Using Autonomous Drone Interactions towards Mobile Personal Spaces for Indoor Environments

Eric Marin Velazquez
Department of Computer Science
University of Colorado Colorado Springs
Colorado Springs, CO, 80918, USA
evelazqu@uccs.edu

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

ABSTRACT

We propose an extension of a recent work using convolutional neural networks and drones, such as Learning to fly by using DroNet [8] that can possibly safely drive a drone autonomously. The combination of (i) the DroNet architecture and weights to apply to CNNs to avoid the crashes; (ii) combining it with DLIB tracker, a correlation implemented tracker based on Danelljan et al.'s paper [3] work; (iii) the extraction of descriptors using Speeded Up Robust Features [1]; and (iv) Fast Library for Approximate Nearest Neighbors [10] for the feature matching – leads a drone to track any object and avoid crashes autonomously without any prior information about the object. The main goal is to create a partnership between the drone(s) and the participant as the drone follows the participant and avoids collisions. Our work extends existing methods to also included a way for a drone to follow a person even if the person is hidden for a few frames. Our algorithms also work in low/poor ambient light satisfactorily. In future, our technique can be used to provide novel indoor applications for drones.

Keywords

Drones, Navigation, Deep Learning, Tracking a person

1 INTRODUCTION

A drone, in a technological context, is an unmanned aircraft formally known as unmanned aerial vehicles (UAVs) or unmanned aircraft systems (UASs). Essentially, a drone is a flying robot. These flying robots can be controlled or can fly autonomously using neural networks, for example, working in conjunction with on board sensors, cameras or GPS [**?**]. Our focus is to use machine learning algorithm to control the functionality of drones and extend the personal spaces. State-of-the-art approaches on this topic have already provided models and algorithm to autonomously fly a drone indoor or outdoor avoiding crashes. While such approaches are able to safely fly the drones, they are not using it to provide any other application than just flying the drone. By combining flying with object tracking with drones' navigation we have implemented a new application for drones in our work, and extended drones usage to follow the participant in the indoor spaces. Our efforts can allow the participant to extend there environment and work with drones in a collaborative manner .

2 RELATED WORK

There have been several attempts to use drones effectively in both outdoor and indoor environments [8, 3, 4, 1, 10, 7, 6, 11, 5, 2, 9]. DroNet [8] is a convolutional neural network (CNN), whose purpose is to reliably drive an autonomous drone through the streets of a city.

2.1 Learning to fly by driving

DroNet [8] is trained with data collected by cars and bicycles, this system learns from the collected data to follow basic traffic rules, e.g, do not go off the road, and to safely avoid other pedestrians or obstacles. Surprisingly, the policy learned by DroNet can be generalized. For example, we were able to extend the training to fly a drone in indoor corridors and parking lots. Learning approaches predicts a steering angle and a probability of collision from the drone on-board forward-looking camera. These are later converted into control/flying commands which enable a UAV to safely navigate while avoiding obstacles. Since the goal is to reduce the image processing time, this paper advocate a single convolutional neural network (CNN) with a relatively small size. The architecture is partially shared by the two tasks to reduce the networks complexity and processing time, but is then separated into two branches at the very end: Steering prediction is a regression problem, which means that it requires the prediction of a quantity, while collision prediction is addressed as a binary classification problem. During the training procedure, only imagery recorded by manned vehicles is used. Steering angles are learned from images captured from a car,

while probability of collision is learned from images captured from a bicycle. DroNet system has been tested [8] for autonomously navigation on a number of different urban trails including straight paths and sharp curves as described in citedronet. Moreover, to test the generalization capabilities of the learned policy, they also performed experiments in indoor environments. They compare the approach against two baselines: Straight line policy and Minimize probability of collision policy. One of the metric is to use the average distance travelled before stopping or colliding. Results indicate that DroNet is able to drive a UAV for a long time on almost all the selected testing scenarios. The main strengths of the policy learned by DroNet are twofold:

- the platform smoothly follows the road lane while avoiding static obstacles.
- the drone is never driven into a collision, even in presence of dynamic obstacles, like pedestrians or bicycles, occasionally occluding its path.

