 doi:10.20944/preprints202012.0307.v1

# A Cooperative Multi-Agent Based Path-Planning and Optimization Strategy for Dynamic Environment

**Junior Sundar[1], Sumith Yesudasan[2*] and Sibi Chacko[1]**

[1] School of Engineering and Physical Sciences, Heriot-Watt University, Dubai, UAE
[2] Department of Engineering Technology, Sam Houston State University, Huntsville, TX, USA

**Author emails:**

Junior Sundar: juniorsundar@gmail.com
Sibi Chacko: c.sibi@hw.ac.uk
*Corresponding Author Email: sumith.yesudasan@shsu.edu

## Abstract

This investigation explores a novel path-planning and optimization strategy for multiple cooperative robotic agents, applied in a fully observable and dynamically changing obstacle field. Current dynamic path planning strategies employ static algorithms operating over incremental time-steps. We propose a cooperative multi-agent (CMA) based algorithm, based on natural flocking of animals, using vector operations. It is preferred over more common graph search algorithms like A* as it can be easily applied for dynamic environments. CMA algorithm executes obstacle avoidance using static potential fields around obstacles, that scale based on relative motion. Optimization strategies including interpolation and Bezier curves are applied to the algorithm. To validate effectiveness, CMA algorithm is compared with A* using static obstacles due to lack of equivalent algorithms for dynamic environments. CMA performed comparably to A* with difference ranging from -0.2% to 1.3%. CMA algorithm is applied experimentally to achieve comparable performance, with an error range of -0.5% to 5.2%. These errors are attributed to the limitations of the Kinect V1 sensor used for obstacle detection. The algorithm was finally implemented in a 3D simulated space, indicating that it is possible to apply with drones. This algorithm shows promise for application in warehouse and inventory automation, especially when the workspace is observable.

**Keywords**: CMA, Path Planning, Dynamic Environment, Multi Agent, Autonomous Navigation

## 1. Introduction

Path-planning and optimization for autonomous systems is an increasingly significant branch of modern artificial intelligence. With the growing use of unmanned aerial and ground vehicles, to perform tasks such as delivery and terrain-exploration, autonomy in systems is becoming an industrial standard. In this work, the possibility of having multiple autonomous agents navigating a space cooperatively is explored. The space's properties are that it is fully observable, dynamic, and stochastic.

Currently there are no well-established solutions for such environments because such configurations fall under a niche category because they only apply if the field is fully observable. Hence this strategy will be a new addition to this field.

The algorithm developed in this research has multiple aspects that need to be tested. To keep the investigation succinct, it is broken into segments to focused on verifying and validating the performance of the algorithm. The objective is to determine whether this algorithm can provide an optimal path-plan. First, the development of the algorithm will be detailed. This algorithm will then be verified numerically against existing alternatives using a standardized method. A similar test will be conducted practically to validate effectiveness when applied. Finally, the strategy will be tested in a simulated 3D observable and dynamic workspace.

### 1.1. Background

Before moving forward, the concept of an 'agent' must be defined. The definition followed in this research is established by Woolridge et al. [1], as a system operating in an arbitrary environment capable of autonomous action to meet a set objective. This can be broken down into robotic and software agents; former relies on sensor inputs to provide mechanical output, while latter utilizes similar inputs to provide numeric or picture outputs [2]. The mode of operation of these agents can be decomposed into a layered architecture, where each level is equal with no central control. An alternative system is employed by Singh et al. [3] and Wang et al. [4], where although the architecture appears parallel, there is a distinct separation between the agents. In Singh et al. [3], the agents are communicating with a central facilitator, apart from which there is also communication from each individual agent to external systems. In Wang at al. [4], the architecture lacks parallelism as only certain agents can communicate among each other. The use of a facilitator assures scalability, and lack of such infrastructure in the latter study can cause bottlenecks at large scale applications. Moreover, it was found that for robotic systems, a layered architecture of software agents is found to be most efficient [5], [6].

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 December 2020 doi:10.20944/preprints202012.0307.v1



Path-planning and optimization strategies through agent-based infrastructures are classified based on their applied environment. The environment can be categorized as fully or partially observable, cooperative, or competitive, deterministic, or stochastic, and as static or dynamic [2]. This field has a dichotomy based on the observability and predictability of the environment. A robotic agent traversing a partially observable (limited to vicinity of agent) and static environment are geared for exploration. This employs probabilistic models such as in [7]–[9]. These papers focused on the search-and-rescue (SAR) application of path-planning, while Teatro et al. [9] explores implementation with dynamic environment. Another strategy used in this is SLAM (simultaneous localization and mapping) as employed by Clemens et al. [10]. The other half of this field is based on a fully observable and static environment. The most implemented algorithm in static environments is the A*-Algorithm such as in Silver et al. [11] and Persson et al. [12]. A* is the most optimal in these uses due to its heuristic function that narrows the search to the best possible route. These literatures explore methods of improving the scope of this algorithm. In Persson et al. [12], the heuristics is improved by implementing a weight to a region in space based on the density of the explorable region through sampling. Similarly, Silver [11] focuses the implementation of pathfinding through A* in computer graphics and games, through the use of multi-agent based software. Moreover, fuzzy logic controllers and ripple spreading algorithm (RSA) is also explored with relevance to this strategy [13], [14]. The fuzzy logic system offers a primitive solution to the problem, as it offers basic obstacle avoidance without optimization. The ripple spreading algorithm is beneficial when the operational nodes are limited, as multiple nodes to the search space causes computation to become time consuming. Additionally, within this section, another option explored was the genetic algorithm [15]. This strategy is effective for optimization; however, the computational strain far outweighs the benefit of this strategy. Comparing to A*, the genetic algorithm requires multiple generations to provide a comparatively optimal path. Furthermore, if the search space is complex, the algorithm tends to struggle for multiple generations. The genetic algorithm, and other optimization techniques are used in a wide range of investigations in different fields [16]–[22]. The limitation of these algorithms is that it is difficult to extend their use into a dynamic environment.

The challenge is to provide path-planning and optimization while in a dynamic and stochastic environment. The workspace where the strategy will be applied is fully observable, thus full knowledge of the environment can be obtained through a sensor module at any time. Since it is dynamic and stochastic, its properties are constantly changing, and they cannot be known precisely. To traverse this space, the robotic agents need to work cooperatively, therefore robotic agents will be working together as opposed to competitively to solve the problem. An existing strategy involves using A* algorithm in intervals of time [23]. This breaks the continuous problem into discrete steps and applies A* algorithm over small time intervals while in the presence of moving obstacles. Although effective, it is not scalable. Alternate methods look towards biological systems for optimal strategies. One strategy was implemented for small-object detection and avoidance through frequency analysis of optic signals [24]. Wider research produced a conference proceeding modeled flight path of flocks, herds, and schools. This strategy implements flocking behavior through a mathematical model [25]. This was originally utilized for computer generated graphic implementation, however it showed potential for application to an on-line obstacle avoidance, path-planning, and optimization problem.

### 1.2. Principles of Flocking Model

The flocking mathematical model by Craig Reynolds [25] is unique in the way it emulates flock behavior. It is not intended as a path-planning strategy and is built on four principles: (1) Inter-community collision avoidance; (2) Community velocity matching; (3) Flock centering; (4) Migration urge.

The first principle ensures that there is no collision between constituents of a flock as they move. The second principle is used to ensure that constituents move in unison. The third principle is used to maintain cohesion, while the final principle drives the flock towards an end point. Each of the above principles provide a velocity vector $(v)$ output that contributes to the change in position $(p)$ of each constituent in the flock. An overview is shown in Equation 1; note that the subscript corresponds to the output from the numbered principle.

$$\Delta p = v_1 + v_2 + v_3 + v_4 \quad (1)$$

A generic pseudocode for these principles can be obtained from Parker [26]. Drawing inspiration, a version more relevant to the current problem was programmed in Algorithm 3-6. The beauty of this model is that since it is based on mathematical summation of vectors, it can be built upon. Hence an agent implementing obstacle avoidance will be developed and integrated.

