ISSN 1213-6972 (print)
ISSN 1213-6964 (online)
Journal of WSCG
http://www.wscg.eu
Vol.29, No-1-2, 2021

Drone interactions within the field of Augmented Reality

Damia Fuentes Escote
Department of Computer Sciene
University of Colorado Colorado Springs
Colorado Springs, CO, 80918, USA
dfunetes@uccs.edu

Sudhanshu Kumar Semwal
Department of Computer Sciene
University of Colorado Colorado Springs
Colorado Springs, CO, 80918, USA
ssemwal@uccs.edu

ABSTRACT

Using drones and augmented reality paradigm, new forms of interactive algorithms has been created and proposed. We start with a first person view interaction where the drone mimics the movement of one person's head wearing a HMD so that movements of the head can be mapped to actions by the drones. We then provide two novel AR/VR applications of drones to create something similar to third person view in 2D and 3D. To get started, our first idea is to control a drone using head movements. The second application which we implemented is to provide an implementation where tangible platforms are used by the drone to react to the movements of the character. Finally our third implementaton if to create and AR world using real outdoor scenery and asking a drone to mimic a third person view combining the real scenery with a synthetic actor so that based on the synthetic actor movement the drone changes its behavior correctly in the real-word trying to provide a synchronized view of the real and synthetic word. There are three novel ideas providing a new form of interactions which will improve with drones functionality in future. Our implementation shows the feasibility of our idea as discussed in the paper.

Keywords

Augmented and Virtual Realty, New Paradigm using Drones.

1 INTRODUCTION

While present focus for drones is mostly for disasters, product delivery and public safety [1-6,10-15], we think there are other major applications that can be useful too. For example, to provide new camera angles, 3D AR games, web-contents generation, individual recreational activities and computationally generated social interactions, instead of drones just taking photos/videos and racing.

A drone is a relatively new technology that could also be used for video games as well. This paper proposes a new paradigm between drones, AR and user interaction. The drone camera could be used as the camera view of an AR, 3D or 2D video game. This way, video games could be based in real life environments that are visible to the drone, not to the phone or the user. This newer paradigm provides augmentation of a different kind, bringing the immediately reality surrounding the drones to our AR worlds. This idea is novel and we provide a successful implementation of our idea with video-sequences of our work. We focused on feasibility of controlling drones, and drones being able to play the game and show the user how the game can be played, these are simple games, still the feasibility of our proposal is shown specially that using drones Augmented Reality applications provide new experiences. In future, we are planning IRB studies by extending our drone based ideas, including our other works [7-9], to include drone generated scenes from far away and project them for interaction on large screens [7-9].

The environment would be a real park field seen from the drone camera, as shown in Figure 1. The character, Figure 2, would be superimposed to the real environment and it will be controlled by the user. As an example, imagine being a video game of this type as shown in Figure 3. The user may be situated anywhere in the surrounding environment, at his home, or at any place of the world. Notice that in an interaction of this type the user moves the character, not the camera view which is drone's view. Our work is different that all the drone interactions and games released up until now where the user controls the drone. To the best of our knowledge, no implementation exist where a drone is controlled automatically while the user controls an AR character, and that is precisely our goal in this paper.

The main idea of our paper is new form of interactions emerge where the view point is the drone's camera and the user controls an AR character that is graphically imposed over the video feed. Then, the drone *moves* automatically depending on how the user moves the character.

With this idea in mind, we now could create games and interactions in the streets, in a park, in our garden or inside our house. We could create them in 2D or 3D, online or offline, multiplayer or single player. We could have games where the game's behavior depends on how the real world is behaving in that exact moment. For example, if there is a physical car, even though not implemented by us, we imagine that the game could simulate an enemy behind it. If there is a tree, the enemy could appear from behind a tree. In other words, we are providing a new way of designing AR 2D/3D games based on what the interests of the game creator. Only a proof of concept is provided in this paper. A feasibility study of this new way of engaging AR-Drone Games is presented.

2 FPV WITH HEAD TRACKING CONTROL INTERACTION

We started by controlling drone movements with the head mounted display (**?**). The user wears VR goggles and sees what the drone sees through its' camera. Also called First Person View, or FPV. This implementation included a smartphone inside the VR goggles which using an own Android app will send orders to the drone via WiFi. For this idea, we used the DJI Spark drone. As seen in Figure 7-11 any movement of the users head, is executed by the drone.