2.2 Learning to fly by crashing

In this paper [4], a drone whose goal is to crash into objects is built: it samples random trajectories to crash into random objects. A larger number of crashes are recorded (11,500 times) to create one of the biggest UAV crash related data-set. This negative dataset provides scenarios of different ways that a UAV can crash. It also represents the policy of how UAV should NOT fly. They use all this negative data in conjunction with positive data samples from the same trajectories to learn a simple yet surprisingly powerful policy for UAV navigation. A simple self-supervised paradigm is quite effective in navigating the UAV even in extremely cluttered environments with dynamic obstacles such as humans. Parrot Ar-Drone 2.0 is used. The research is very interesting as no additional sensors/cameras are attached in the flying space during the data collection process. A two-step procedure for data collection is implemented. First, sample naive trajectories lead to collisions with different kind of objects. Then an annotation procedure for collected trajectories is described. The trajectories are first segmented automatically using the accelerometer data. This step restricts each trajectory up to the time of collision. Finally, trajectories are segmented into positive and negative data classification. For the Neural network portion, AlexNet architecture uses the ImageNet-pertained weights as initialization for the network. The network learns how to do a simple classification which uses an input image to predict if the drone should move forward in straight line or not. Based on the right cropped image, complete image and left cropped image, the network predicts the probability to move in right, straight and left direction respectively. If the straight prediction (P(S)) is greater than alpha, drone moves forward with the yaw proportional to the difference between the right prediction (P(R)) and left prediction (P(L)). Intuitively, based on the confidence predictions of left and right, robot is turned to left or right while moving forward. Model's performance is tested using (i) a Straight-line policy, (ii) a depth prediction based policy and (iii) a human controlled policy. The method is tested on 6 complex indoor environments doors, floors, entrance, etc. To evaluate the performance of different baselines, average distance and average time of flight without collisions is used as the metric of evaluation. This metric also terminates flight runs when drones take small loops (spinning on spot). On every environment/setting, this method performed better than the depth baseline. The best straight baseline provides an estimate of how difficult the environments are. The human controlled baselines have better results than their method for most environments. However, for some environments such as hallway and chairs, presence of cluttered objects makes it difficult for the participants to navigate through narrow spaces, and in this case drone method surpasses human level control in this environment.

2.3 DroNet Architecture

DroNet[8] is a convolutional neural network which is able to predict the probability of collision and the steering angle in real time having as the input, each frame of the drones camera. It was trained using data collected by cars and bicycles driving through different cities. It can guide a drone through a road following the traffic signs, avoiding obstacles and without going off the road. To reduce the image processing time, DroNet architecture is shared by the two tasks (probability of collision and steering angle) but then is separated into two branches at the end. Steering angle is a regression problem, which means that requires the prediction of a quantity, and probability of collision is a binary classification problem. DroNet uses the ResNet-8 architecture plus a dropout layer of 0.5 and a reLu non-linearity. The residual blocks of ResNet were proposed by He et al. [6]. After the ReLu layer, the two tasks share more parameters and the architecture splits into two different fully-connected layers. The first output is the steering angle and the second one is the probability of collision. The input of this convolutional neural network is a 200x200 frame in gray-scale.

2.4 Drone control

To control the drone, we use the outputs of DroNet. They used probability of collision to modulate the forward velocity and the predicted steering angle to calculate the yaw angle of the drone. For the forward velocity, the drone fly at a maximal speed when the probability of collision is zero and the drone stops when the probability of collision is close to one. A low-pass filtered version

of the modulated velocity to drive the drone smoothly is used.

$$v_k = (1-\alpha)V_{k-1} + \alpha(1-p_t)V_{max} \quad (1)$$

For the steering angle, a [-1, 1] range is mapped to a desired yaw angle in the range $[-\pi/2, \pi/2]$ and low-pass filtered using:

$$\omega_k = (1-\beta)\omega_{k-1} + (\beta\pi s_k/2) \quad (2)$$

Depending on the environment the value can be changed as well. One of the benefits of using DroNet is that it works perfectly in many indoor and outdoor scenes that contain line features but it fails in environments such as forests because of the lack of this features in the data set.