## 2. Methodology

### 2.1. Dynamic Obstacle Avoidance Strategy

In this algorithm, a flock will be called 'community', and its constituents, 'vehicles'. Adhering to the concepts of the flocking model, input will be the properties of one vehicle in the community, the output will be the velocity contribution. For this, first assume that the position of all '$n$' obstacles is known and is stored in an array as in Equation 2, and that the region of the obstacles (area in 2D or volume in 3D) is also stored.

$$Obstacles = [(x1, y1, z1), (x2, y2, z2), \ldots, (xn, yn, zn)] \quad (2)$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 December 2020 doi:10.20944/preprints202012.0307.v1



A solution implemented in Ribeiro [27], following a "Bug's Algorithm" based on an "Artificial Potential Field Approach" [28], was adapted. Essentially, the obstacle avoidance agent activates if a vehicle is near an obstacle. The adapted version utilizes discretized potential fields, in which the vehicle is either in or out of the field around the obstacle. To draw this field, the stored obstacles' region is amplified by a weight criterion as in Equation 3.

$$Regionof\ Obstacle[i] = Regionof\ Obstacle[i] \times Weight \quad (3)$$

If the vehicle is within this region, it will immediately move in a direction opposite from the obstacle. If regions of multiple obstacles overlap, multiple vectors will contribute to the new velocity. This can be seen in Figure 1.

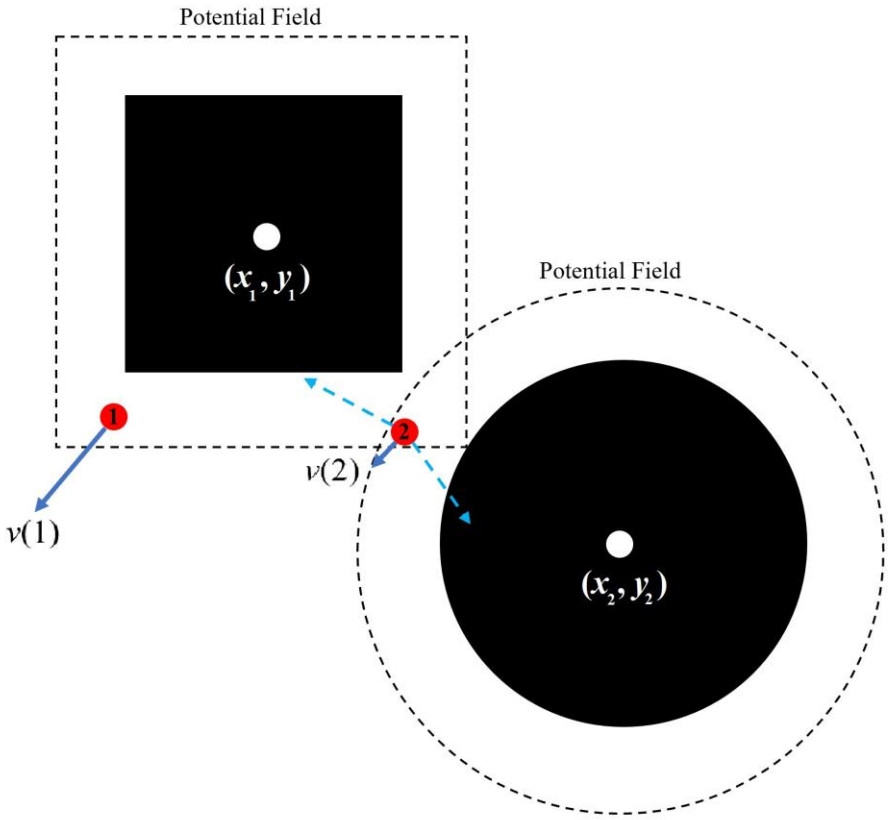


Figure. 1 Influence of potential fields of stationary obstacles on vehicles 1 and 2

This logic is then extended to cases where obstacles are moving with a constant speed. One issue is that the vehicles are given incentive to turn only after they enter the influence of an obstacle. If the obstacle is moving towards the vehicle at a greater speed, collision can occur before the vehicle has an opportunity to respond. This is solved by actively changing the potential field according to relative movement of the obstacles. This is derived from a paper by Pandey et al. [13] on application of fuzzy logic controllers in path-planning. Compared to the least change in obstacle position, other obstacles' field weight is amplified by a factor. Referring to Figure 2, since obstacle 1 is moving faster than obstacle 2, potential field weight is amplified. This is also expressed as a function in Algorithm 1 (see Appendix 1).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 December 2020 doi:10.20944/preprints202012.0307.v1



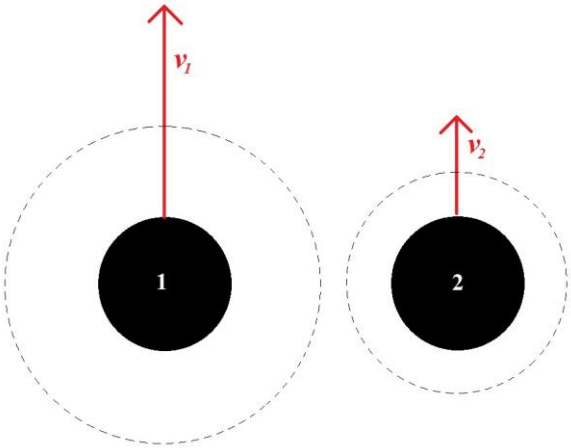


Figure. 2 Relative change in weight of potential field around similar obstacles based on relative motion

### 2.2. Developing the Algorithm

All principles were adapted and programmed for a robotic environment. Consider a community of vehicles, and a field of dynamic obstacles. The main loop presented in Figure 3 iterates through each of the vehicles in the community and applies the principles derived from the flocking model, as well as the obstacle avoidance principle. These software agents are implemented in a parallel architecture. The algorithm terminates when the target is reached.

The principles, that are defined as agents in the main algorithm, are shown in Algorithm 3-7 (see Appendix 1). The inputs to these agents are the position and velocity components for one vehicle in the community. The output from the agents is a unit velocity vector component whose summation, from the "Move i$^{th}$ Vehicle" function in Algorithm 2 (see Appendix 1), determines the next step the vehicle needs to take. Within this function, the vector contribution from each principle can be prioritized with a multiplier "c1, c2, c3, c4 and c5". For instance, if cohesion is preferred over goal seeking, $c3$ can be set greater than 1, and $c4$ can be reduced. Henceforth, this algorithm will be referred to as cooperative multi-agent (CMA) strategy or algorithm.

### 2.3. Path optimization Strategy

Direct implementation of the CMA algorithm from Section 2.2 produced non-optimal results. With each iteration, subsequent velocity of the vehicle changes excessively causing eccentric paths around obstacles. This was reduced by applying vector interpolation. By interpolating the velocity vector ($v^*_{n+1}$) halfway ($r$ = 0.5) from the current velocity ($v_n$) and the calculated subsequent velocity ($v_{n+1}$) as in Equation 4.

$$v^*_{n+1} = (1 - r)v_{n+1} + rv_n \tag{4}$$

However, there was still significant angular change at turning points that resulted in excessive motion. Research indicated that efficient optimization strategies include b-splines or Bezier curves. B-splines are superior as they are easier to optimize to minimize path lengths. However, in real-time, they are computationally expensive to evaluate [29]. Bezier curves can be evaluated quickly and can provide an equivalent path by employing strategies presented in [30], [31]. In online pathing, this algorithm virtually predicts steps, and a Bezier curve is plotted through these points with an order ($k$) equal to the steps predicted. Repeated empirical simulated tests helped determine optimal order $k$ = 3; if $k$ > 3 then there is increased probability of collision with obstacles, and if $k$ < 3 the path remains eccentric. The changeover point between two curves is kept common ($P_0$ for subsequent curve is $P_k$ of current curve) to maintain approximate gradient and curvature continuity. The Bezier curve formula in Equation 5 is a function of parameter '$t$'.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 December 2020 doi:10.20944/preprints202012.0307.v1