2.1 Mimicking Drone Algorithm Implementation

In order to have roll, pitch and yaw of the android phone mimicked, two sensors are needed: accelerometer and magnetic field. A rotation matrix is created from the readings of these sensors and is then expressed as three orientation angles in degrees: roll, pitch and yaw. In another thread, these variables are retrieved and are sent to the drone every short period of time together with the throttle value. We send the change in degrees of the phone to the drone. In order to do this we first get the orientation of the drone and after that we sum the change difference of the phone yaw. Roll and pitch are treated a little bit differently as they are received in degrees and then converted to be sent as velocities. We have them first normalized from -1 to 1, and then finally we multiply by a pre-set maximum speed, see Algorithm 1.

2.2 VR Goggles feedback

Previous section explained how to control the drone using head movements. But the user also needs to see the live video feed of the drone to actually move the drone to where he wants to go. And he needs to see it in the VR goggles. Since we are using one live stream from the drone, the same image is displayed in both eyes. Therefore, depth of field can not be perceived still that will be something which is feasible in future as that will require two coordinated drones to keep same distance from each other equal to the inter-eye distance. See Algorithm 2.

Algorithm 1 Pseudocode for Mimicking drone control

1: $roll$, $pitch$, yaw, $lastpyaw \leftarrow$ 0.0,0.0,0.0,0.0
2: **loop**
3: $pacce \leftarrow$ get phone accelerometer reading
4: $pmagn \leftarrow$ get phone magnetometer reading
5: $proll$, $ppitch$, $pyaw \leftarrow$ calculate rotation matrix using $pacce$ and $pmagn$ readings and express it as three orientation angles: roll, pitch and yaw
6: **if** yaw is 0.0 **then**
7: $yaw \leftarrow$ get drone compass orientation (between -180 Â° and 180)
8: **else**
9: $yaw \leftarrow yaw - (lastpyaw - pyaw)$
10: **end if**
11: $lastpyaw \leftarrow pyaw$
12: $yaw \leftarrow ((yaw + 180)\%360) - 180$
13: $nroll$, $npitch \leftarrow$ normalize $roll$ and $pitch$ to fall between -1 and 1
14: $sroll$, $spitch \leftarrow nroll * maximumspeed$, $npitch * maximumspeed$
15: move drone according to yaw, $sroll$ and $spitch$
16: **end loop**

Algorithm 2 Pseudocode for user feedback

1: $screenx \leftarrow$ get mobile phone screen size height
2: $screeny \leftarrow$ get mobile phone screen size width
3: **loop**
4: $frame \leftarrow$ get actual frame from drone
5: $frameeye \leftarrow$ center fill crop $frame$ with height $screenx$ and width $screeny/2$
6: Update the UI by placing $frameeye$ both in the right and the left of the screen
7: **end loop**

2.3 Results: Mimiking Drones

The experience was a high-fidelity implementation, and to us, it felt like that we are actually the drone. This interaction could be fully utilized as FPV (First-Person-View) games like shooters. We also have some drawbacks as follows:

- Noticeable delay. Around 1 second between user turn and user sees the turn.
- Camera field of view of the DJI Spark is too narrow.

There are following improvements which could provide better interaction in future:

- Have a wide camera field of view. The one used in the DJI Spark is too narrow and it was difficult

to see what the drone has around it. With a wider field of view the experience would be more real and attractive.

- Make the drone have two cameras separated 2.3 inches simulating two eyes. Then put the live video feed of each camera to the right and the left of the phone screen. This way, the user will be able to perceive in 3D.
- Having real time information of the drone like the IMU in order to be able to add AR to the real time video feed.

3 IMPLEMENTING PLATFORM INTERACTION USING DRONE

In a platform game, the controlled character must jump and climb between suspended platforms while avoiding obstacles. In order to make this type of game to adapt to new AR and drones paradigm we thought that the platforms could be drawn physically in the real world. The virtual character would jump between those real platforms based on the drone's camera, while drone will try to have the character always in the center of the screen. For implementation, we used the DJI Tello and DJI Tello EDU drones, while simulating the output and controls from the computer.