2.5 DLIB correlation tracker

DLIB tracker is a correlation implemented tracker based on [3]. Their work is based on MOSSE tracker [2]. The MOSSE tracker works well for objects that are translated but it does not work properly for objects that change in scale. DLIB tracker uses a scale pyramid to accurately estimate the scale of an object after the optimal translation is found. This helps us to track objects that change in translation but also in scaling throughout a video stream and furthermore in real-time. The approach works by learning discriminating correlation filters based on scale pyramid representation. They learn separate filters for translation and scale estimation. This scale estimation approach is generic and it can be incorporated into any tracking method with no inherent scale. Incorporating scale estimation makes the computational cost become much higher and their goal was to have an accurate and robust scale estimation approach and at the same time computationally efficient. Their method propose a fast scale estimation approach by learning separate filters for translation and scale. This restricts the search area to smaller parts of the scale space. In order to estimate the scale of the target in an image, [3] uses separate I-dimensional filter. and compares several trackers. The list includes ASLA tracker[7], SCM tracker [**?**], Struck[11], the LSHT tracker [5]. DLIB tracker works well for scaling and translation, and 2.5 times faster than Struck, 25 times faster than ASLA and 250 times faster than SCM in median FPS [**?**].

Obviously, the main advantage of using this correlation tracker in our research is that it will track our selected target and it will compute the translation and the scale of it. This lets our algorithm know either the object is moving left or right and furthermore we will be able to know if the tracking object is moving closer or further from us by just computing the area of the tracking object. While this algorithm helps us in a very prominent and efficient way it has a very big disadvantage which could make our algorithm fail. This disadvantage occurs when the object we are following, completely disappears from the camera field and then it appears again. As this tracker is based in the correlation between frames, it will not be able to detect and track the same object again. To address this problem, we implemented a novel algorithm using descriptors and key points of the tracked object so that when the object we are tracking disappears for a few frames, the algorithm will be able to find it again. This feature will be explained and discussed later.

2.6 Feature extraction: SURF

Speeded Up Robust Features (SURF) is a feature extraction algorithm presented in [1]. It is a faster version of SIFT algorithm [9]. While Scale Invariant Feature Transform (SIFT) uses Lowe approximated Laplacian of Gaussian with Difference of Gaussian for finding scale-space, SURF approximates LoG with Box Filter. The main advantage is that convolution with box filter can be easily calculated with help of integral images and it can be done in parallel for different scales. SURF rely on determinant of Hessian matrix for both scale and location. For orientation, SURF uses wavelet responses in horizontal and vertical direction. Gaussian weights are also applied to it. The dominant orientation is estimated by calculating the sum of all responses within a sliding orientation window of angle 60 degrees. For feature description, this algorithm uses Wavelet responses in horizontal and vertical direction. A neighborhood of size 20sX20s is taken around the key-point where s is the size. It is divided into 4x4 sub-regions. For each sub-region, horizontal and vertical wavelet responses are taken and a vector is formed. This when represented as a vector gives SURF feature descriptor with total 64 dimensions. Lower the dimension, higher is the speed of computation and matching, which provide better distinctiveness of features. Another important improvement is the use of sign of Laplacian (trace of Hessian Matrix) for underlying interest point. It adds no computation cost since it is already computed during detection. The sign of the Laplacian distinguishes bright blobs on dark backgrounds from the reverse situation. In the matching stage, we only compare features if they have the same type of contrast. This minimal information allows for faster matching, without reducing the descriptor's performance. SURF adds a lot of features to improve the speed in every step. Analysis shows it is 3 times faster than SIFT while performance is comparable to SIFT. SURF is good at handling images with blurring and rotation, but not good at handling viewpoint change and illumination change.

3 OUR ALGORITHM FOR A DRONE TO FOLLOW A PERSON

The most useful feature that we have taken profit from DroNet is the probability of collision because the steering angle must cause some bad prediction. This is caused

Algorithm 1 DroNet usage pseudo code

```
WHILE TRUE{
    frame = drone_camera()
    probability of collision = DroNet(frame)
    WHILE probability of collision > THRESHOLD_PROB {
        STOP Drone movement
        frame = drone_camera()
        probability of collision = DroNet(frame)
    }
    DRIVE Drone
}
```

Figure 1: DroNet Usage pseudo code

Algorithm 2 DLIB tracker usage pseudo code

```
WHILE TRUE {
    frame = drone_camera()
    confidence = DLIB(frame)
    IF confidence < THRESHOLD_TRACK {
        Tracking_object = Lost
        DRONE try to RE-FIND object
    }
    ELSE{
        Tracking_object = Found
        DRONE MOVES FOLLOWING Tracking_object
    }
}
```

Figure 2: DLIB tracker usage pseudo code

because DroNet is trained to follow roads and not objects, by that we mean that the steering angle will follow the road direction and will not follow the object if the object tries to get off-road. Knowing that we have used the probability of collision as basis of our implementation, we are proposing that if the probability of collision is higher than a threshold, the drone should stop immediately. If this probability of collision is lower, then the velocity of the drone will not be calculated based on that probability but based on the velocity of the object we are following. This is a very different feature than the one used in DroNet but since our objectives are very different, our proposal is the best way to add their work into our project.

