

## Efficient method for finding nearest neighbors in flocking behaviors using k-dimensional trees

Marwan Al-Tawil<sup>1</sup>, Moh'd Belal Al-Zoubi<sup>1</sup>, Omar Y. Adwan<sup>1,2</sup>, Ammar Al-Huneiti<sup>1</sup>,  
Reem Q. Al Fayez<sup>1</sup>

<sup>1</sup>Department of Computer Information Systems, King Abdullah II School of Information Technology, The University of Jordan, Amman, Jordan

<sup>2</sup>Department of Computer Science, Faculty of Information Technology, Al-Ahliyya Amman University, Amman, Jordan

### Article Info

#### Article history:

Received Nov 21, 2022

Revised Jan 17, 2023

Accepted Jan 30, 2023

#### Keywords:

Algorithms

Flocking behavior

K-dimensional trees

Nearest neighbor

Simulation

### ABSTRACT

Flocking is a behavior where a group of objects travel, move or collaborate together. By learning more about flocking behavior, we might be able to apply this knowledge in different contexts such as computer graphics, games, and education. A key steppingstone for understanding flocking behavior is to be able to simulate it. However, simulating behaviors of large numbers of objects is highly compute-intensive task because of the n-squared complexity of nearest neighbor for separating n objects. The work in this paper presents an efficient nearest neighbor method based on the k-dimensional trees (KD trees). To evaluate the proposed approach, we apply it using Unity-3D game engine, together with other conventional nearest neighbor methods. The Unity-3D game simulation engine allows users to utilize interaction design tools for programming and animating flocking behaviors. Results showed that the proposed approach outperform other conventional nearest neighbor approaches. The proposed approach can be used to enhance digital games quality and simulations.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



### Corresponding Author:

Marwan Al-Tawil

Department of Computer Information Systems, King Abdullah II School of Information Technology

The University of Jordan

Amman, Jordan

Email: m.altawil@ju.edu.jo

## 1. INTRODUCTION

Animal behavior has always been a source of study and amazement for mankind [1], [2]. Flocking is a behavior in which several objects work or move together as coherent entities [3] or behave in a unified manner while changing their direction and shape [4]. Examples of animal flocking behavior include birds (flies in swarms), sheep (moving as herds), and fish (swimming in schools) [5]. In recent years, studying and understanding animal flocking behavior through simulations has attracted researchers from various scientific and engineering fields [6]. Simulating flocking behavior has been widely studied in computer graphics [3], games [7], [8], mobile networks [9], road design [10] and education [11], [12]. Such simulations enable users to test complex scenarios that have no impact on a real environment. To produce flocking behavior in groups of computer characters (objects), three rules in section 2.1 were applied to all objects in the group with convincing flocking behavior results: Alignment (i.e. steer to mean heading of mates in a flock), Cohesion (i.e. steer to mean position of flock mates), and Separation (i.e. steer to avoid crowded mates) [3], [8], [13].

As the object arrangement in a flock change constantly, every object in the flock must update its view by cycling through all objects in the flock to collect the required data for each object. This is computationally expensive, particularly when the number of objects rapidly increases. Simulating large numbers of objects in

real time is a challenging task, particularly when the objects are realistically separated, and collisions are avoided. This requires the development of intelligent approaches to facilitate the separation of flocking behavior. A common method for approaching the separation rule is to use the nearest neighbor (NN) algorithm [4]. This algorithm calculates the distance between object  $x$  and every other object in the training set. The NN algorithm sorts the calculated distances and then selects the  $k$  samples closest to the object  $x$  [14]. However, the complexity of the NN algorithm was  $O(n^2)$ . This is because of the separation rule in which the distance of every object from other objects in the entire flock is calculated. Such high complexity makes it inefficient for use in modern applications, such as games [15], crowd mobility [16], mobile network mobility [17], and aerial robots [18]. Therefore, an optimized NN is required to reduce complexity when performing the separation rule for flocking behavior.

