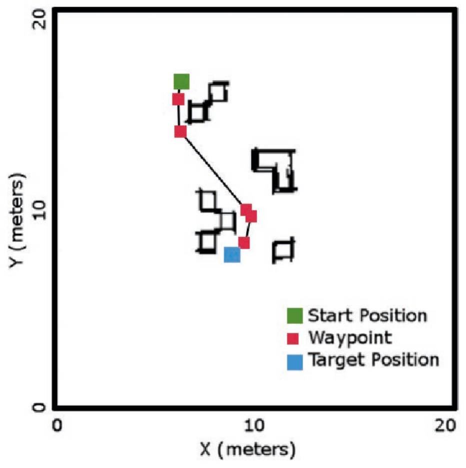
Figure 9. Results of path planning in the *Doorways* environment

We run PSO for 50 experiments using three, four and five waypoints as shown in Figure 5.2.1. While the non-determinism of PSO suggests that the outcome will differ across runs, there should be a bound on the variables such that the optimal path is consistently chosen. The best, worst, and average convergence curves are shown in Figures 5.2.2, 5.2.2, and 5.2.2 for three, four and five waypoints, respectively.

Notice that in Figures 5.2.1, 5.2.1, and 5.2.1, two waypoints are overlapped. The waypoint symbols in those cases are slightly offset for clarity, but the actual waypoint's coordinates are identical. Such behavior indicates that more waypoints are being assigned than required for the problem. Additional waypoints increase the problem complexity and runtime substantially.



(a) Experiment 1 PSO solution path

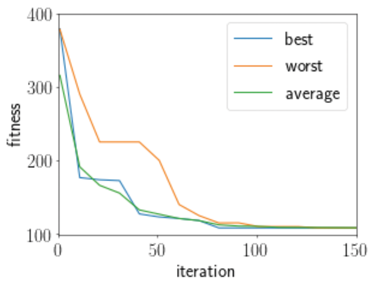


(b) Experiment 2 PSO solution path

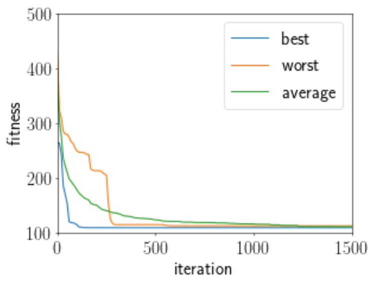
Figure 10. Results of path planning in the *Open* environment

5.2.2 Performance with a various number of iterations:

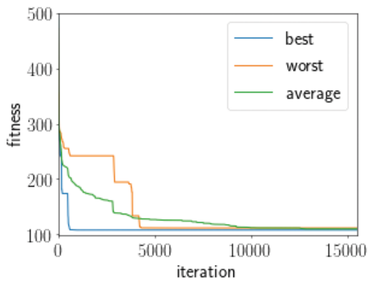
Recall that the objective is to minimize the fitness, so a lower fitness (i.e., minimum distance) value is better. The expectation is that the initial fitness value of PSO will be very high since it is calculated from the initial random placement of particles. As the solution improves using local and global information, the particles should converge to an optimal solution. This behavior is demonstrated in Figures 5.2.2, 5.2.2, and 5.2.2. Notice that the X-axis varies substantially across figures, increasing in range as the number of waypoints increases. The best, worst, and average converge to approximately the same solution within 100 iterations using only 3 waypoints. It takes almost 10000 iterations to observe the same when using 5 waypoints. These results indicate that, while the number of iterations to do so may increase dramatically, in all trials performed the solution converges to a tight fitness bound.



(a) PSO convergence with three waypoints



(b) PSO convergence with four waypoints



(c) PSO convergence with five waypoints

Figure 11. PSO results while evolving three, four and five waypoints

Further evidence of successful convergence is the repetition of a solution over multiple iterations as well as redundant waypoints in a solution, as seen in Tables 2, 3, and 4. In each case, the final two rows are repeated. These waypoints are shown in the discrete occupancy grid space rather than the continuous space of *Gazebo*. Notice that the final solution when using five waypoints includes duplicate waypoints. This redundancy indicates that five waypoints are unnecessarily large.