While the probability of collision is higher than the $THRESHOLD_PROB$, the drone will be stopped at the same position as shown in Figure 1. This is because our goal is to follow anything, for in our example a person. If we are behind this person, following him, we will not have any collision if he has no collision either but we could have a collision if something crosses between the drone and the person. In this case the drone will stop and once this object has crossed, the drone will restart following the person again.

3.1 Using DLIB tracker in our research

The DLIB tracker is a correlation algorithm. So this algorithm does not know what is exactly following does not have memory. This means that if for example, the light goes off and on, the tracker will not know what to follow and it will get lost. This tracker returns a number which tells the confidence the tracker has for following the object correctly. If the object is partially occluded then this probability will decrease. Knowing this number, we can tell when we are tracking our object or when it is lost. Therefore, our algorithm for the tracker is very similar to the Algorithm in Figure 1. If the confidence of the tracker is lower than a threshold, then we will mark our tracking object as lost and we will start trying to find it again. If the confidence of the tracker is high enough, then we will move the drone as the object is moving as shown in Figure 2.

As we are going to follow every object from the same approximate distance, we know exactly which is the distance that separates the drone and the tracking object. We can also, easily know, before losing the object what was the direction it was moving (right or left). By knowing that, we could re-find our object again if our tracker would know what was the object we were following. As our tracker does not know it, we use descriptors and key points to be able to re find the lost object.

3.2 Novel Algorithm: use of descriptors

We extract features, called descriptors, of the object we are tracking. We use this descriptor to re-find the object we were tracking. The basic idea is that we create a buffer of size N where we store these descriptors while we are tracking the object and the confidence of the tracker is high. Once the object has been lost, we use this buffer to check descriptor by descriptor if someone matches with what the drone is capturing. The main point of using a buffer, instead of a single variable is that we want to store several descriptors of the object we are following in case this object changes its form during the following process (due to moving faster or for example sitting down etc). Imagine we are following a person from the back, but then this person turns 180 degrees and he is facing the drone, then we want the descriptors of the person facing the drone and not his back. To be able to do that, our buffer has a fixed size N and every position has a different descriptor value. This value is overwritten once the buffer is full. The only position we do not overwrite is the first position of the buffer where we store the descriptors of the Region of Interest (ROI) of the first frame we used to track because the first ROI will have for sure the descriptors of the object we want to follow without any kind of occlusion. As the drone is sending up to 30 frames per second (fps) and the object we are pursuing will rarely move or change its form that fast we just store 2 descriptors per second. Using this technique, our algorithm becomes more efficient and we

Algorithm 3 Saving descriptors usage pseudo code

```
Buffer size = N
frame = drone_camera()
Once ROI selected from Frame -> Save descriptors at position
Start tracking {
    Frame = Drone_camera
    confidence = DLIB(Frame)
    POS = 1
    IF confidence > THRESHOLD_TRACK {
        Save descriptors of ROI at position POS of the buffer
        IF POS == N { POS = 1}
        ELSE { POS = POS + 1}
    }
}
```

Figure 3: Saving Descriptors Pseudo Code

Algorithm 4 Using descriptors to re-find objects usage pseudo code

```
frame = drone_camera()
Actual_descriptor = descriptor(Frame)
Found = False
FOR descriptor in BUFFER {
    If actual_descriptor matches descriptor {
        Found = True
        BREAK
    }
}
IF Found == True {
    Homography = H(actual_descriptor, descriptor)
    Object_in_frame = Homography(frame)
    Start tracking DLIB(Object_in_frame)
}
```

Figure 4: Using Descriptors to re-find the object Pseudo-code

need less space to store the buffer. The algorithm of storing descriptors is shown in Figure 3.