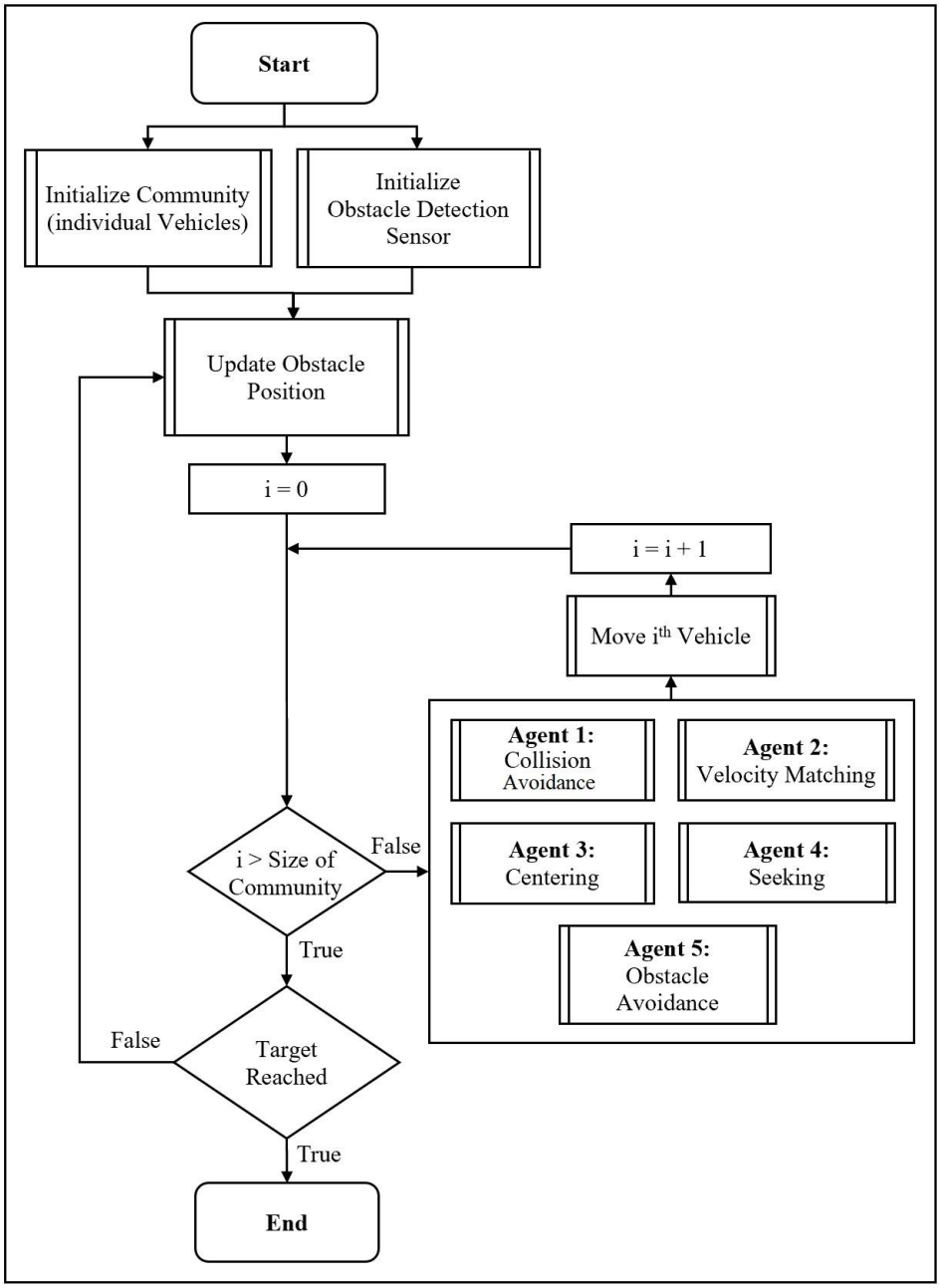


Figure. 3 Final algorithm (also referred to as CMA algorithm or strategy) for path-planning in a dynamic environment

$$P(t) = \sum_{i=0}^{i=k} \binom{k}{i} (1-t)^{k-i} (t)^{i} P_i \quad (5)$$

The CMA algorithm was applied to a static workspace with a singular obstacle in its path. As seen in Figure 4, total path length in this instance was improved by 56%.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 December 2020 doi:10.20944/preprints202012.0307.v1



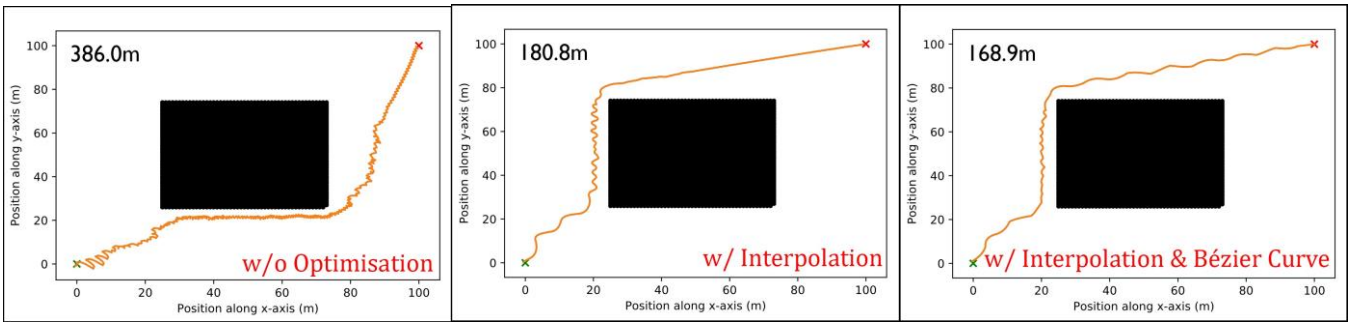


Figure. 4 Reduction in path-length around a single stationary obstacle with application of optimization strategies

Further testing in static conditions proved similar level of improvement. A sample run of this algorithm in a simulated environment can be seen at the YouTube link (click here).

### 2.4. Numerical Verification Method

Since there are no existing strategy that can be applied to an observable and dynamic workspace, there was no way to verify the effectiveness of the formulated strategy directly numerically. Moreover, unless the obstacles are controlled directly (which opposes the nature of the workspace) a controlled comparative analysis *ceteris paribus* cannot be conducted. Generally, strategies found were implemented in non-observable fields [32], [33]. Hence, CMA algorithm was modified to work within a stationary, observable environment so that it can be compared against an established strategy such as A* algorithm. The purpose of this verification strategy is to formulate a conclusion through deductive reasoning. This is since a conclusion through induction is not possible because of the above reasons. If the CMA algorithm performs comparably to A* in a stationary workspace, by extension, it should be able to perform optimally in a dynamic workspace.

This numerical verification process is completely simulated through software. The dependent variable will be the path length formulated through the algorithms. The independent variable is the number of obstacles in the field. This leads to an issue of repeatability and consistency regarding trends produced with the verification. Obstacles can be scattered anywhere within the workspace. This means that for certain number of obstacles results may vary based on arrangement or favorability of their positioning. To reduce bias though such occurrences, the following strategies were employed: 1) workspace was limited to only two-dimensions; 2) obstacle will be place at random at up to five different orientations; 3) the dimensions of the workspace will be fixed (240×180). For a given number of obstacles at a particular orientation in the workspace, three trails will be conducted to observe any variations.

### 2.5. Practical Validation Strategy

To supplement the verification, an experiment was conducted as part of validation. The validation will be a repetition of the tests in the verification but with a practical set-up. The purpose of validation is to determine whether the algorithm can be applied, and if it can provide comparable results to the simulation.

*Obstacle Detection and Computer Vision*

The CMA algorithm relies on two components: the control center and the obstacle detection sensor. The former corresponds to the computer running the algorithm. Many options were considered for the latter, ultimately the decision was to utilize a Microsoft Kinect. The choice was based on multiple factors. Firstly, the algorithm is programmed in C++, and the Kinect has sufficient infrastructure through the Microsoft Kinect SDK optimized for C++. Additionally, the Microsoft Kinect has a relatively low error of ±4cm at maximum range, and a high degree of functionality [34].