The work in [4] utilized the k-nearest neighbor (KNN) method to compute the nearest neighbors in Flocking Behavior, where the nearest object is found by the majority of flock objects within  $k$  adjacent neighbors. However, two key issues are related to KNN [19]. The first concerns choosing parameter  $k$ , a user-predefined parameter that represents the number of nearest neighbors. A straightforward method for determining the best  $k$ , is cross-validation (i.e., trying several  $k$  values and choosing  $k$  with the highest performance). The second issue with the KNN is its high computational complexity for large sets of objects. Therefore, the work in [19] introduced the extended nearest neighbor (ENN) method. Unlike the classification method in KNN, in which information about nearby neighbors is considered, the ENN method considers information from all available objects. The performances of the KNN and ENN methods were discussed in [20]. The results showed that the ENN method is less prone to errors, although both have a high computational complexity for large sets. The partial distance (PD) algorithm proposed in [21] uses a premature exit condition during the search process to terminate the distance calculation at an early stage. However, its efficiency depends on the initial distance (i.e. it is less useful for larger distances). For this purpose, an efficient method for identifying initial distances to the PD algorithm was proposed in [22]. An NN method based on PD was proposed in [23] to provide an enhanced version of the partial distance approach (EPD).

This paper presents an enhanced version of the partial-distance approach proposed in [23] to solve the flocking fish problem. We propose an efficient NN method based on the k-dimensional trees (KD trees) algorithm. The KD trees algorithm is a multidimensional binary tree, which is a specific storage structure that efficiently represents training data [14]. The proposed approach, together with other conventional NN methods, was applied using a Unity-3D game engine. Unity-3D allows users to use a set of powerful animation and interaction design tools to visually program and animate flocking behavior. The experimental results show that the performance of the proposed KD trees is better than that of the conventional NN methods. The remainder of this paper is organized as follows. Section 2 provides information related to the flocking simulation together with the main algorithms used in this paper. Section 4 describes the implementation of the proposed method. Section 5 presents the results, and section 6 concludes the paper.

## 2. METHOD

An efficient NN method based on the KD trees algorithm was proposed in this paper. Flocking simulation models and main classification algorithms were presented to better understand the research context. We also describe the implementation of the proposed method using Unity-3D game engine.

### 2.1. Flocking simulation models

Simulations are important for understanding flocking behavior [5]. The first flocking-behavior simulation (called 'Birds') was performed in 1986 by Craig Reynolds [3]. The simulation program was made after the simple agents in the system. Three conducts [24] (i.e. steering rules) were used in the simulation to determine how birds would move, namely cohesion, alignment, and separation. Another flocking simulation model was proposed by Heppner in 1990 [25]. This model consists of three conducts: Homing (every object aims to stay in the roosting area), Velocity regulation (every object aims to fly at a particular predefined flight speed, which aims to return to that speed if perturbed), and Interaction (two flock entities try to move apart when they are too close to one another). This paper focuses on the first simulation model proposed by Reynolds [3]. The Reynolds model is the simplest model for simulating animal behavior, such as that of birds. It is the most widely used simulation model [5]. Figures 1(a) to 1(c) illustrates the three rules of Reynolds' model [8].

- Cohesion rule: Figure 1(a) [8] shows the cohesion rule, which implies that all flock objects remain together in a group without going on a separate way away from the group. To obtain this rule, every object must steer towards the average position of its neighbors in the flock (i.e., steer towards the center of a mass). Assume we have  $N$  objects ( $o_1, o_2, \dots, o_N$ ) and the position of an object is denoted by  $o.position$ , then the 'center of mass'  $c$  of all  $N$  objects is given by (1).

$$c = (o_1.position + o_2.position + \dots + o_N.position)/N \quad (1)$$

- Alignment rule: Figure 1(b) [8] illustrates the alignment rule, which implies that all objects in a flock generally head in the same direction. Every object in the flock must steer towards a heading that is determined by the unit's velocity vector [5] and is equal to the average heading of the object's neighbors. Normalizing the velocity vector for every unit gain unit heading vector makes the steering force a linear function of the angle between the current unit heading and the average heading of its neighbors.
- Separation rule: Figure 1(c) [8] shows the separation rule that aims to avoid collisions between objects. Although objects may try to get closer to each other because of the alignment and cohesion rules, the separation rule aims to maintain objects at a minimum distance from each other.