Saving these descriptors in a good and efficient way is really important to be able to re-find our object once it has been lost. Once the object is lost, our algorithm tries to re-find the object by using the descriptors stored in the buffer position by position and starting at position 0 which is where the initial descriptors were stored. If the object is found, then we start tracking again the object using the DLIB tracker. This is a major improvement proposed towards resiliency of our work, we added descriptors to to re-find the object as shown in Figure 4

4 HANDLING TOUGH SCNEARIOS

The most common tough cases our system could face are:

- No light
- Occlusions
- Very sharp turns

Algorithm 5 General edge cases usage pseudo code

```
Distance_from_object = D (meters)
Last_direction = Right/Left
WHILE DRONE MOVES D (meters) straight {
    IF object found {
        Start tracking again
        BREAK
    }
}
DRONE TURNS Last_direction
IF object found {
    Start tracking again
    BREAK
}
ELSE {
    Keep rotating until object found or 360 degrees
    IF object found {
        Start tracking again
        BREAK
    }
    ELSE {
        LAND Drone
        STOP
    }
}
```

Figure 5: General Edge case usage Pseudo-code

These are handled by implementing (a) Re-finding algorithm and (b) and an algorithm to handle poor lighting conditions.

4.1 Re-finding the object of interest

As we cannot distinguish each of these cases while we are flying, we have implemented a general algorithm to be able to re-find the object. As we previously know the distance between the object and the drone and also the last direction that our object moved, the RE-FIND object algorithm is shown in Figure 5:

Using this algorithm, the drone will be expected to first move straight D meters and then turn either left or right based on that last direction the object moved to find it again. If the object has not been found, then the drone will make a 360 degrees rotation movement to try to find the object again, see Figure 5 for pseudo code. If the object has not been found, then the drone will land and the program will be stopped.

4.2 No light: Novel Algorithm

When there is no light at all, our tracking algorithm confidence value is NaN (Not a Number). Therefore, our implementation is to set the probability of collision to 1 when the tracking algorithm value is NaN. Using this approach, our drone will stop when the light is off and then re-start the tracking when the light is on as shown in Figure 6.

Algorithm 6 No light pseudo code

```
WHILE TRUE{
    Frame = Drone_camera
    confidence = DLIB(Frame)
    IF confidence == NaN {
        STOP drone
        SET probability of collision = 1
    }

    ELSE {
       Start tracking again
    }
}
```

Figure 6: No light Pseudo code

5 OVERALL WORKING OF OUR ALGORITHM

To control the drone, we have to combine all the algorithms we have explained before into one single algorithm. The most relevant parameter we have to check is the probability of collision. If it is higher than the threshold, then we will stop the drone movement until this probability has decreased. No matter what is happening, we do not want our drone to crash. The second parameter we have to take into consideration is the DLIB tracker confidence. If this confidence is higher than the threshold, then we will follow the object while we save the correspondent descriptors. If it is lower than the threshold, then we will stop saving descriptors and we will start to find the object again. If it is found, then we will start the tracking movements again, if it is not after completing the edge-cases movements, we will land the drone and stop the program.

Some parameters can change when we follow different objects. It is not the same to follow a human than a bike or even a car. These parameters are initialized by the user. These parameters are the following: Distance between the drone and the object; Height of the drone; Velocity of the drone. Complete drone algorithm implemented by us is shown in Figure 7 as a Pseudo code.