The Microsoft Kinect line was discontinued late 2017, however a Kinect V1 (late 2010) module was available for use. It is an RGB-Depth sensor with 640×480px resolution. The infrared (IR) depth sensor operates at 10fps, and it has an ideal range of 1-3m [35]. The IR camera provides vector of 11-bit unsigned integers as a depth map between 0-2047. The open Frameworks library contains both the Kinect SDK as well as OpenCV (computer vision toolbox). This is used to extract metric depth information based on the 11-bit depth map from the Kinect.

The sensor is used to extract the obstacle position and area. To detect obstacles at a distance of 'Z', depth images at positions $\pm\Delta Z$ are obtained and, using the OpenCV kit, the image intersection ('cvAnd' function [36]) using Equation 6 is obtained. The intersection corresponds to the obstacle.

$$Image_{Z+\Delta Z} \cap Image_{Z-\Delta Z} \tag{6}$$

The image is run through the OpenCV's "blob" recognition algorithm to detect object location and area in terms of screen coordinates $(u,v)$. An example of output is shown in Figure 5.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 December 2020 doi:10.20944/preprints202012.0307.v1



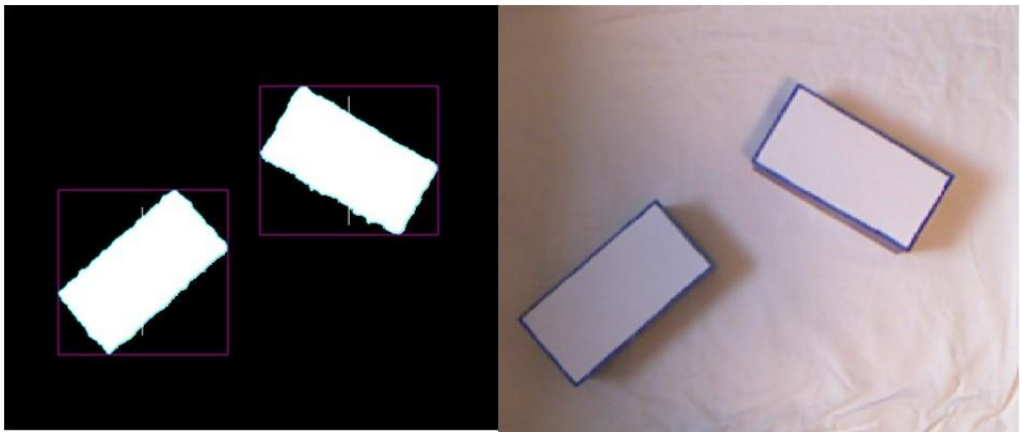

Figure. 5 Usage of computer vision to detect obstacles as 'blobs' in a workspace, with the aid of an XBox 360 Generation 1 Kinect Sensor; (left) OpenCV 'blob' recognition output (right) Obstacles positioned in workspace

The purple rectangular boundary is drawn by the program to indicate the position of the object, the yellow 'plus' indicates effective radius of the object. Using this sensor, the "Initialize Obstacle Detection Sensor" function from Figure 3 is as shown in Figure 6.

With real-world depth values, screen coordinates can be converted to real-world coordinates $(X, Y, Z)$ using matrix transformation as in Equation 7 [37].

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{7}$$

The matrix can then be resolved to Equation 8 to determine '$X$' and '$Y$'.

$$X = \frac{Z(u-c_x)}{f_x}; Y = \frac{Z(u-c_y)}{f_y} \tag{8}$$

Here, '$f_x$' and '$f_y$' correspond to the focal length while '$c_x$' and '$c_y$' correspond to the principal points of the camera. These are intrinsic camera properties, and for the IR camera the published values are in Table 1.

Table 1 Intrinsic properties of Kinect V1 IR camera [38]

| Resolution | 640x488 px |
|---|---|
| $f_x$ | 314.649 |
| $f_y$ | 240.160 |
| $c_x$ | 572.882 |
| $c_y$ | 542.739 |

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 December 2020 doi:10.20944/preprints202012.0307.v1



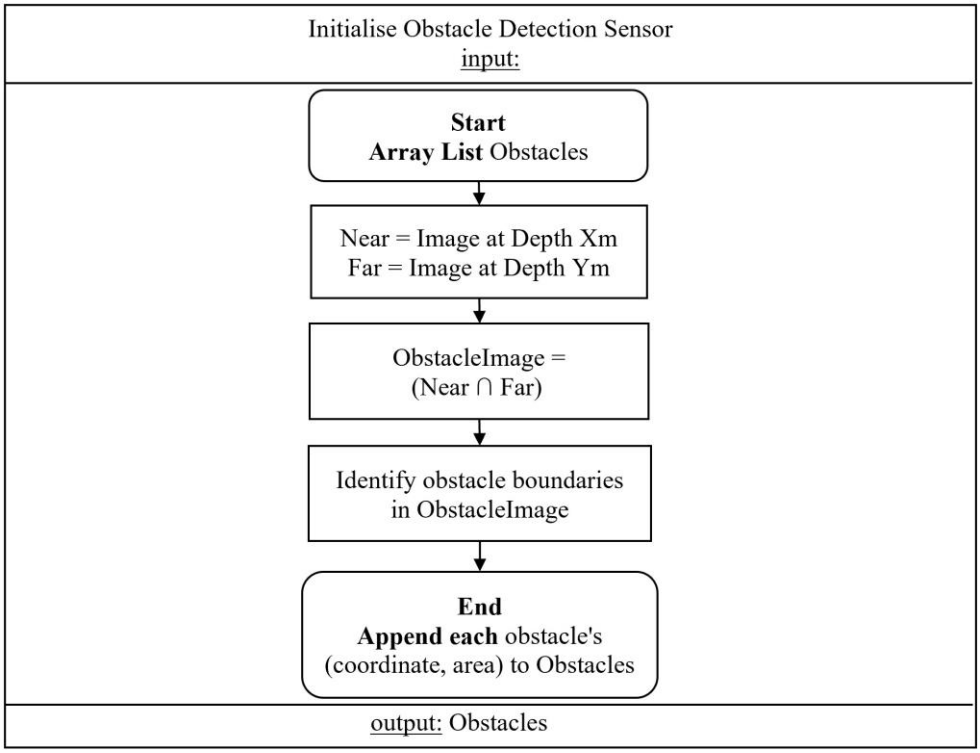


Figure. 6 "Initialize Obstacle Detection Sensor" function as applied in the CMA algorithm

*Experiment Method*

For practical validation, apart from the control center and sensor, an Arduino controlled Vehicle was used. Communication between center and vehicle was through a Bluetooth serial interface sending movement commands. No other sensors are present on the vehicle. All apparatus used are presented in Figure 7.