3.1 Drone detecting the platforms

Our algorithm detects the tangible (drawn) physical platforms using the live camera video sequence of a drone. Platforms are drawn in black in a big white paper as seen in Figure 13-18, we have used black tape to indicate platforms.

We implemented a simple algorithm to detect the platforms in real-time so that we can determine if the character is in a platform or not at every frame. For every frame, we check if the foot of the character is over black or white pixels. If the foot is falling on black pixels this means that the character is over a platform. Also, if it is falling over white pixels this means that it is not on a platform. Pixels in black and white have a value from 0 to 255. 0 is the complete black while 255 is the complete white. In order to differentiate if a pixel is over blacks or whites we used a $THRESHOLD$ of 50. Meaning that any pixel with a value lower than 50 will be considered black. And therefore that pixel will be understood as a platform. The pseudocode for this can be found in Algorithm 3.

Algorithm 3 Pseudocode for detecting if character is in a platform

```
x0 ← get horizontal center position of the character
x1 ← get horizontal left position of the character
x2 ← get horizontal right position of the character
y ← get vertical bottom position of the character
if (x0,y), (x1,y), (x2,y) fall inside the frame and the pixel color in black and white is lower than THRESHOLD then
    return True
else
    return False
end if
```

3.2 Moving the character on Platforms

Once we know if the character is on a platform, we want the character to start moving and jumping. For that, we use inputs from the user indicating move left or move right and jump key-strokes. Move left and right will be always directly applied to the character. The jump input will only by applied if the character is on top of a platform. Finally, a gravity effect will be applied when the character is not on top of a platform. Also, the character stops to fall once it hits a platform. At that moment, its vertical velocity will be set up to 0. The pseudocode for this can be found in Algorithm 4.

Algorithm 4 Pseudocode for moving the character

```
movey ← 0
loop
    frame ← get actual frame from drone
    if user is pressing right then
        movex ← STEPS
    else if user is pressing left then
        movex ← −STEPS
    else
        movex ← 0
    end if
    isinplatform ← check if character is in platform
    if user pressed jump and isinplatform then
        movey ← −STEPS × JUMPACCELERATION
    else if movey greater or equal than 0 and isinplatform then
        movey ← 0
    else
        movey ← movey + gravity
    end if
    move character movex vertically and movey horizontally
end loop
```

3.3 Applying AR to a simple character

Right now the character is just on the screen, and its movement is 100% relative to the computer screen. When the drone moves, the camera and the background will also move; the character will stay at the same place relatively to the screen. The expected behavior would be that the character should be pinned relatively to the

game map. And the game map is the live video feed of the drone's camera. We have simulated this behavior as follows:

1. Track key-points and descriptors of every frame using an ORB detector (explained below).
2. Find the same key-points from the past and actual frame using a BF matching algorithm (explained below).
3. For each match get their movement in the x and y directions from the past frame to the actual one.
4. Order the matches in order of relevance so that very match has a distance that tells objectively how accurate it is.
5. Find outlier matches and remove them (how to find outliers is explained below).
6. Calculate the weighted average x and y movement of the resulting matches. The weights give more importance to the first matches and less to the last ones.
7. Move the character the same amount in x and y as the average calculated in the step before.

With this strategy, the character stays in the same game map place no matter how the drone moves. The pseudocode for this can be found in "Algorithm 5".

3.3.1 ORB detector

We used the FAST (Features from Accelerated Segment Test) [1] detection test for a corner detection method, which is used to extract feature points and to track and map objects in many computer vision tasks. BRIEF (Binary Robust Independent Elementary Features) use binary strings as an efficient feature point descriptor. It is very fast both to build and to match. ORB (Oriented FAST and Rotated BRIEF) [1] is a fusion of FAST keypoint detector and BRIEF descriptor with many modifications which enhances the performance.

3.3.2 Brute Force (BF) matcher and Outliers

BF matcher finds the closest descriptor in the second set by trying all possible combinations. An outlier is a data point that significantly differs from the other data points.

3.4 Moving the drone

In our algorithm, as the character moves through all the platforms, the drone follows the character so that the character is always at the center of the screen. Naturally the camera of the drone will not include all the game map, instead, just a part of it. Results of the drone hovering over the map with platforms and keeping the character at the center is provided as successful implementation of our algorithm. In order to achieve this, we check where the character is at every frame and depending on that the drone is moved up, down, left or/and right. For instance, if the character is in the left side, the drone is moved to the left. If the character is in the right-up corner, the drone is moved up and right. The drone is moved until the character falls in the center of the screen again. The pseudocode for this can be found in Algorithm 5-6.