6 IMPLEMENTATION AND RESULTS

We have programmed everything using Python and PyCharm as the IDE. The libraries we have used for this project are the following: AR Parrot 2.0, DJI Tello python API, OpenCV, DLIB: main tracker API, and Tensorflow.The most useful feature that we have taken profit from DroNet is the probability of collision because the steering angle must cause some bad prediction. This is caused because DroNet is trained to follow roads and not objects, by that we mean that the steering angle will follow the road direction and will not follow the object if the object tries to get off-road. Knowing that we have used the probability of collision as main feature of our project. we are proposing that if the probability of collision is higher than a threshold, the drone should stop immediately. If this probability of collision is lower, then the velocity of the drone will not be calculated based on that probability but based on the velocity of the object we are following. This is a very different feature than the one used in DroNet but since our objectives are very different, our proposal is the best way to add their work into our project. While the probability of collision is higher than the $THRESHOLD_PROB$, the drone will be stopped at the same position. This is because our goal is to follow anything, for example a person. If we are behind this person, following him, we will not have any collision if he has no collision either but we could have a collision if something crosses between the drone and the person. In this case the drone will stop and once this object has crossed, the drone will restart following the person again. This DLIB tracker returns a number which tells the confidence the tracker has for following the object correctly. If the object is partially occluded then this probability will decrease. Knowing this number, we can tell when we are tracking our object or when it is lost. Therefore, our algorithm for the tracker is very similar to the Algorithm 1. If the confidence of the tracker is lower than a threshold, then we will mark our tracking object as lost and we will start trying to find it again. If the confidence of the tracker is high enough, then we will move the drone as the object is moving. As we are going to follow every object from the same approximate distance, we know exactly which is the distance that separates the drone and the tracking object. We can also, easily know, before losing the object what was the direction it was moving (right or left). By knowing that, we could re-find our object again if our tracker would know what was the object we were following. As our tracker does not know it, we use descriptors and key points to be able to re find the lost object. We extract features, called descriptors, of the object we are tracking using the algorithms we have developed. We use this descriptor to re-find the object we were tracking. The basic idea is that we create a buffer of size N where we store these descriptors while we are tracking the object and the confidence of the tracker is high. Once the object has been lost, we use this buffer to check descriptor by descriptor if someone matches with what the drone is capturing.

The main point of using a buffer, instead of a single variable is that we want to store several descriptors of the object we are following in case this object changes its form during the following process (due to moving faster or for example sitting down etc). Imagine we are following a person from the back, but then this person turns 180 degrees and he is facing the drone, then we want the descriptors of the person facing the drone and not his back. To be able to do that, our buffer has a fixed size N and every position has a different descriptor value.

Algorithm 7 Drone control pseudo code

```
Distance_from_object = D (meters)
Last_direction = Right/Left

WHILE True{
    Frame = Drone_camera
    Probability of collision = DroNet(Frame)
    IF Probability of collision > THRESHOLD_PROB{
        STOP DRONE MOVEMENT
        }
        ELSE{
                Confidence = DLIB(Frame)
                IF confidence == NaN {
            STOP drone
            SET probability of collision = 1
        }
                ELIF Confidence < THRESHOLD_TRACK {
                        Tracking_object = Lost
                        WHILE Tracking_object == Lost {

                                WHILE DRONE MOVES D (meters) straight {
                                        find_descriptors()
                                        IF object found {
                                                Tracking_object = Found
                                                BREAK
                                        }
                                }
                                DRONE TURNS Last_direction
                                find_descriptors()
                                IF object found {
                                        Tracking_object = Found
                                        BREAK
                                }

                                ELSE {
                                        Keep rotating until object found or 360 degrees
                                        find_descriptors()
                                        IF object found {
                                                Tracking_object = Found
                                                BREAK
                                        }
                                        ELSE {
                                                LAND DRONE
                                                STOP
                                        }
                                }

                        }

                }
                ELSE{
                        Tracking_object = Found
                        DRONE MOVES FOLLOWING Tracking_object
                        Last_direction = Right/Left
                        save_descriptors()

                }

        }
}
```

Figure 7: Drone Control Pseudo-code

This value is overwritten once the buffer is full. The only position we do not overwrite is the first position of the buffer where we store the descriptors of the Region of Interest (ROI) of the first frame we used to track because the first ROI will have for sure the descriptors of the object we want to follow without any kind of occlusion. As the drone is sending up to 30 frames per second (fps) and the object we are pursuing will rarely move or change its form that fast we just store 2 descriptors per second. Using this technique, our algorithm becomes more efficient and we need less space to store the buffer. Saving these descriptors in a good and efficient way is really important to be able to re-find our object once it has been lost. Once the object is lost, our algorithm tries to re-find the object by using the descriptors stored in the buffer position by position and starting at position 0 which is where the initial descriptors were stored. If the object is found, then we start tracking again the object using the DLIB tracker.