Table 2. PSO path of three waypoints with various iteration generations

Gen	X1	Y1	X2	Y2	X3	Y3
1	46	84	61	144	10	89
100	93	124	94	128	120	148
200	106	127	105	131	120	146
300	112	116	114	140	120	146
400	112	112	116	142	120	146
500	112	112	116	142	120	146

Table 3. PSO path of four waypoints with various iteration generations

Gen	X1	Y1	X2	Y2	X3	Y3	X4	Y4
1	28	113	37	111	8	65	19	64
1000	85	116	104	120	93	112	144	115
5000	110	132	117	144	120	144	124	142
10000	110	110	116	143	121	144	124	141
15000	112	112	116	143	121	144	124	141
20000	112	112	116	142	119	144	123	143
25000	112	112	116	142	119	144	123	143

5.2.3 Performance with the optimal number of waypoints:

Having determined that three waypoints was effective for experiment 1 in *Doorways*, the other experiments were evaluated using 3 waypoints to see if the result is transferable. Because different environments and goals will affect the convergence rate, each trial was allowed 5000 iterations. We run PSO for 50 experiments to obtain the optimal values of the waypoints. The calculated fitness value at 5000 iteration is shown in Table 5 with average, best, and worst results for each experiment.

With 3 waypoints, a collision-free path was generated in each experiment. Given the convergence rate of experiment 1 in *Doorways* (see Figure 2), a very tight bound between the best and worst runs was expected after 5000 iterations. This was the case with the widest fitness range 10.256 and the tightest 1.153. At the start of PSO, trials often range in fitness over 100, as shown in Figures 2 - 4. 5000 iterations complete in under 30 seconds, which is observed to be more than enough to ensure high-quality solutions in these experiments. These results suggest that despite very different obstacle configurations, PSO waypoint parameters have the potential to be reused when the number of required turns is no larger than that required by the experi-

Table 4. PSO path of five waypoints with various iteration generations

Gen	X1	Y1	X2	Y2	X3	Y3	X4	Y4	X5	Y5
1	0	0	0	0	0	0	0	0	0	0
1000	110	104	106	102	100	100	97	101	95	101
5000	110	104	106	102	100	101	95	101	95	101
10000	109	135	118	144	135	140	136	138	135	136
15000	116	142	118	144	135	140	135	138	135	139
20000	116	142	118	144	135	140	135	138	135	139

ment where the parameter was determined.

Table 5. Results of running PSO using three waypoints for the experiments using *Doorways* and *Open* environment. 50 experiments with 5000 iterations results

Setup	Experiment	Worst Fitness	Average Fitness	Best Fitness
Doorways	1 (Fig. 5.2.1)	111.80	108.33	108.26
Doorways	2 (Fig. 5.2.1)	116.83	113.80	106.58
Open	1 (Fig. 5.2.1)	108.74	102.60	101.33
Open	2 (Fig. 5.2.1)	65.50	65.22	64.35

5.3 Path Tracking Results

The final phase is path tracking, where the TURTLEBOT follows the solution path generated in the path planning phase. To analyze PSO convergence, more paths were generated in the standalone PSO module than were evaluated within the full system. As part of navigation, infeasible paths are automatically rejected and planning occurs until a feasible path is found. Here, feasibility refers to the path planner's detection of intersecting obstacles in the occupancy grid. The path tracking experiments are needed to ensure that the allowed paths are truly feasible in the *Gazebo* environment, given the TURTLEBOT'S dimensions and dynamics.

Ten path tracking trials were performed for the two experiments in each setup for a total of forty trials. In each case, the robot was observed to navigate to the target location without collisions or any other issues. The robot did approach the sides of the doorways very closely in the *Doorways* setup; the 2-pixel expansion of the occupancy grid was barely sufficient to avoid an accident in such narrow regions.