Algorithm 5 Pseudocode for applying AR to the character

```
Initialize newkeypoints as newdescriptors as None
loop
    frame ← get actual frame from drone
    previouskeypoints ← newkeypoints
    previousdescriptors ← newdescriptors
    newkeypoints, newdescriptors ← run ORB detector on frame
    if previousdescriptors is not None then
        matches ← run BF matcher with newdescriptors and previousdescriptors
        matches ← sort matches in the order of their distance
        Initialize movementsx and movementsy as emtpy lists
        for match in first 20 matches do
            listmovx.insert(using match, previouskeypoints and newkeypoints calculate the vertical movement of that match)
            listmovy.insert(using match, previouskeypoints and newkeypoints calculate the horizontal movement of that match)
        end for
        Remove outliers of listmovx and listmovy
        Move character the weighted average of movementsx vertically and movementsy horizontally.
    end if
end loop
```

3.5 Discussion

With this approach, we have proved that this new paradigm is possible and feasible. See the video-sequences submitted with this paper. The user is able to control an AR character through the real time video feed of the drone. And the drone is moving through the real world in order to maintain the AR character in the center of the video feed all the time. We can see this in 13-16. The accuracy of the method is high. When the AR character goes too much towards the borders, the drone automatically moves in order to maintain it

Algorithm 6 Pseudocode for moving the drone

```
1:  loop
2:      leftrightvelocity ← 0
3:      updownvelocity ← 0
4:      characterx ← get character horizontal center position
5:      charactery ← get character vertical center position
6:      hhss ← half of the horizontal screen size
7:      hvss ← half of the vertical screen size
8:      if characterx is less than hhss − CENTEROFFSET then
9:          leftrightvelocity ← −DRONEVELOCITY
10:     else if characterx is bigger than hhss + CENTEROFFSET then
11:         leftrightvelocity ← DRONEVELOCITY
12:     end if
13:     if charactery is less than hvss − CENTEROFFSET then
14:         updownvelocity ← DRONEVELOCITY
15:     else if charactery is bigger than hvss + CENTEROFFSET then
16:         updownvelocity ← −DRONEVELOCITY
17:     end if
18:     move drone according to leftrightvelocity and updownvelocity
19: end loop
```

inside the screen. We can see this in 17-18. Hardly ever the character goes off the screen, even intentionally. So, our algorithm has been successful in that sense. For this algorithm we set up the velocities so that the drone is faster than the character. Even so, if that is the case and the character moves really fast and gets out of the screen, the drone will keep trying to move till the character falls inside the center of the screen. But while the character is outside the screen, it will not be able to stop on a platform because the platform to stop will be inside the field of view of the camera.

The delay is pretty immediate although there is some, but it is almost exactly the same delay as the delay between sending commands to the drone and see the results in the real time video feed. Which is around 0.5s. Our is interactive as we can see the drone reacting to the scene and moving accordingly, thus the implementation shows feasibility of our proposed AR paradigm. In our opinion, set-up time for this novel interaction is time consuming. Set-up time include: having to charge the drone, set up the drone, connecting to it's wifi and taking off. There are other parts like creating the map with papers and black tape, and the actual playing that were comfortable and fun. Playing the game felt like something novel and creative. We draw the platforms. We knew that we could change them, we could add/remove/change platforms while we were playing which provides customizability to our taste as to where we will have platforms. In some sense, we could design a new level of the game at our pleasure by simply moving the platform-tapes and have slightly different interaction very quickly. This is a novel AR paradigm for Games in our opinion.

This implementation is just the beginning of a very broad topic in VR-Drones and Augmented Computer Games area. There are so many different directions for research to further develop. One drawback of our implementation was that when the character goes out of the screen and lands on a platform, it wont stop at it because the game would not know that there is a platform there. A more sophisticated platform recognition algorithm would be needed instead of the one we use to show the feasibility.