7 DRONE CONTROL

The DJI Tello is a newer and smaller drone but it is also cheap and robust. As it is newer, the implementation of the python API is more efficient and more reliable. The most important specifications for the DJI Tello drones are: Weight: 0.2 lbs; Dimensions: 3.86 x 1.61 x 3.64 inches; Max flight time: 13 minutes; Battery: 1100 mAh LiPo; Camera: 5MP camera 720p, 30 fps video output. The drone has three axes to control: Pitch (Y axis), Yaw (Z axis) and Roll (X axis). If the aircraft rotates around the Pitch axis it will move in the X axis direction. If the Pitch angle is positive, the direction will be backwards, or in the negative X axis. The same will be applied when rotating the Roll axis, which will move the aircraft in the Y axis direction. If the pitch value is positive, the drone will move forward and if it is negative the drone will move backwards as we can see in the following image. If the yaw value is positive, the drone will rotate in clockwise direction on itself and if the yaw is negative, the drone will rotate counter clockwise on itself. If the roll value is positive, the drone will move right and if it is negative the drone will move left as we can see in the following image. There is another drone variable which has to be controlled, the throttle. This controls the aircraft's average thrust from its propulsion system. When the aircraft is level, adjusting the throttle will move the aircraft up or down as all the thrust is in the vertical direction. However, when the aircraft is not level (has non-zero pitch or roll), the thrust will have a horizontal component, and therefore the aircraft will move some horizontally. A larger pitch or roll angle will result in more horizontal thrust and therefore faster horizontal movement. To control this axes of the drone autonomously, what we want is to have always our tracking object in the middle of the frame. If the object we are tracking is in the green zone of the

Figure 8: Simple walk result 3 (left) and 4 (right)

frame, we will not move the drone. If the object is in any blue zone, we will set the throttle of the drone to either go up or down. We always set the roll angle to 0 as we do not want the drone to move right or left but in case the object is in any white space, we will set the yaw in order to rotate the drone to the correct direction. To be able to set the drone's pitch to move the drone forward or backward, what we do is to calculate the area of the tracking object at the beginning of the tracking and if this area increases, we move the drone backwards because it means that the object is approaching the drone, if this area decreases, we move the drone forward because the tracking object is moving further. The yaw, pitch and throttle velocities are changed depending on the environment and the distance between the drone and the object has to be defined by the user in order to be able to run the re-finding algorithm when the object is lost.

8 RESULTS: TEST CASES FOR ALGORITHMS DEVELOPED

The test cases we are going to test are going to follow three different things/objects: a human, a bike and a car. Each of the test cases are going to have different situations which its difficulty will have a range between easy, medium, hard and extreme as explained in the following table. Also, we show several frames which were recorded by the drones. Test cases can be divided based on situation and difficulty: (i) Human walking with no interference is easy; (ii) Human walking with a probability of collision is harder; (iii) Human walking with interference is of medium complexity; (iv) Human walking when person disappears and then appears is of extreme difficulty; (v) Human walking with sharp turns is harder; (vi) Low light is of medium complexity, and (vii) Light on and off is harder. We have implemented all these cases with success as the following images and results show below. Two video sequences submitted with this paper also show that these algorithms have been successfully implemented.

8.1 Human walking: no interference

This is the most basic test case. We want our drone to follow a human while he is walking and there are no interference in the tracking. By that, we mean that there are no objects crossing between the drone and the human.

8.2 Human walking: with interference

This test case is the same as the last test case but with objects/persons crossing between the drone and the hu-

Figure 9: Walking with interference 5 (left) and 6 (right)

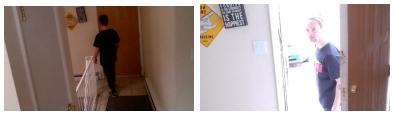

Figure 10: Probability of collision 3 (left) and right (7).

Figure 11: Disappear and appear 4 (left) and right (7)

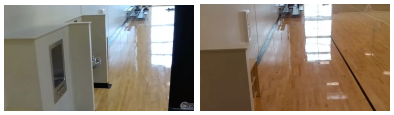

Figure 12: Sharp turns 5 (left) and 6 (right)

man. Having this interference will force our algorithm to re find the human if the obstacles occlude him.

8.3 Human walking: probability of collision

We want to test if our drone stops moving when the probability of collision is really high and if it starts tracking again when this probability of collision becomes lower.

8.4 Human walking: disappear/appear

When the tracking object disappear and appears again we ant to make sure we recognize it as the object we were following before. We wan to test how we apply the descriptors algorithm against this situation.

8.5 Human walking: sharp turns

Sharp turns are really difficult to overcome. The sharper the turn is the more challenging is to follow the tracking object.

8.6 Low light

We also want to test our algorithm again adverse conditions and one of them is the low ambient light. The tracker will need to track the human with low light and this could cause some problems.

Turning the light off could cause a lot of problems calculating the probability of collision, tracking the object and saving descriptors. We want to test our system against these difficulties.