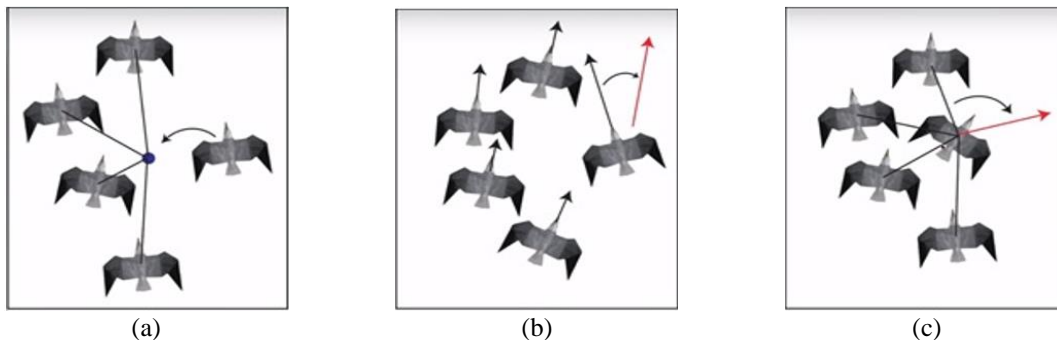


Figure 1. Shows three flocking rules: (a) cohesion, (b) alignment, and (c) separation [8]

## 2.2. Nearest neighbor

The NN algorithm is a well-known supervised machine learning algorithm used for classification and regression problems in different contexts. An NN was proposed in 1967 by Cover and Hart [26]. The identification of the nearest neighbors depends on the distance between the tested and training data samples. Given a query vector  $x_0$  and a set of  $N$  labeled instances, the aim is to predict the class label of  $x_0$  on a predefined set of classes. One way to find the NN efficiently is to conduct a three-dimensional bucket sort and then check the neighbors' adjacent buckets. The bucket positions of the objects with excessive deviation from the center of the bucket were adjusted incrementally as the buckets were transformed along with the flock. The bucket size involves a time-space trade-off: the smaller the buckets, the more buckets are needed, but the fewer members per bucket on average. Nevertheless, the  $O(n^2)$  problem is not completely eliminated because of the worst-case distributions [4], [5], [27]–[29].

## 2.3. KD trees

The KD trees algorithm is a multidimensional generalization of binary trees designed to handle spatial data in a simple manner [14], [30]. The root of the KD trees represents the  $k$ -dimensional space that contains a set of data points. Every node is a subset of the data points. Each nonterminal node has two child nodes, left and right, which are obtained by partitioning the subset of the node into two parts using a hyperplane orthogonal to one of the  $k$  coordinates axes (called the discriminating axis). The position of the hyperplane was selected such that each child contained approximately half of the parent's data points. The discriminating axis is determined by selecting the most dispersed coordinates. Each terminal node defines a cell (called a bucket) in a multidimensional space containing a subset of data points with bucket size less than a predefined threshold. To find the nearest neighbor for a query point ( $q$ ), we first find a bucket that contains  $q$  by traversing the tree. We then find the nearest point to  $q$  in the bucket. This will result in finding a "candidate" nearest neighbor, but there is still the possibility of a nearest point. Therefore, we traverse backward and start finding the distance from the other points in the neighboring boxes. The point with the nearest distance to  $q$  is the nearest neighbor that we are looking for. The KD trees data structure provides an effective method for examining the objects closest to the query objects, thereby significantly reducing the computation required to find the best matches [30]. The complexity of the KD trees is  $O(K \log n)$  (expected) lookup times for the  $K$  nearest objects to a query object [31], [32]. Based on [32], the object nearest to the query object is determined by applying the following steps,

- Start by exploring the bucket that contains the query object.
- Compute distance to each other objects at bucket.

- Backtrack and find the nearest neighbor to the query object by finding the point with the shortest distance in adjacent buckets.