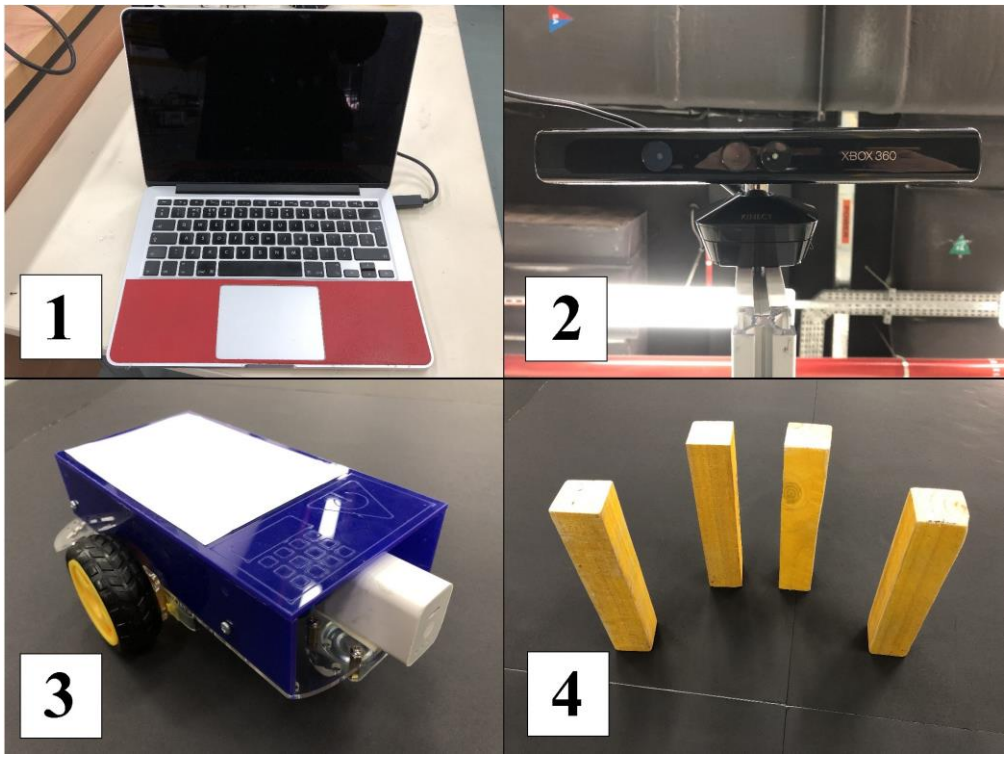


Figure. 7 (1) Control center MacBook Pro 2012; (2) Kinect V1 RGB-Depth sensor; (3) Arduino controlled vehicle with Bluetooth module; (4) Obstacles (25cm$^2$ cross-section)

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 December 2020 doi:10.20944/preprints202012.0307.v1



Since verification was limited to two-dimensions, only one Kinect sensor is required. The Kinect sensor was positioned directly facing the workspace to obtain an overhead view. The work area needed to match the size used for verification (240×180), hence the vertical distance to the camera is 210cm. Kinect is wire-connected to the control center, which computes the depth image to obtain obstacle information from the workspace. The algorithm runs and commandeers the vehicle via Bluetooth.

To differentiate between obstacles and the vehicle, a computer vision program was written that would differentiate objects in the workspace based on elevation difference. The vehicle is at a lower elevation to the obstacles, hence using Equation 6 the vehicle and obstacles can be differentiated. For clarification refer to the side view sketch of the set-up in Figure 8.

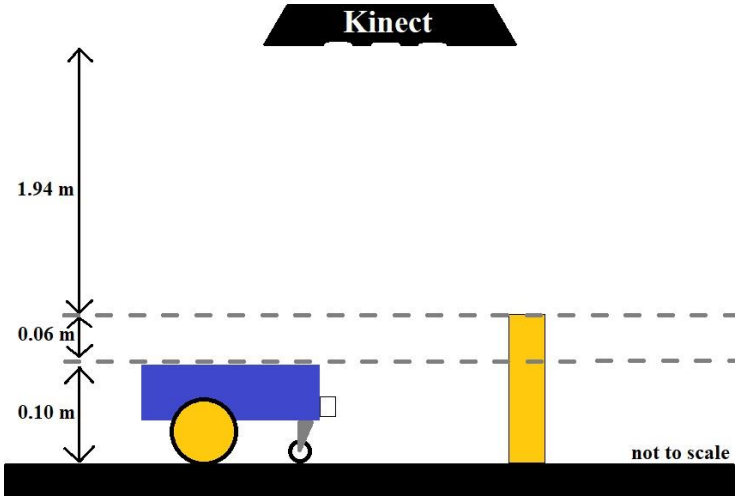


Figure. 8 Side view sketch of the set up with elevation data

There is a clear elevation separation between the vehicle and the obstacles. Thus, using Algorithm 8 the position and area of the vehicle and the obstacles was found. This method was sufficient for validation because the verification was limited to stationary obstacles.

## 3. Extension and Application of Strategy

### 3.1. Simulation in 3D Dynamic Workspace

With the validated algorithm, a 3D simulation was programmed using C++. The obstacles are now 3D shapes with similar dynamic and stochastic properties as Section 2; essentially the workspace properties from before having been re-applied here. Additional complexity was added by allowing elastic collision between obstacle causing sudden velocity changes. The community is set to move from origin, diagonally across the room to the farthest corner of the room. One result from a run of this simulation is shown in Figure 9.

The vehicles are red spheres, while their trail is a red line. Obstacles may pass over the trail if the interference is not in the immediate vicinity of the vehicles. Programming this simulation was complicated, as the additional dimension required viewport manipulation options. Since the program was in C++ using the open Frameworks library, it was possible to apply a mouse-controlled viewport using OpenGL. This could be rotated, expanded, contracted, and translated using traditional CAD mouse controls.

Concurring with the earlier conclusions, the algorithm capably produces an optimal path through this obstacle field. With the additional dimension, it was observed that the pathing was much smoother, and the vehicles seldom had reason to remain stationary. It is assumed in the simulation that the potential obstacles in the path of the vehicles are simple shapes. In fact, the simulation in Section 2 also simplifies the potential obstacles and types of interference. However, since it was applied successfully in a practical 2D workspace, it is inferred that it is also possible to apply this algorithm with drones. This validation could not be performed given the time and funding constraints.

### 3.2. Applications

Given the limitation of this strategy with regards to where it can be applied, it is still relevant in the current market for various uses. Firstly, this strategy can be applied in navigation of emergency response vehicles through traffic and congested regions. Most modern cities have implemented central surveillance cameras in a large scale. Alternatively, it can also be integrated with GPS and satellite systems. These systems provide a complete knowledge of the environment, making it possible to apply this strategy.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 December 2020 doi:10.20944/preprints202012.0307.v1



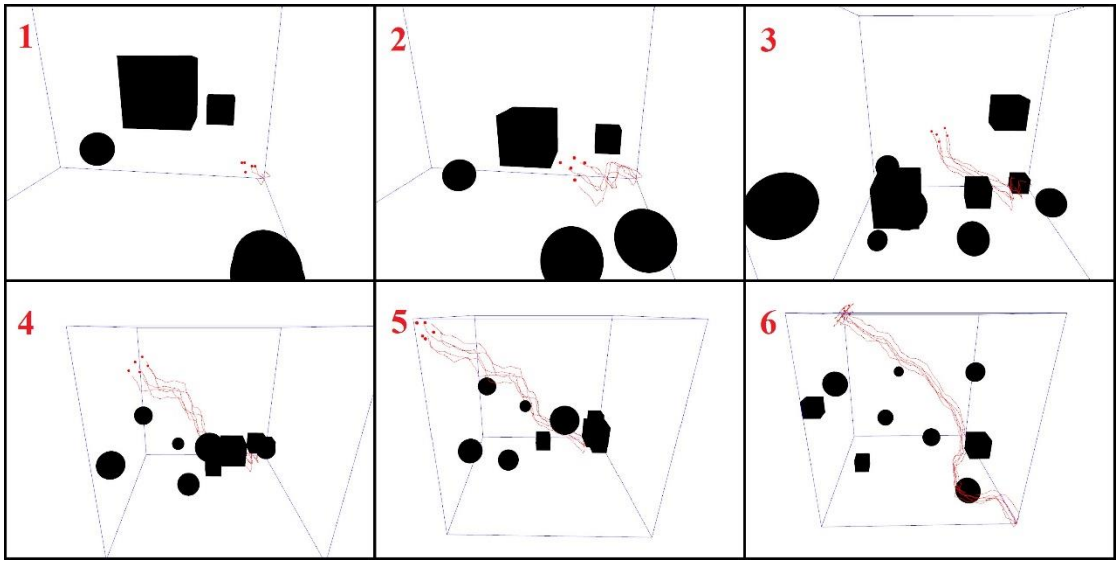


Figure. 9: Example output from one run of the programmed simulation in a 3D workspace with dynamic environment [link]

CMA strategy can also be applied in warehouse management and factory floor traversal. Since factories and warehouses are frequently monitored, this strategy can be very easily integrated into this environment. It will be possible to control a fleet of unmanned ground vehicles (UGV) or drones to transport inventory from point A to B as a cohesive unit, without worrying about interference from other physical factors.

This strategy can also be applied in close room aesthetic drone displays. Although this does not have a significant impact in society. In summary, although this strategy has a limitation in terms of where it can be applied, there is guarantee that it will thrive in that scenario.