4 THIRD PERSON VIEW INTERACTION USING DRONES AND AR

The third novel idea which we implemented tries to simulate a 3D third person view interaction. Third-person is a perspective view where the player can see the body of the controlled character. In this novel interaction, the users sees a 3D character on top of the real ground and controls it. Then, the drone moves according to how the character is moved. For implementing this idea, we used the DJI Tello and DJI Tello EDU drones and the output and controls are from the computer.

The input for this type of interaction is how the user moves the character (forward, backward, right, left, turning right or left, or any combination) and the output is how the drone should move in terms of yaw, pitch and roll velocities, i.e. how the selection of moving buttons change the drone view which in turn will change the camera views which the user sees and interactively effect the choices by the user. This is shown in our third demo submitted with this paper for review.

As seen in Figure 19 if the user moves the character forward, the drone moves forward, and the same for moving backwards. As seen in Figure 20 if the user moves the character to the left, the drone moves to the left, and the same for moving to the right. But the things get a little bit more tricky when rotating the character, because the drone has to now rotate about that character correctly. In that case, as seen in Figure 21 if the user rotates the character to the left, the drone has to orbit around the character. The drone has to rotate slight to the left (negative yaw) and at the same time move fast to the right (positive roll). So, both yaw and roll are applied at the same time. The workflow of this algorithm can be seen in Algorithm 7.

The implementation is an open-loop which means that no feedback is created. The character is pinned to the screen. Key-points and descriptors are not used. The reason why this idea has been implemented this way is

Algorithm 7 Pseudocode for 3rd person view drone control

```
loop
    cLeftRight, cForwBack, cRotLeftRight ← get character movement from the user (1, -1 or 0, which express no movement)
    roll, pitch, yaw ← 0.0,0.0,0.0,0.0
    if cForwBack is not 0 then
        pitch ← cForwBack ∗ SPEED
    end if
    if cLeftRight is not 0 then
        roll ← cLeftRight ∗ SPEED
    end if
    if cRotLeftRight is not 0 then
        yaw ← −cRotLeftRight ∗ ROTATIONSPEED
        roll ← roll + cRotLeftRight ∗ SPEED ∗ 2
    end if
    move drone according to yaw, roll and pitch
end loop
```

because in order to give feedback to the algorithm we would need to detect the ground plane, and that can not be done at this time as we have limited access to sensors of the DJI Tello or DJI Tello EDU drones.

4.1 Discussion

We have submitted our results the third video-sequence. Results show that this idea could benefit from having some feedback loop. Nevertheless, we showed that an algorithm of this type can be implemented and showed the feasibility of this idea where the user controls the AR character through the real time video feed of the drone; and the drone moves through the real world in order to simulate the camera to provide 3rd person view.

Moving the character forward was straight forward. One of the things we tested was to set up a real person next to the AR character and moved the character at the same speed as the person. We can see this in 22-25. It was possible to simulate this effect as if it was an AR game. We also tested rotating the character, as seen in 26-30. While the character is rotating, it is moving a little bit from its own piece of ground like if it was moving with a little circumference. With perfect parameters of drone rotating velocity and left/right velocity we could have the AR character all the time in the same piece of ground. But then, this parameters would be different for a different drone height which makes it cumbersome/impractical to implement at this time. A lot of fine-tuning of parameters were required in these experiements. Finally, we also tested moving the AR character sideways. We can see this in Figures 31-35. The results for this movement were very similar as moving the character forward: it was possible to simulate this effect as if it was an AR game. Some time lapse exists when the user moved the character move and the drone catches up with that event. There is a noticeable 1s of lag-time in this case. Interactions in which the character needs to rotate quickly can be improved in future. The velocity of the drone is limited. As the drone orbits around the character, bigger is the orbit faster the drone needs to move. To orbit around the character, the drone uses two velocities: rotating and moving sideways. Drones provide limited functionality and may need to improve in future.

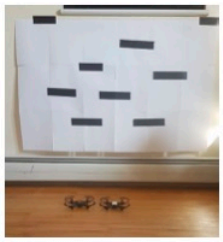

Figure 1: The game platforms/background

Figure 2: The environment

Figure 3: The character graphics

Figure 4: The AR game with the drone camera as the camera view

4.2 Future work

As we have seen, the main drawback of our open-loop implementation is when we want to rotate the character. One thing that we could do is to detect the key-points and descriptor of the piece of ground where the character is. Then, move the drone trying to maintain the character over those key-points and descriptors all the time. Also, detecting the ground-plane could be used in this type of interaction in the future. Future drones will need sensor data, together with the live video feed for that to be feasible.