## 2.4. Implementation

Computer game developers usually face the problem of developing games that are fun, without having to program directly in low-level languages [33]. For this, game engines with graphical editors were developed not only to create visual artwork but also to animate it and scripting interactions with users. Several game engines have recently become available for educational and research purposes. Among several game engines, such as unreal engine, CryEngine, and Unity-3D, we utilized the latter in this research. In [8], de Byl and Penny used a Unity-3D game engine with the C# programming language to develop a computer game that simulated fish flocking behavior. Figure 2 shows a flock of 50 fish simulated by [8]. The figure shows a clear separation between the fish.



Figure 2. A flock of 50 fish simulated by [8]

Fish flocking in [8] was implemented using the Unity-3D game engine. Subsequently, we implement our approach to enhance the algorithm [8]. The Unity-3D game engine provides important features for studying flocking behavior, such as an easy-to-use graphical user interface for animating objects and controlling the interaction between objects. Furthermore, the Unity-3D game engine has a strong professional community that supports its progress, and is freely available for education and research. Figure 3 shows a snapshot (in C#) of how the NN computes the distances between fish objects. The figure shows that we have a value for the neighbor distance, which is the maximum distance that the fish need to be in order for them to flock. If they are outside this distance from each other, they will not take notice of each other, so they will only go to flock if they are near each other. To apply the separation rule, each fish needs to know all the other fish. An avoidance vector (vavoid) was created to avoid any nearby body. If we obtain within a small distance (1.0 unit), we want to avoid it; therefore, we calculate the (vavoid) vector as a vector that is facing off in the other direction. Our focus is on the (Vector3.Distances) function in Figure 3, which computes distances between fish 60 times per second with a complexity of  $O(n^2)$ . However, this is a time-consuming process. To reduce this time, we applied KD trees with a complexity of  $O(K \log n)$  lookup times for the  $K$  nearest objects to a query object.

```

1  GameObject[] fish;
2  float neighborDistance = 2.0f
3  float dist;
4  foreach (GameObject go in fish)
5  {
6      if(go != this.gameObject)
7      {
8          dist = Vector3.Distance(go.transform.position, this.transform.position);
9          if (dist <= neighborDistance )
10         {
11             if (dist < 1.0f)
12             {
13                 vavoid = vavoid + (this.transform.position - go.transform.position);
14             }
15         }
16     }
17 }

```

Figure 3. A snapshot of how the NN works to compute the distances between fish objects

### 3. RESULTS AND DISCUSSION

To examine the effectiveness of the KD trees approach, fish flocking in [8] was implemented together with the conventional NN [19], PD [22] and EPD [23] approaches using the Unity-3D game engine. The frames per second (FPS) were measured using different numbers of objects. Figures 4(a) and 4(b) show screenshots of the simulation results for the conventional nearest neighbor NN and the KD trees approach, respectively, when the number of objects is  $N=100$  (minimum number of objects). The performance of the conventional NN in Figure 4(a) provided a lower FPS (111.2) than that of the KD trees (123.6 FPS), as shown in Figure 4(b). Furthermore, Figures 5(a) and 5(b) show screenshots of the simulation results for the conventional nearest neighbor NN and the KD trees approach, respectively, when the number of objects is  $N=1000$  (the highest number of objects). The Results show that the performance of the conventional NN (9.8 FPS) in Figure 5(a) is much lower than that of the KD trees approach (102.2 FPS), as shown in Figure 5(b).