## 4. Results and Discussion

Numerical verification of the CMA algorithm's performance against A* algorithm provided promising results. Based on Figure 10, the two path-planning algorithms produce comparative paths with similar lengths for varying number of obstacles.

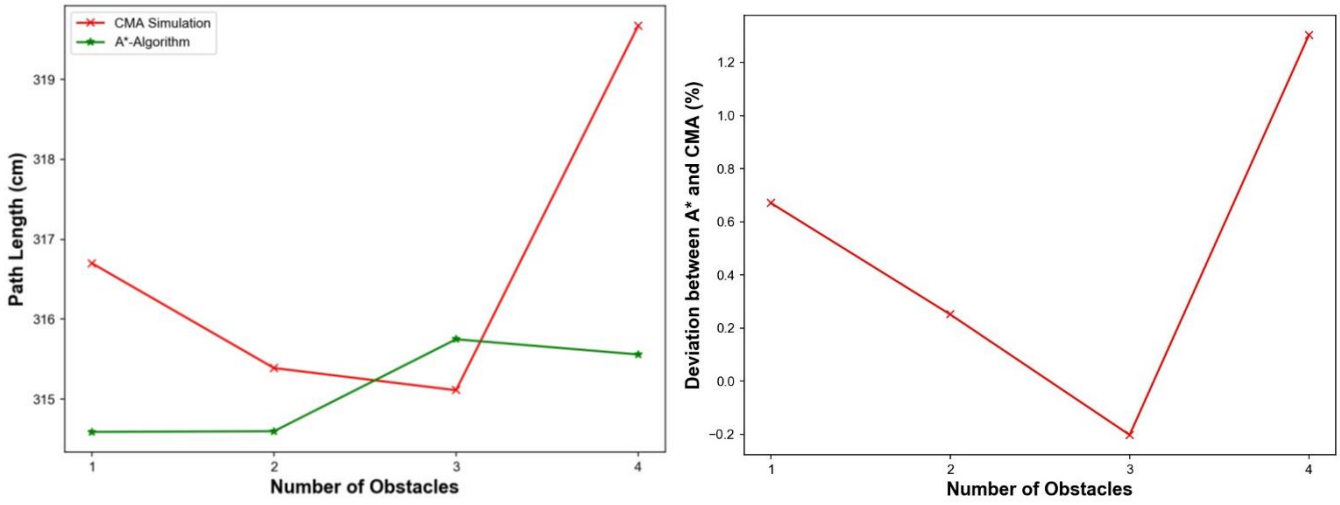


Figure. 10 (left) Path length produced by CMA algorithm and A*-algorithm with increasing number of obstacles in a workspace; (right) Percentage deviation in path length by CMA algorithm and A*-algorithm with increasing number of obstacles in a workspace

The results from this verification indicate that at worst, CMA algorithm produces a path length 1.3% more than that of A*, and at best it was 0.2% less. These results are substantiated through large number of trials and variations; thus, bias was sufficiently minimized. Additional changes in trend can be observed based on increase in field size and obstacle cross-section, however such factors were kept consistent across the numerical verification and practical validation processes. Numerical verification concluded that the CMA algorithm performs comparably to A* algorithm in stationary and two-dimensional workspaces. The only other strategy that was found to fit the same dynamic and stochastic environment as the CMA algorithm was based on

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 December 2020 doi:10.20944/preprints202012.0307.v1



applying A* algorithm in closed intervals. Hence through deductive reasoning it can be concluded that CMA algorithm should provide the most optimal path in a dynamic environment as well.

To further evaluate the effectiveness of the CMA algorithm, the practical validation was undertaken. The results from the validation proved similar for 1-3 obstacles, however some deviation from the simulated solution was observed for greater number of obstacles. Referring to Figure 11, the practical validation of the CMA algorithm produced comparative results with error less than 1% while the number of obstacles were 3 or less.

With 4 obstacles the error between the practical and the CMA algorithm simulation was 5.2% and against A* algorithm it was 6.5%. One reason for some deviation in path lengths can be attributed to the area of the vehicle traversing the obstacle field. The algorithms in simulation assume that the vehicle is a point mass, however in practicality its cross-section must be considered in path planning through tight spaces. In a few cases, while the simulations path planned through a gap between two obstacles, the vehicle could not and had to move around resulting in a longer path.

However, to substantiate a conclusion for the validation, a secondary analysis of the methodology was conducted. It was found that the camera's viewpoint tended to skew the observed workspace. Even with proper calibration and orientation of the Kinect sensor, the objects positioned at the periphery of the workspace were captured at a skewed angle compared to those at the center which were observed head-on. To determine the error resulting from the perspective of the sensor, one obstacle was placed at points in a discrete grid over the workspace. The obstacle area observed by the camera was compared against the actual area and is presented in the surface plot in Figure 12.

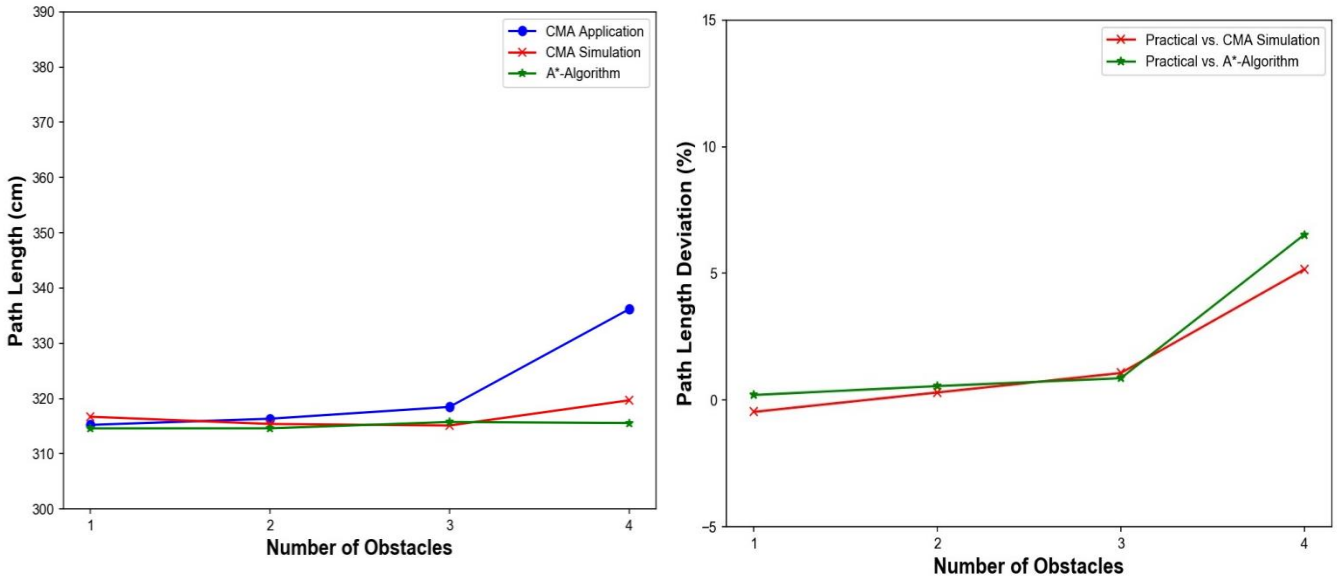


Figure. 11 (left) Path length produced by practical validation compared against CMA algorithm and A*-algorithm with increasing number of obstacles in a workspace; (right) Percentage deviation in path length by practical validation compared against CMA algorithm and A*-algorithm with increasing number of obstacles in a workspace