5.4 Computational Complexity

PSO performs some number of iterations to converge to an approximately optimal solution. Thus, runtime performance depends on the time for a single iteration and the number of iterations required. Both depend on the number of waypoints. As has been demonstrated, increasing the number of waypoints increases the dimensionality of the problem and the number of iterations required to converge. It remains to determine the runtime of a single iteration concerning the number of waypoints.

Each iteration requires computing the fitness function (Algorithm 7) as part of PSO (Algorithm 1). The complexity of the fitness function depends on the number of waypoints because it has to evaluate the path between adjacent waypoint pairs. This requires summing the distance between each pair as well as checking for obstacles at each cell intersected by the line between waypoints. The distance summation is simply repeated calls of the Euclidean distance, but checking for obstacles is more involved. When waypoints are far apart, a large number of cells need to be checked.

Overall, the complexity of a single iteration is $O(n)$ where n is the number of waypoints to search for. But for a more detailed look at how the runtime scales with the number of waypoints, Table 6 shows the mean iteration time and the estimated and measured runtimes to complete 10,000 iterations. All runtimes are based on processor time. The purpose of comparing the estimated and measured overall runtimes is to suggest that using the mean iteration time is appropriate for estimating the runtime of a given number of iterations.

Table 6. The effect of the number of waypoints on the runtime of each PSO iteration. The mean iteration runtime is reliable enough to allow estimating the runtime of a given number of iterations.

Waypoints	Mean iter time (ms)	Estimated 10000 iters (s)	10000 iters (s)
3	0.1141	11.41	11.43
4	0.1450	14.50	14.63
5	0.1719	17.19	17.51
6	0.1971	19.71	20.29
7	0.2298	22.98	23.94
8	0.2560	25.60	25.87
9	0.2821	28.21	31.81
10	0.3054	30.54	31.14
11	0.3391	33.91	33.41
12	0.3690	36.90	38.41
13	0.3879	38.79	38.75
14	0.4209	42.09	42.06
15	0.4533	45.33	45.10

Unlike the number of iterations needed to converge, the per-iteration runtime is independent of the environment and start/stop locations. Thus, an estimate of the convergence duration can be made for any scenario where an estimate of the required iterations has been made. Table 7 gives the estimated duration for convergence in experiment 1 in the *Doorway* setup (see Figure 5.2.1). The estimated required iterations are taken from the convergence curves in Figures 5.2.2, 5.2.2, and 5.2.2 for 3, 4 and 5 waypoints, respectively.

Table 7. Runtimes for convergence can be estimated using the runtime per iteration and number of iterations to converge. Number of required iterations is based on experiment in Figure 5.2.1.

Waypoints	Required Iterations	Estimated Runtime (s)
3	100	11.4
4	350	50.75
5	50000	8595.00

5.5 PSO Parameter Sensitivity

The results discussed so far used the PSO parameters shown in Table 1. The values for c_1, c_2, w_{min}, and w_{max} are the defaults of the PSO library used [16]. The performance when using default parameters is of interest because typical users are expected to do little to no change. Even in literature where PSO or similar algorithms are used, parameter tuning is commonly absent (as observed by Tewold et. al. [27]). As these algorithms become increasingly available through convenient software libraries, the barrier using such algorithms is very low. It is encouraging to observe reliable convergence to feasible, approximately optimal solutions for the complicated path planning problem without any modification of PSO parameters.

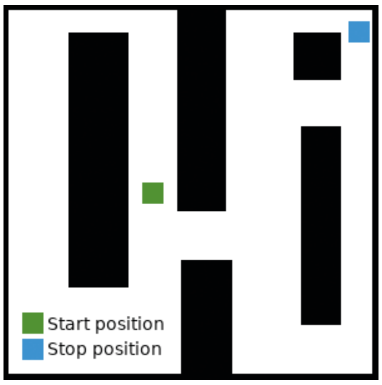


Figure 12. Environment setup for PSO parameter testing

However, it is known that PSO is dependent on its parameters and so next they are modified in an attempt to increase performance. The PSO parameters are the cognitive coefficient c_1, social coefficient c_2, and inertia weight w.