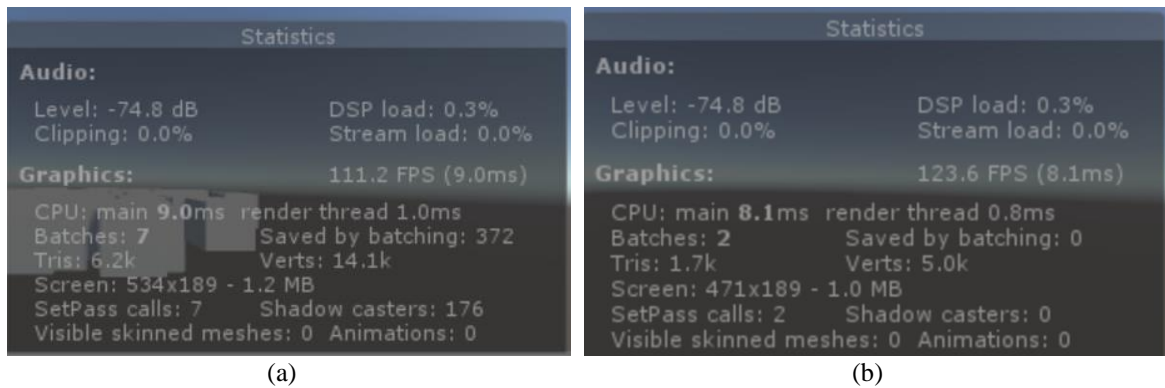


Figure 4. Simulation results representing FPS values when the number of objects ( $N=100$ ) for: (a) the conventional NN approach and (b) the proposed KD approach



Figure 5. Simulation results representing FPS values when the number of objects ( $N=1000$ ) for: (a) the conventional NN approach and (b) the proposed KD approach

Figure 6 shows the graphical performance of all the methods (average results of 10 runs). The results show that the performance of the KD trees approach proposed in this study is better than that of other approaches. The KD trees approach provided higher FPS rates compared to NN, PD, and EPD, while increasing the number of objects. The FPS using the KD trees approach remained at the same level for 100 and 200 objects and then slightly decreased as the number of objects increased, whereas the FPS rates for other approaches dropped at 200 and 600 object sizes. This indicates that the KD trees approach provides a more stable and consistent gaming quality. Furthermore, the NN, PD, and EPD approaches provided FPS rates lower than 60 compared with the KD trees, which maintained FPS rates above 80, while the number of objects increased. This shows that using KD trees is better for interactive games compared to other approaches that can suffer from noticeable motion "flickering" artifacts. To test whether there was a significant difference between the performance of the KD trees approach and other approaches, the Mann-Whitney U test was applied in Table

1. Notably, the difference between the KD method and NN, PD, and EPD approaches was significant ( $P < 0.05$ ), and the difference was higher than that of the EPD method ( $P < 0.05$ ); this difference was positive (i.e. median of  $KD >$  medians of NN, PD, and EPD).

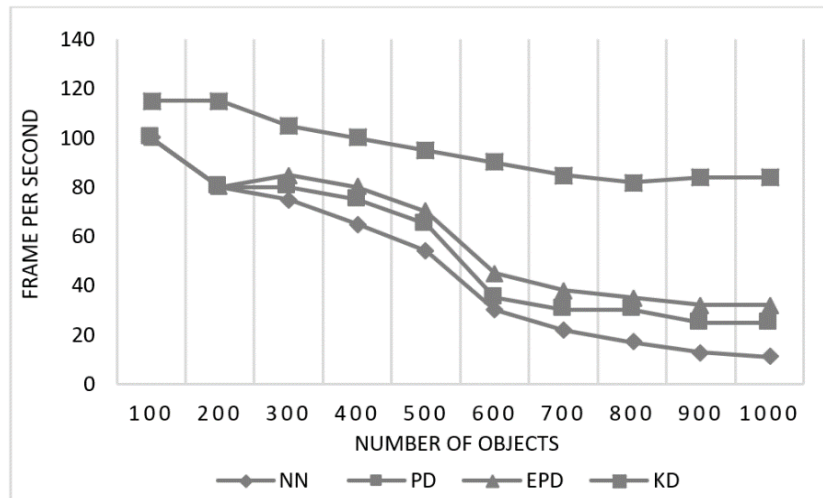


Figure 6. Graphical results (FPS) of the fish flock

Table 1. Statistically significant differences of the performance values in Figure 6 (Mann-Whitney, 1-tail,  $N_a=N_b=10$ , Median of  $KD=92.5$ )

Difference in performance values	Z-score	p	Median values
$KD > NN$	3.25	$P < 0.001$	$NN=42$
$KD > PD$	3.25	$P < 0.001$	$PD=50$
$KD > EPD$	2.985	$P < 0.05$	$EPD=57.5$

Furthermore, the Pearson correlation coefficient was applied to test the correlation between the number of objects and FPS for each approach in Table 2. The results showed a strong negative correlation between each of the given flocking methods and the number of objects in the training set ( $-1 < R < 0$ ). Hence, the performance of the algorithms (i.e., FPS) is influenced by the number of objects in the flock.