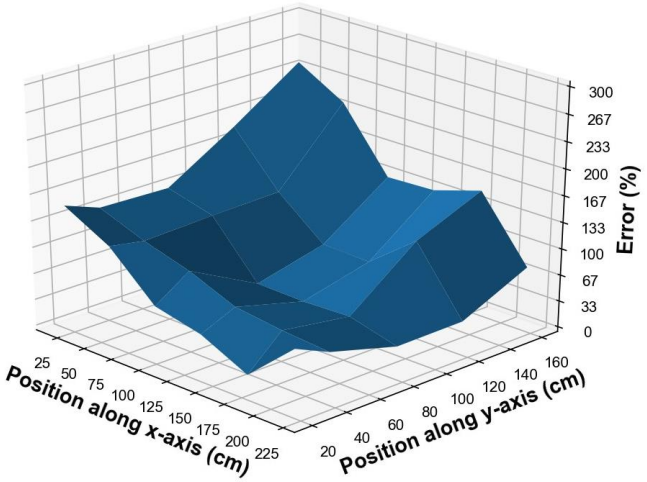


Figure. 12 Percentage error in area of obstacles as observed by overhead Kinect sensor

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 December 2020 doi:10.20944/preprints202012.0307.v1



It was determined that for obstacles in the periphery, the area can be amplified by up to 233%. This is significant, as the original 25cm$^2$ obstacle can be recognized as a 58.25cm$^2$ obstacle. Path planning around this larger obstacle can be eccentric, which results in a much longer path length. Further errors near the center of the workspace are attributed to the age of the first-generation Kinect sensor used. A solution would be to utilize multiple newer sensors at orthogonal perspectives. However, the Kinect sensor product line had been discontinued and additional quantities could not be obtained.

Hence, although the practical validation proved that the algorithm can be applied successfully in two dimensions, there is still more scope for research which will be undertaken in the future. The validation proved that given proper infrastructure for the set-up, the CMA algorithm can be applied in practice to solve the problem of path-planning and optimization. Future works will remodel the experiment completely, utilizing different types of sensors, and applying it to a dynamic environment. Tests in dynamic environments with the current set-up were not repeatable due to the above-mentioned sensor issues, and the very limited computer vision program. Simulations were performed in three-dimensional, stochastic, and dynamic workspaces as in Section 3. The observations from those simulations indicated that the CMA algorithm can is applicable in these scenarios as well. This will also be tested in future research, with drones as the vehicles and obstacles. The final aspect of the algorithm that will be tested in future works, is its ability to coordinate groups of vehicles.

## 5. Future Works

This investigation established the CMA strategy and proved that it works as planned. It is limited to testing the path-planning and optimization capabilities and its applicability in practice. This leaves space for more work. For future investigation, we can 1) explore and fine tune the cooperative aspect of the algorithm; 2) test the obstacle avoidance in dynamic environments; 3) evaluate efficacy in a 3D environment with drones.

During this work, there were multiple barriers that resulted from the lack of time, funding, and other factors. These barriers were highlighted in the report when they were faced. To test the cooperative aspect, a superior control center is needed with ability to communicate with multiple Bluetooth entities. To test the obstacle avoidance fully, a better obstacle recognition strategy needs to be implemented with memory of obstacles so that movement can be tracked and monitored. This requires more advanced computer vision programming. Finally, knowledge of control and programming of drones is needed to perform the final stage of validation. Addressing all these barriers will provide a highly comprehensive, multi-part report that will be a well-rounded evaluation of this strategy.

## 6. Conclusion

This investigation explored a path-planning and optimization strategy for a dynamic and observable workspace. It accommodated cooperation amongst robotic agents and was based on flocking principles. The strategy was verified by comparing against A* algorithm for stationary environments. This was because no comparable algorithms existed that could be used to verify the efficacy of this algorithm in this type of workspace. Verification proved that CMA algorithm is comparable to A* in providing an optimal path. The path length varied between -0.2% to 1.3% for 1-4 obstacles placed randomly. This helps conclude that the CMA algorithm can provide an optimal path-plan in a dynamic workspace. The algorithm was then applied with a practical experiment to validate the algorithm. The experiment had a maximum error of up to 5.2%, which was further evaluated to justify the conclusion. It was found that the Kinect sensor used for obstacle detection had some errors due to its age. Additional avenues for investigation were explored and were suggested for future works. The remaining aspects to this algorithm include testing group behavior and path planning in three-dimensional workspaces. A simulation was programmed to evaluate effectiveness in three-dimensions and proved successful, however it needs to be validated with drones. Overall, this investigation was successful in establishing the effectiveness of this novel path-planning and optimization algorithm set for dynamic environments.

## 7. Appendix 1: Algorithms

This appendix lists out all the algorithms used in our study.

**Algorithm 1:** Update Obstacle Position

**input :** NewObstacles ← new obstacles' data

```
// find minimum change in position of obstacles
ref ← min(NewObstacles.position-Obstacles.position)
for i ← 0 to size of Obstacles do
    weight ← (NewObstacles.[i].position − Obstacles.[i].position) / ref
    Obstacles.[i].region ← Obstacles.[i].region × weight
end
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 December 2020 doi:10.20944/preprints202012.0307.v1



**Algorithm 2:** Move $i^{th}$ Vehicle

```
input : i
cv ← Community [i]
v1 ← Agent1(cv); v2 ← Agent2(cv)
v3 ← Agent3(cv); v4 ← Agent4(cv)
v5 ← Agent5(cv)
v ← (v1 × c1) + (v2 × c2) + (v3 × c3) + (v4 × c4) + (v5 × c5)
v ← v_max × v / magnitude(v)
cv.position ← cv.position + v
```

**Algorithm 3:** Agent 1: Collision Avoidance

```
input  : cv
output : v1
v1 ← (0, 0)
for j ← 0 to size of Community do
    if Community [j] is not cv then
        difference ← Community [j].position- cv.position
        if magnitude(difference) less than allowance then
            v1 ← v1- difference
        end
    end
end
v1 ← v1 / magnitude(v1)
```

**Algorithm 4:** Agent 2: Velocity Matching

```
input  : cv
output : v2
v2 ← (0, 0)
for j ← 0 to size of Community do
    if Community [j] is not cv then
        v2 ← v2 + Community [j].velocity
    end
end
v2 ← v2 / (size of Community − 1) − cv.velocity
v2 ← v2 / magnitude(v2)
```

**Algorithm 5:** Agent 3: Centering

```
input  : cv
output : v3
v3 ← (0, 0)
for j ← 0 to size of Community do
    if Community [j] is not cv then
        v3 ← v3 + Community [j].position
    end
end
v3 ← v3 / (size of Community − 1) − cv.position
v3 ← v3 / magnitude(v3)
```

**Algorithm 6:** Agent 4: Seeking

```
input  : cv
output : v4
v4 ← end- cv.position
v4 ← v4 / magnitude(v4)
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 December 2020 doi:10.20944/preprints202012.0307.v1



**Algorithm 7:** Agent 5: Obstacle Avoidance

```
input  : cv
output : v5
v5 ← (0,0); for j ← 0 to size of Obstacles do
    if cv.position is in Obstacles.[j].region then
        v5 ← v5 + cv.position- Obstacles.[j].position
    end
end
v5 ← v5 / magnitude(v5)
```

**Algorithm 8:** Obstacle and Vehicle Detection

```
// obtain depth images of vehicle at two locations
DIVehicleNear ← getDepthPixels(depth = 2.00)
DIVehicleFar ← getDepthPixels(depth = 2.05)

// obtain depth images of obstacles at two locations
DIObsNear ← getDepthPixels(depth = 1.94)
DIObsFar ← getDepthPixels(depth = 1.97)

// evaluate intersection to separate the vehicle and obstacles
DIVehicle ← cvAnd(DIVehicleNear, DIVehicleFar)
DIObs ← cvAnd(DIObsNear, DIObsFar)

// determine 'blobs' using contour recognition
VehicleContours ← findContours(DIVehicle)
ObstaclesContours ← findContours(DIObs)

// obtain position and area of vehicle and obstacles
Vehicle.position ← VehicleContours.blobs [0] .centroid
Vehicle.area ← VehicleContours.blobs [0] .area

for i ← 0 to size of ObstaclesContours do
    Obstacles [i].position ← ObstaclesContours.blobs [i].centroid
    Obstacles [i].area ← ObstaclesContours.blobs [i].area
end
```

## 8. References


[1] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *Knowl. Eng. Rev.*, vol. 10, no. 2, pp. 115–152, 1995, doi: 10.1017/S0269888900008122.