Adopting the tuning parameters used in [15], the parameters are restricted to $c_1 \in [0,4]$ and $c_2 \in [0,4]$. Samples are uniformly selected with an interval of 0.2. Each pair of values is tested with 10 trials. The mean, min, and max fitness of each pair are plotted as well as the number of infeasible solutions.

Figure 13 shows the topography of the parameter space. The z-axis is the mean fitness over 10 experiments, with valleys having the lowest (i.e., best) fitness and peaks having the worst. Some observations include:

– The topography is complex with excellent solutions (valleys) situated among tall high-fitness peaks. This suggests complex relations between c_1 and c_2.

– There is an overall behavior that shows stronger performance with higher values of c_1 and lower values of c_2.

– High-quality solutions are even found when disabling c_2 entirely, but subtly better overall fitness is obtained when c_2 is 0.5.

– Intuition expects that a balance of cognitive and social emphasis would produce better planning, but for the particular fitness function and PSO implementation, it appears that the social coefficient can easily lead to poor fitness values. This is in contrast with results from [28] where the opposite was found: lower c_1 and higher c_2 gave significantly improved performance. This result emphasizes the importance of exploring the effect of weights for every specific application.

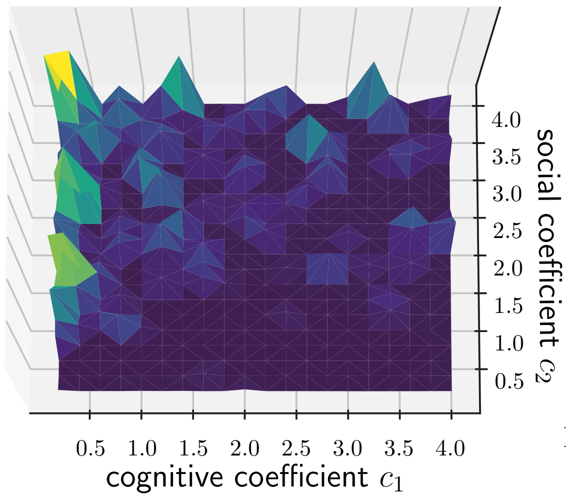


Figure 13. Parameter sensitivity test of mean fitness over social and cognitive coefficients

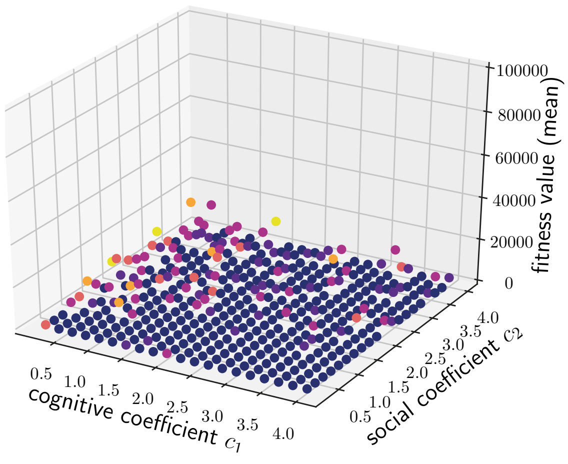


Figure 14. Parameter sensitivity test of mean fitness over social and cognitive coefficients