9 CONCLUSION

In this paper we implemented a new way to track and follow objects in real time with no previous information

Figure 13: Low light recognized the edges of a mobile phone.

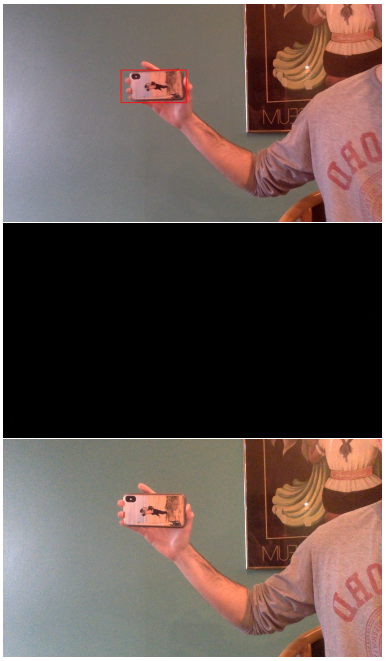

Figure 14: Light on/off 3

about that object before. This mean that the user selects the object to track in real time. As the real time feature and the lack of delay between what the user is seeing and what the drone is doing were some of our priority goals, we used a multi-threading approach in order to implement our algorithm. We found a new application to apply DroNet and the probability of collision. We have also introduced a new technique on how to use features descriptors. Storing descriptors in a buffer and then using this information in case that the object is lost is a new and efficient way to re-find lost objects which worked for us successfully as can be seen by two video sequences we have submitted with this paper. There has been a huge trade off between the tacker and the probability of collision. As our primary requirement was not to crash the drone, no matter what, the probability of collision predominates the control of the drone and this decision could lead to sometimes loosing the tracking object in some cases. In future, we would like to extend our ideas to providing interactive spaces where drone-interactions can lead to novel aapplications.

10 REFERENCES

[1] H Bay, T Tuytelaars, and L Van Gool. 2006. SURF:

Speeded Up Robust Features. In *In: Leonardis A., Bischof H., Pinz A. (eds) Computer Vision ? ECCV 2006. ECCV 2006. Lecture Notes in Computer Science, vol 3951. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11744023*$_3$*2.ECCV*.

[2] DS Bolme, JR Beveridge, BA Draper, and YM Lui. 2010. Visual Object Tracking using Adaptive Correlation Filters. In *CVPR*. ACM, 2544–2550.

[3] Martin Danelljan, H Gustav, S. Khan, and Michael Felsberg. 2014. Accurate Scale Estimation for Robust Visual Tracking. In *Proceedings British Machine Visual Conference*. BMVA Press, 1–11.

[4] Dhiraj Gandhi, L Pinto, and A Gupta. 2017. Learning to Fly by Crashing. In *IEEE RSJ International Conference on Intelligent Robots and Systems (IROS)*. IROS, 3948–3955.

[5] Sam Hare, Amir Saffari, and Phillipe Torr. 2013. Visual tracking via locality sensitive histograms. In *CVPR*. IEEE, 1–8.

[6] H He, Q Yang, R Lau, J Wang, and M Yang. 2012. Deep Residual Learning for Image Recognition. In *CVPR*. ACM, 1–12.

[7] Xu Jia, H Lu, and Yang Ming-Hsuan. 2012. Visual Tracking via Adaptive Structural Local Sparse Appearance Model. In *IEEE Conference on Computer Vision and Pattern Recognition*. ACM, 1–2.

[8] Antonio Loquercio, Ana Maqueda, Carlos del Blanco, and David Scaramuzza. 2018. DroNet: Learning to Fly by Driving. In *IEEE ROBOTICS AND AUTOMATION LETTERS, vol. 3, no 2 April*. IEEE, 1088–1095.

[9] DG Lowe. 2004. Distinctive Image Features from Scale-Invariant Keypoints. In *International Journal of Computer Vision vol 60, issue 2*. Springer, 91–110.

[10] M Muja and David Lowe. 2014. FLANN - Fast Library for Approximate Nearest Neighbors. In *Pattern Analysis and Machine Intelligence (PAMI) vol. 36,*. IEEE, 1–14.

[11] W Zhong, H Lu, and Ming-Hsuan Yang. 2011. Struck Structured output tracking with kernels. In *ICCV*. ACM, 263–270.