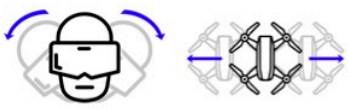

Figure 5: Tilting the head to the right, the drone should move to the right

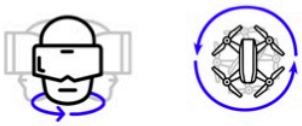

Figure 6: Rotating the head, the drone should rotate

Figure 7: Tilting the head to the front/back, the drone moves to the front/back

Figure 8: Mobile phone screen

Figure 9: Start of the program

Figure 10: Tilting the head to the right, the drone moves to the right

Figure 11: Rotating 90 degrees to the right, the drone rotates 90Â° to the right

Figure 12: Tilting the head to the front, the drone moves to the front

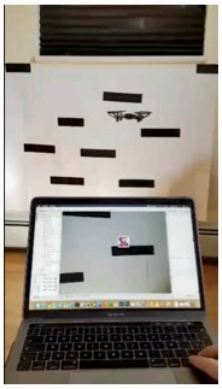

Figure 13: Platform results 1

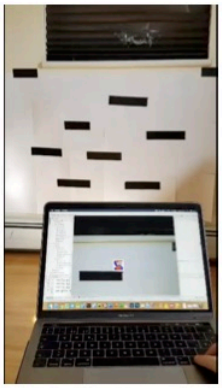

Figure 14: Platform results 2

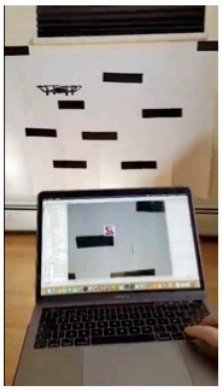

Figure 15: Platform results 3

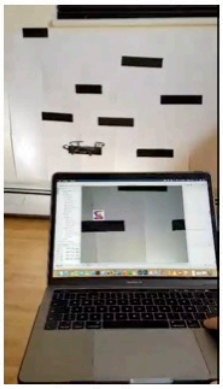

Figure 16: Platform results 4

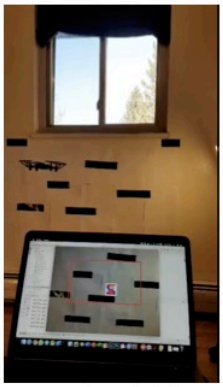

Figure 17: Platform results with drone moving bounds 1: Before jumping

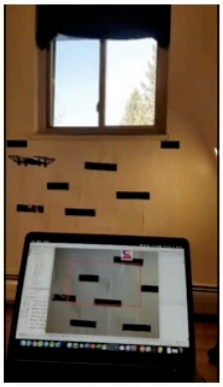

Figure 18: Platform results with drone moving bounds 2: While jumping going out of the rectangle