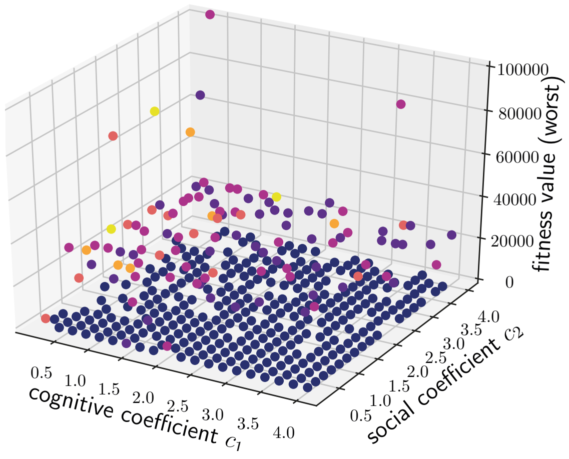


Figure 15. Parameter sensitivity test of the worst fitness over social and cognitive coefficients

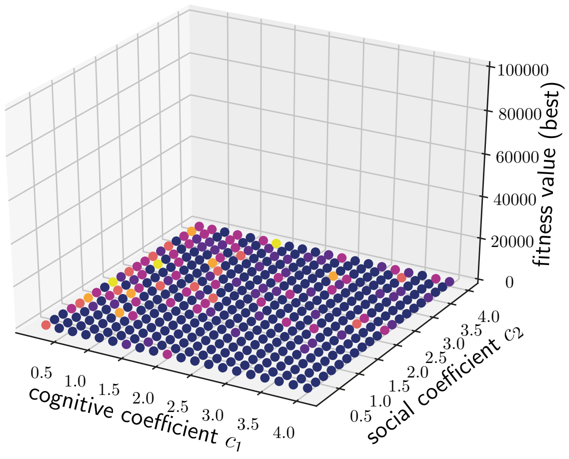


Figure 16. Parameter sensitivity test of best fitness over social and cognitive coefficients

The fitness values are shown with colors indicating the proportion of infeasible solutions for the mean, minimum, and maximum fitness in Figures 14, 16, and 15, respectively. The colors graduate from blue to yellow with blue indicating no infeasible solutions and yellow indicating the most. The maximum number of infeasible paths was found to be 40 (40% of the trials for the parameters pair). For the worst combinations of c_1 and c_2, it is almost a coin flip as to whether the solution will be feasible. However, in the regions where the value of c_2 is lower, the infeasible paths are almost absent.

Even for poor parameter selections, PSO would occasionally perform as well as if it were using the best parameters. The parameters strongly affect reliability but have almost no impact on best-case fitness. Given enough opportunities, a high-quality feasible solution can be found.

Because of the parameter space's complex topography, the tuned coefficients were selected based not only on their fitness but by their neighbor's as well. High performing valleys in between large peaks are assumed to be less stable than smooth plains. Using the results of the parameter optimization is shown the selected parameters are $c_1 := 3, c_2 := 1$.

6 Conclusions and Future Work

This research proposed a robot navigation system using a TURTLEBOT robot to autonomously map an unknown environment using a 3D physics-based simulator. PSO demonstrated high performance in searching for the optimal path (i.e., waypoints). The results show that the robot can generate and follow effective paths in multiple environments with various characteristics.

Future work is to explore adaptive assignment of the number of waypoints. While the experiments shown were able to make use of three or four waypoints, this will be insufficient in a more complex environment which requires more turns to reach the goal. A mechanism should exist to dynamically add more waypoints if needed, but still use as few as possible to reduce complexity. One approach would be to add a new waypoint each time PSO fails to find a feasible solution after a pre-defined number of iterations. Another area to explore more sophisticated search space exploration methods. This work focus on path planning, but the exploration phase based PSO is critical to the ARN system in an unknown environment. Frontier-based, information-theoretic, and other approaches should be evaluated when exploring with both single and multiple robots.

References

[1] A hybridization of an improved particle swarm optimization and gravitational search algorithm for multi-robot path planning, 28.

[2] M Shahab Alam, M Usman Rafique, and M Umer Khan. Mobile robot path planning in static environments using particle swarm optimization. International Journal of Computer Science and Electronics Engineering (IJCSEE), 3(3):253–257, 2015.