Table 2. Pearson correlation coefficient R-values

Approach	R-value
No of objects $>$ KD	-0.95
No of objects $>$ NN	-0.97
No of objects $>$ PD	-0.95
No of objects $>$ EPD	-0.95

#### 4. CONCLUSION




Virtual reality has been used in learning for the last two decades, such as immersive technology that focuses on increasing interaction with the world to aid learners in learning new knowledge. Learners can create scenarios that are not possible in the real world and test the results. For example, learners interested in the fish industry can use digital technologies to create scenarios to better understand and learn how fish behave in flocks. Flocking behaviors have been applied in different yet related research fields such as gaming and computer vision. However, utilizing a large number of objects in a real-time simulation environment is highly demanding in terms of the computing power. This is because of the high NN complexity ( $n$  squared) required to separate the  $n$  objects. We propose an efficient flocking algorithm and its application to fish flocking behavior. The KD trees method for computing the nearest neighbors was presented. The experimental results showed that the proposed algorithm provided better results than other algorithms when applied to the flocking fish problem. To evaluate the proposed approach, we applied it together with other conventional NN methods using the Unity-3D game engine, which allows users to visually program and animate flocking behavior. The results demonstrate that the proposed approach outperforms the conventional NN approaches.

## REFERENCES




- [1] A. Kifouche and A. Guessoum, "Tracking times in temporal patterns embodied in intra-cortical data for controlling neural prosthesis an animal simulation study," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 10, no. 5, pp. 4721–4737, 2020, doi: 10.11591/ijece.v10i5.pp4721-4737.
- [2] L. E. Beaver and A. A. Malikopoulos, "An overview on optimal flocking," *Annual Reviews in Control*, vol. 51, pp. 88–99, 2021, doi: 10.1016/j.arcontrol.2021.03.004.
- [3] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH '87*, 1987, pp. 25–34, doi: 10.1145/37401.37406.
- [4] J. M. Lee, "An efficient algorithm to find k-nearest neighbors in flocking behavior," *Information Processing Letters*, vol. 110, no. 14–15, pp. 576–579, 2010, doi: 10.1016/j.ipl.2010.04.024.
- [5] M. Sajwan, D. Gosain, and S. Surani, "Flocking behaviour simulation: Explanation and enhancements in boid algorithm," *International Journal of Computer Science and Information Technologies*, 2014.
- [6] B. Wiandt, A. Kokuti, and V. Simon, "Application of collective movement in real life scenarios: Overview of current flocking solutions," *Scalable Computing: Practice and Experience*, vol. 16, no. 3, 2015, doi: 10.12694/scpe.v16i3.1099.
- [7] M. Joselli *et al.*, "A flocking boids simulation and optimization structure for mobile multicore architectures," pp. 83–92, 2012.
- [8] P. B. Byl, *Holistic game development with Unity: An all-in-one guide to implementing game mechanics, art, design, and programming*, AK Peters/. CRC Press, 2017.
- [9] G.-G. Wang, C.-L. Wei, Y. Wang, and W. Pedrycz, "Improving distributed anti-flocking algorithm for dynamic coverage of mobile wireless networks with obstacle avoidance," *Knowledge-Based Systems*, vol. 225, p. 107133, 2021, doi: 10.1016/j.knosys.2021.107133.
- [10] J. M. Lee and S. Kim, "A simulation of multiple grouping movements for pedestrians," *International Journal of Computational Vision and Robotics*, vol. 7, no. 3, pp. 276–284, 2017, doi: 10.1504/IJCVR.2017.083445.
- [11] B. Kawan and S. Alaliyat, "3D virtual fish population world for learning and training purposes," 2018, pp. 487–494, doi: 10.3384/ecp17142487.
- [12] N. T. H. Giang and L. H. Cuong, "Evaluating feasibility and effectiveness of digital game-based instructional technology," *International Journal of Emerging Technologies in Learning (IJET)*, vol. 16, no. 16, 2021, doi: 10.3991/ijet.v16i16.23829.