Figure 19: 3rd person view interaction: Moving character forward

Figure 20: 3rd person view interaction: Moving character right and left

Figure 21: 3rd person view interaction: Rotating the character to the left

Figure 22: 3rd person view interaction results forward 1

Figure 23: 3rd person view interaction results forward 2

Figure 24: 3rd person view interaction results forward 3

Figure 25: 3rd person view interaction results forward 4

Figure 26: 3rd person view interaction results rotating example 1

Figure 27: 3rd person view interaction results rotating example 2

Figure 28: 3rd person view interaction results rotating example 3

Figure 29: 3rd person view interaction results rotating example 4

Figure 30: 3rd person view interaction results rotating example 5

Figure 31: 3rd person view interaction results facing side 1

Figure 32: 3rd person view interaction results facing side 2

Figure 33: 3rd person view interaction results facing side 3

Figure 34: 3rd person view interaction results facing side 4

Figure 35: 3rd person view interaction results facing side 5

5 CONCLUSION

In this paper, we have explored the drone interactive applications within the field of Augmented Reality. Furthermore, new forms of interactive algorithms has been created and proposed, mainly focused in first and third view interactions. We have been successful in implementing three different basic introductory types of interactions. Our contributions is in the area of novel interactions paradigm for future drone applications in AR world/games. The first type of interaction was the drone VR FPV with head tracking control, which demonstrated how easy is to make the user feel as if they were the drone. The second one was the platforms interaction, which demonstrated that AR games where the user controls the AR character are possible, feasible, curious and fun. Finally, the third idea which we implemented was the third person view interaction, where we argue that this interaction can be used for many AR games in future. We are planning future IRB studies and detailed evaluation of our work in future research. Detailed evaluations strategies are being planned with several drones collaborating together to bring video-scenes to far-away audiences, and allowing interaction. Finally, we believe that much more would be done in this new and exciting novel area of AR with drones as the drones improve in future.

REFERENCES

1. E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," 2011 International Conference on Computer Vision, 2011, pp. 2564-2571.

2. Calonder M., Lepetit V., Strecha C., Fua P. (2010) BRIEF: Binary Robust Independent Elementary Features. In: Daniilidis K., Maragos P., Paragios N. (eds) Computer Vision ? ECCV 2010. ECCV 2010. Lecture Notes in Computer Science, vol 6314. Springer, Berlin, Heidelberg.

3. Viswanathan, N., Kelty-Stephen, D.G. Comparing speech and nonspeech context effects across timescales in coarticulatory contexts. Atten Percept Psychophys 80, 316-324 (2018).

4. Yunchao Wei, Wei Xia, Min Lin, Junshi Huang, Bingbing Ni, Jian Dong, Yao Zhao and Shuicheng Yan, IEEE Transactions on Pattern Analysis and Machine Intelligence, HCP: A Flexible CNN Framework for Multi-Label Image Classification, 2016

5. Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, Microsoft Research, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2016.

6. Bo Li, Junjie Yan, Wei Wu, Zheng Zhu, Xiaolin Hu, The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), High Performance Visual Tracking With Siamese Region Proposal Network, 2018.

7. Guilleum Budia Tirado, Sudhanshu Kumar Semwal, Automatic Parking Spot Detection System for Mobile Phnes using Drones and Deep Learning, Computer Science CSRN Notes, pp. 1-13, WSCG 2021 Proceedings, Plzen, CZ.

8. Guilleum Budia Tirado, Sudhanshu Kumar Semwal, Automatic Parking Spot Detection System with Image-Based machine learning, drones, and mobile platforms, MS thesis, UCCS, CO, USA. Advisor: SK Semwal, pp. 1-104, 2020.

9. Eric Marin Velazquez, Sudhanshu Kumar Semwal, Using Autonomous Drone Interactions towards Mobile Personal Spaced for Indoor Environments, Computer Science CSRN Notes, pp. 1-10, WSCG 2021 Proceedings, Plzen, CZ.

10. Punarjay Chakravarty, Klaas Kelchtermans, Tom Roussel, Stijn Wellens, Tinne Tuytelaars and Luc Van Eycken,IEEE International Conference on Robotics and Automation (ICRA), CNN-based Single Image Obstacle Avoidance on a Quadrotor, , pp.6369-6374, 2017.

11. Widodo Budiharto , Alexander A S Gunawan , Jarot S. Suroso , Andry Chowanda , Aurello Patrik and Gaudi Utama, International Conference on Computer and Communication Systems, Fast Object Detection for Quadcopter Drone using Deep Learning, pp. 192-195, 2018.

12. Z. Kaleem and M. H. Rehmani, "Amateur Drone Monitoring: State-of-the-Art Architectures, Key Enabling Technologies, and Future Research Directions," in IEEE Wireless Communications, vol. 25, no. 2, pp. 150-159, April 2018.

13. L. V. Santana, A. S. Brandão, M. Sarcinelli-Filho and R. Carelli, "A trajectory tracking and 3D positioning controller for the AR.Drone quadrotor," 2014 International Conference on Unmanned Aircraft Systems (ICUAS), 2014, pp. 756-767.

14. J. D. Renwick, L. J. Klein and H. F. Hamann, "Drone-based reconstruction for 3D geospatial data processing," 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), 2016, pp. 729-734.

15. Kang, H., Li, H., Zhang, J., Lu, X., Benes, B. (2018). FlyCam: Multitouch Gesture Controlled Drone Gimbal Photography. IEEE Robotics and Automation Letters, 3(4), 3717-3724.