[3] Dora-Luz Almanza-Ojeda, Yazmín Gomar-Vera, and Mario Ibarra-Manzano. Occupancy Map Construction for Indoor Robot Navigation. 10 2016.

[4] Ismail Altaharwa, Alaa Sheta, and Mohammed Alweshah. A mobile robot path planning using genetic algorithm in static environment. Journal of Computer Science, 4, 01 2008.

[5] J. C. Bansal, P. K. Singh, M. Saraswat, A. Verma, S. S. Jadon, and A. Abraham. Inertia weight strategies in particle swarm optimization. In 2011 Third World Congress on Nature and Biologically Inspired Computing, pages 633–640, Oct 2011.

[6] Ján Bačík, Frantisek Durovsky, Milan Biros, Karol Kyslan, Daniela Perdukova, and P Sanjeevikumar. Pathfinder – development of automated guided vehicle for hospital logistics. IEEE Access, 5:26892 – 26900, 10 2017.

[7] Sumana Biswas, Sreenatha G. Anavatti, and Matthew A. Garratt. Obstacle avoidance for multi-agent path planning based on vectorized particle swarm optimization. In George Leu, Hemant Kumar Singh, and Saber Elsayed, editors, Intelligent and Evolutionary Systems, pages 61–74, Cham, 2017. Springer International Publishing.

[8] E. A. S. Carballo, L. Morales, and F. Trujillo-Romero. Path planning for a mobile robot using genetic algorithm and artificial bee colony. In 2017 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE), pages 8–12, Nov 2017.

[9] Xin Chen and Yangmin Li. Smooth Path Planning of a Mobile Robot Using Stochastic Particle Swarm Optimization. In 2006 International Conference on Mechatronics and Automation, pages 1722–1727, Luoyang, June 2006. IEEE.

[10] R. Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical Report CMU-RI-TR-92-01, Carnegie Mellon University, Pittsburgh, PA, January 1992.

[11] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages, ECAI '96, pages 21–35, London, UK, UK, 1997. Springer-Verlag.

[12] Avinash Gautam and Sudeept Mohan. A review of research in multi-robot systems. 08 2012.

[13] C. Georgoulas, T. Linner, A. Kasatkin, and T. Bock. An ami environment implementation: Embedding turtlebot into a novel robotic service wall. In ROBOTIK 2012; 7th German Conference on Robotics, pages 1–6, May 2012.

[14] S. Ghosh, P. K. Panigrahi, and D. R. Parhi. Analysis of fpa and ba meta-heuristic controllers for optimal path planning of mobile robot in cluttered environment. IET Science, Measurement Technology, 11(7):817–828, 2017.

[15] Suhanya Jayaprakasam, Sharul Kamal Abdul Rahim, and Chee Yen Leow. Psogsa-explore: A new hybrid metaheuristic approach for beampattern optimization in collaborative beamforming. Applied Soft Computing, 30:229–237, 2015.

[16] Kyriakos Kentzoglanakis. Particle swarm optimization in c. https://github.com/kkentzo/pso, 2017.

[17] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), volume 3, pages 2149–2154 vol.3, Sept 2004.

[18] E. Masehian and D. Sedighizadeh. Multi-Objective PSO- and NPSO-based Algorithms for Robot Path Planning. Advances in Electrical and Computer Engineering, 10(4):69–76, 2010.

[19] Amin Zargar Nasrollahy and Hamid Haj Seyyed Javadi. Using Particle Swarm Optimization for Robot Path Planning in Dynamic Environments with Moving Obstacles and Target. In 2009 Third UKSim European Symposium on Computer Modeling and Simulation, pages 60–65, Athens, Greece, 2009. IEEE.

[20] Millie Pant, Radha Thangaraj, and Ajith Abraham. Particle swarm optimization: performance tuning and empirical analysis. Foundations of Computational Intelligence, 3:101–128, 2009.