- [13] E. Adams, *Fundamentals of game design third edition*. Pearson Education, Limited, 2014.
- [14] W. Hou, D. Li, C. Xu, H. Zhang, and T. Li, "An advanced K Nearest Neighbor classification algorithm based on KD-tree," in *IEEE International Conference of Safety Produce Informatization (IICSPI)*, 2018, pp. 902–905, doi: 10.1109/IICSPI.2018.8690508.
- [15] M. Bardi and P. Cardaliaguet, "Convergence of some Mean field games systems to aggregation and flocking models," *Nonlinear Analysis*, vol. 204, p. 112199, 2021, doi: 10.1016/j.na.2020.112199.
- [16] D. S. Noonan, S. M. Breznitz, and S. Maqbool, "Flocking to the crowd: Cultural entrepreneur mobility guided by homophily, market size, or amenities?," *Arts, Entrepreneurship, and Innovation*. Springer Nature Switzerland, pp. 577–611, 2021, doi: 10.1007/978-3-031-18195-5\_4.
- [17] T. A. Assegie, T. Suresh, R. Subhashni, and D. M., "Mobility models for next generation wireless mesh network," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 22, no. 1, p. 379, 2021, doi: 10.11591/ijeecs.v22.i1.pp379-384.
- [18] G. Vásárhelyi, C. Virág, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek, "Optimized flocking of autonomous drones in confined environments," *Science Robotics*, vol. 3, no. 20, p. 3536, 2018.
- [19] M. M. R. Khan, R. B. Arif, M. A. B. Siddique, and M. R. Oishe, "Study and observation of the variation of accuracies of KNN, SVM, LMNN, ENN algorithms on eleven different datasets from UCI machine learning repository," in *International Conference on Electrical Engineering and Information & Communication Technology (iCEEICT)*, 2018, pp. 124–129, doi: 10.1109/CEEICT.2018.8628041.
- [20] B. Tang and H. He, "ENN: Extended nearest neighbor method for pattern recognition [research frontier]," *IEEE Computational Intelligence Magazine*, vol. 10, no. 3, pp. 52–60, 2015, doi: 10.1109/MCI.2015.2437512.
- [21] S.-H. Chen, "Fast algorithm for VQ codebook design," *IEE Proceedings I (Communications, Speech and Vision)*, vol. 138, no. 5, pp. 357–362, 1991, doi: 10.1049/ip-i-2.1991.0048.
- [22] M. B. A.- Zoubi, A. Hudaib, A. Huneiti, and B. Hammo, "New efficient strategy to accelerate k-means clustering algorithm," *American Journal of Applied Sciences*, vol. 5, no. 9, pp. 1247–1250, 2008, doi: 10.3844/ajassp.2008.1247.1250.
- [23] O. Adwan, "Efficient method to find nearest neighbours in flocking behaviours," *Signal & Image Processing: An International Journal*, vol. 10, no. 6, pp. 1–7, 2019, doi: 10.5121/sipij.2019.10601.
- [24] R. C., "Boids," p. 324, 2001.
- [25] F. Heppner and U. Grenander, "A stochastic nonlinear model for coordinated bird flocks," *The Ubiquity of Chaos*, 1990.
- [26] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967, doi: 10.1109/TIT.1967.1053964.
- [27] R. Parent, "Computer animation algorithms and techniques," 2012.
- [28] B. Chazelle, "The convergence of bird flocking," *Journal of the ACM*, vol. 61, no. 4, pp. 1–35, 2014, doi: 10.1145/2629613.
- [29] M. Moussaïd, D. Helbing, and G. Theraulaz, "How simple rules determine pedestrian behavior and crowd disasters," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 108, no. 17, pp. 6884–6888, 2011, doi: 10.1073/PNAS.1016507108.
- [30] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975, doi: 10.1145/361002.361007.
- [31] J. H. Friedman, J. L. Bentley, R. A. Finkel, and J. H. Frmdman, "An algorithm fQr finding best in logarithmic expected time," *ACYL Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209–226, 1977.
- [32] R. A. Brown, "Building a balanced k-d tree in o (kn log n) time," 2014.
- [33] C. Bartneck, M. Soucy, K. Fleuret, and E. B. Sandoval, "The robot engine-making the unity 3D game engine work for HRI," in *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2015, pp. 431–437, doi: 10.1109/ROMAN.2015.7333561.