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed., vol. 136. Upper Saddle River, New Jersey: Prentice Hall, 2016.

[3] S. P. Singh and M. K. Tiwari, "Intelligent agent framework to determine the optimal conflict-free path for an automated guided vehicles system," *Int. J. Prod. Res.*, vol. 40, no. 16, pp. 4195–4223, 2002, doi: 10.1080/00207540210155783.

[4] Z. Wang and S. Zlatanova, "Multi-agent based path planning for first responders among moving obstacles," *Comput. Environ. Urban Syst.*, vol. 56, pp. 48–58, 2016, doi: 10.1016/j.compenvurbsys.2015.11.001.

[5] R. A. Brooks, "A Robust Layered Control System For A Mobile Robot," *IEEE J. Robot. Autom.*, vol. 2, no. 1, pp. 14–23, 1986, doi: 10.1109/JRA.1986.1087032.

[6] N. Jennings, "Specification and implementation of a belief-desire-joint-intention architecture for collaborative problem solving," *Int. J. Intell. Coop.*, vol. 2, no. 3, pp. 289–318, 1993, doi: 10.1142/S0218215793000137.

[7] A. MacWan, G. Nejat, and B. Benhabib, "Target-motion prediction for robotic search and rescue in wilderness environments," *IEEE Trans. Syst. Man Cybern. Part B Cybern.*, vol. 41, no. 5, pp. 1287–1298, 2011, doi: 10.1109/TSMCB.2011.2132716.

[8] A. Macwan, J. Vilela, G. Nejat, and B. Benhabib, "A Multirobot Path-Planning Strategy for Autonomous Wilderness Search and Rescue," *IEEE Trans. Cybern.*, vol. 45, no. 9, pp. 1784–1797, 2015, doi: 10.1109/TCYB.2014.2360368.

[9] T. A. V. Teatro, J. M. Eklund, and R. Milman, "Nonlinear Model Predictive Control for Omnidirectional Robot Motion Planning and Tracking With Avoidance of Moving Obstacles," *Can. J. Electr. Comput. Eng.*, vol. 37, no. 3, pp. 151–156, 2014, doi: 10.1109/CJECE.2014.2328973.

[10] J. Clemens, T. Reineking, and T. Kluth, "An evidential approach to SLAM, path planning, and active exploration," *Int. J. Approx. Reason.*, vol. 73, pp. 1–26, 2016, doi: 10.1016/j.ijar.2016.02.003.

[11] D. Silver, "Cooperative pathfinding," *Proc. First Artif. Intell. Interact. Digit. Entertain. Conf.*, pp. 117–122, 2005.

[12] S. M. Persson and I. Sharf, "Sampling-based A∗ algorithm for robot path-planning," *Int. J. Robot. Res.*, vol. 33, no. 13, pp. 1683–1708, 2014, doi: 10.1177/0278364914547786.

[13] A. Pandey, R. K. Sonkar, K. K. Pandey, and D. R. Parhi, "Path Planning Navigation of Mobile Robot With Obstacles Avoidance Using Fuzzy Logic Controller," *Int. Conf. Intell. Syst. Control*, pp. 36–41, 2014, doi: 10.1109/ISCO.2014.7103914.

[14] X. B. Hu, M. Wang, M. S. Leeson, E. A. Di Paolo, and H. Liu, "Deterministic agent-based path optimization by mimicking the spreading of ripples," *Evol. Comput.*, vol. 24, no. 2, pp. 319–346, 2016, doi: 10.1162/EVCO_a_00156.

[15] V. Krukhmalev and V. Pshikhopov, *Genetic Algorithms Path Planning*. Elsevier Inc., 2017.

[16] E. Babu, S. Yesudasan, and S. Chacko, "Cymatics Inspired Self-Cleaning Mechanism for Solar Panels," 2020.

[17] R. D. Averett, S. Yesudasan, T. Scogin, and M. L. Walker, "Finite Element Analysis of Magnetic Microparticle Induced Strain on a Fibrin Matrix due to the Influence of an Electromagnetic Field," *ArXiv Prepr. ArXiv*, vol. 1611, 2016.

[18] V. Hotchandani, B. Mathew, S. Yesudasan, and S. Chacko, "Thermo-hydraulic characteristics of novel MEMS heat sink," *Microsyst. Technol.*, pp. 1–13, 2020.

[19] A. Mohammed, S. Yesudasan, and S. Chacko, "A multilayered photonic emitter for high-performance daytime radiative cooling," *Microsyst. Technol.*, pp. 1–15, 2020.

[20] B. Noronha, S. Yesudasan, and S. Chacko, "Static and Dynamic Analysis of Automotive Leaf Spring: A Comparative Study of Various Materials Using ANSYS."

[21] Y. Sumith and S. Chacko, "Dynamic analysis of a turbo-alternator shaft using finite element method," 2009.

[22] S. Yesudasan and S. Chacko, "Fast Local Pressure Estimation for Two Dimensional Systems From Molecular Dynamics Simulations," in *ASME Power Conference*, 2018, vol. 51401, p. V002T10A004.

[23] M. Phillips and M. Likhachev, "SIPP: Safe interval path planning for dynamic environments," in *Proceedings - IEEE International Conference on Robotics and Automation*, May 2011, pp. 5628–5635, doi: 10.1109/ICRA.2011.5980306.

[24] M. Ohradzansky, H. E. Alvarez, J. Keshavan, B. N. Ranganathan, and J. S. Humbert, "Autonomous Bio-Inspired Small-Object Detection and Avoidance," *2018 IEEE Int. Conf. Robot. Autom. ICRA*, pp. 3442–3447, 2018, doi: 10.1109/ICRA.2018.8461156.

[25] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *ACM SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 25–34, 1987, doi: 10.1145/37402.37406.

[26] C. Parker, *Boids Pseudocode*. 2013.

[27] M. I. Ribeiro, "Obstacle avoidance," *J. Vis.*, vol. 10, no. 11, pp. 1–14, 2005, doi: 10.1167/10.11.17.

[28] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *Int. J. Robot. Res.*, vol. 5, no. 1, pp. 90–98, 1986, doi: 10.1177/027836498600500106.

[29] F. Andersson and B. Kvernes, "Bezier and B-spline Technology," *Master Sci. Thesis*, p. 58, 2003.

[30] L. Piegl and W. Tiller, *The NURBS Book*, 2nd ed. New York: Springer-Verlag, 1997.

[31] R. Cimurs, J. Hwang, and I. H. Suh, "Bezier curve-based smoothing for path planner with curvature constraint," *Proc. - 2017 1st IEEE Int. Conf. Robot. Comput. IRC 2017*, pp. 241–248, 2017, doi: 10.1109/IRC.2017.13.

[32] P. Svec and S. K. Gupta, "Automated planning logic synthesis for autonomous unmanned vehicles in competitive environments with deceptive adversaries," *Stud. Comput. Intell.*, vol. 341, pp. 171–193, 2011, doi: 10.1007/978-3-642-18272-3_12.

[33] E. Raboin, P. Švec, D. S. Nau, and S. K. Gupta, "Model-predictive asset guarding by team of autonomous surface vehicles in environment with civilian boats," *Auton. Robots*, vol. 38, no. 3, pp. 261–282, 2014, doi: 10.1007/s10514-014-9409-9.

[34] M. Barczyk and S. Bonnabel, “Towards realistic covariance estimation of ICP-based Kinect V1 scan matching: The 1D case,” in *Proceedings of the American Control Conference*, 2017, pp. 4833–4838, doi: 10.23919/ACC.2017.7963703.

[35] J. Han, L. Shao, D. Xu, and J. Shotton, “Enhanced computer vision with Microsoft Kinect sensor: A review,” *IEEE Trans. Cybern.*, vol. 43, no. 5, pp. 1318–1334, 2013, doi: 10.1109/TCYB.2013.2265378.

[36] OpenCV Dev Team, *Operations on Arrays — OpenCV 2.4.13.7 documentation*. 2018.

[37] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge: Cambridge University Press, 2004.

[38] J. L. Blanco, *Kinect calibration – MRPT*. 2013.