[21] P. Raja and S. Pugazhenthi. Path Planning for Mobile Robots in Dynamic Environments Using Particle Swarm Optimization. In 2009 International Conference on Advances in Recent Technologies in Communication and Computing, pages 401–405, Kottayam, Kerala, India, 2009. IEEE.

[22] P Raja and S Pugazhenthi. Optimal path planning of mobile robots: A review. International Journal of the Physical Sciences, 7:1314–1320, 02 2012.

[23] L. Scharf, W. Harthill, and P. Moose. A comparison of expected flight times for intercept and pure pursuit missiles. IEEE Transactions on Aerospace and Electronic Systems, 4:672–673, 1969.

[24] K. H. Sedighi, K. Ashenayi, T. W. Manikas, R. L. Wainwright, and Heng-Ming Tai. Autonomous local path planning for a mobile robot using a genetic algorithm. In Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753), volume 2, pages 1338–1345 Vol.2, June 2004.

[25] Dmitri Sokolov. tinyrenderer, 2017.

[26] Monica Anderson LaPoint Maria Gini Nikolaos Papanikolopoulos John Budenske Steven Damer, Luke Ludwig. Dispersion and exploration algorithms for robots in unknown environments. volume 6230, 2006.

[27] Girma S Tewolde, Darrin M Hanna, and Richard E Haskell. Enhancing performance of pso with automatic parameter tuning technique. In 2009 IEEE Swarm Intelligence Symposium, pages 67–73. IEEE, 2009.

[28] Gerhard Venter and Jaroslaw Sobieszczanski-Sobieski. Particle swarm optimization. AIAA journal, 41(8):1583–1589, 2003.

Evan Krell received his M.Sc. from the Department of Computing Sciences, Texas A&M - Corpus Christi, TX, USA, where he also did his undergraduate studies. He is currently pursuing a Ph.D. at the same. He is currently working in the Pixel Island research lab, focusing on mission planning for unmanned surface vehicles and collision avoidance for unmanned aircraft. Previously, he was involved with bioinformatics for the Texas A&M - Corpus Christi Genetics Core Lab. His research interests are artificial intelligence, unmanned systems, image processing and bioinformatics.

Alaa Sheta is Assistant Professor at the Department of Computing Sciences, Texas A&M University-Corpus Christi, TX, USA. He received his B.E., M.Sc. degrees in Electronics and Communication Engineering from the Faculty of Engineering, Cairo University in 1988 and 1994, respectively. He received his Ph.D. from the Computer Science Department, School of Information Technology and Engineering, George Mason University, Fairfax, VA, the USA in 1997. He published three books in the area of Landmine Detection, and Image Reconstruction of a Manufacturing Process. He is also the co-editor of the

book entitled,”Business Intelligence and Performance Management-Theory, Systems, and Industrial Applications” by Springer Verlag, United Kingdom, published in March 2013. Dr. Sheta published more than 130 journal and conference papers. He mainly focus on machine learning and its application in the domain of industrial process modeling, software reliability modeling, software cost estimation, automation, medical/non-medical imagining, and biomedical applications.

Arun Prassanth Ramaswamy Balasubramanian received his Master’s degree in Computer Science from Texas A&M University-Corpus Christi, TX in 2018. He is currently working for Nokia Bell Labs in Sunnyvale, CA. His research interests include robotics and artificial intelligence, ambient intelligence, machine learning, multi-agent robotic systems and computer vision. He has 4+ years of industry experience as Software Engineer in computer networks and is originally form India.

Scott A. King received his Ph.D. from The Ohio State University in 2001 in Computer and Information Science. He joined Otago University in Dunedin, New Zealand immediately after receiving his Ph.D. and was there for two and half years. He then joined Texas A&M University-Corpus Christi in 2004 and has been there since. He is currently the Chair of the Department of Computing Sciences. He expertise in computer graphics, visualization, HCI, autonomous vehicles, smart environments. He is a member of IEEE and ACM.