## BIOGRAPHIES OF AUTHORS






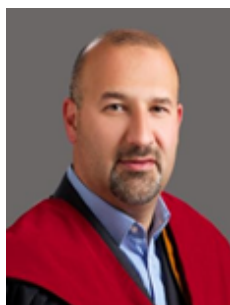
**Marwan Al-Tawil**    is currently an Assistant Professor with the University of Jordan, King Abdullah II School for Information Technology, Computer Information Systems Department. Dr. Marwan holds a B. Sc in Computer Information Systems (Al-Hussein Bin Talal University, 2006) and a M.Sc. in Information Systems (The University of Jordan, 2011), and a Ph. D in Computer Science (The University of Leeds, 2018). He is currently the Dean Assistant for Automated exams at the University of Jordan. His current areas of interest include Knowledge Graphs and Data Exploration and visualization. He can be contacted at e-mail: m.altawil@ju.edu.jo.






**Moh'd Belal Al-Zoubi**    is a Professor with the University of Jordan, King Abdullah II School for Information Technology, Computer Information Systems Department. Prof. Moh'd received a B.S. in Cybernetics from University of Belgrade in 1983, M.S. in Computer Information Systems from Detroit University and Ph.D. in Computer Science from the University of Leeds. Dr al-Zoubi research interests are in Image Processing, Machine Learning and GIS. He can be contacted at e-mail: mba@ju.edu.jo.






**Omar Y. Adwan**    is currently a Professor with the University of Jordan, King Abdullah II School for Information Technology, Computer Information Systems Department. Dr. Omar holds a B. Sc in Computer Science (Eastern Michigan University, 1987) and a M.Sc. in Computer Science (The George Washington University, 1998), and a Ph. D in Computer Science (The George Washington University, 2008). He served as a chairman to CIS Dept. of KASIT during 2012-2016. He is currently the Dean of the Faculty of Information Technology in Al-Ahliyya Amman University. His current areas of interest include Software Engineering, System Engineering Tools, and Databases. He can be contacted at e-mail: adwanoy@ju.edu.jo.



**Ammar M. Huneiti**    is currently a Professor with the University of Jordan, King Abdullah II School for Information Technology, Computer Information Systems Department. Prof. Ammar holds a B. Sc in Computer Science (University of Wales College of Cardiff, 1991) and a M.Sc. in Information Systems Technologies (The University of Wales College of Cardiff, 1992), and a Ph. D in Intelligent Information Systems (Cardiff University, 2004). He served as Vice Dean of KASIT during 2015-2016, and Dean of KASIT during 2020-2021. His current areas of interest include Intelligent Information Systems, Data Mining, Performance Support Systems, Multimedia, Geographic Information Systems, Spatial Databases, Adaptive Hypermedia. He can be contacted at e-mail: a.huneiti@ju.edu.jo.



**Reem Q. Al Fayez**    is currently an Assistant Professor with the University of Jordan, King Abdullah II School for Information Technology, Computer Information Systems Department. Dr. Reem holds a B. Sc in Computer Information Systems (The University of Jordan, 2009) and a M.Sc. in Information Systems (The University of Jordan, 2011) and a Ph. D in Computer Science (The University of Warwick, 2016). She served as the Dean Assistant for Student Affairs at the University of Jordan during 2020. Her current areas of interest include Linked Data, Graph Databases and its application in data management. She can be contacted at e-mail: r.alfayez@ju.edu.jo.