Robotics Project

Object tracking with a Drone

Anton Vo - S300938

Einar Tomter - S331456

Saodat Mansurova - S331447

Contents

1 Introduction

This is a project for the Robotics course ELVE3610 for the fall semester 2020 in OsloMet. The choice for the type of project is open, hence we chose to work with drones, more specifically a quad-copter.

Drones have surged in popularity lately, especially with the rise of more affordable and accessible models. The Tello drone are of particular interest, since you are able to send and receive data with the drone and program it remotely. This opens up the possibilities for controlling the drone in different ways, even using AI to aid it. The AI algorithm by itself can process the data sent by the drone, and allows for object detection. You can pass this data through an algorithm and send the movement directions back to the drone. Hence, the main objective of this project is to be able to provide autonomous control to the drone based on the data it can send, and track a desired object.

There are many tools available to help with this task. A lot of work needs to be done to process the data correctly. For instance there may be some noise in the data that needs to be filtered out. In addition, the instructions to be sent to the drone need to be fine tuned to ensure that it does not become unstable and moves smoothly. In this project we have tested a few methods to try maximizing these results.

2 Related Work/Background

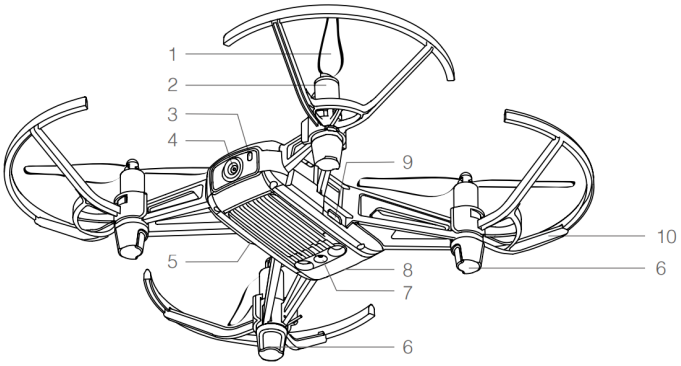
Since 2016, the number of people who have used drones has increased substantially [12]. The usage still follows a positive trend even today. They are used in a wide variety of fields, including but not limited to military use, package delivery services, competitive drone racing, and by hobbyists and enthusiasts. According to a statistic by the Federal Aviation Administration (FAA), the total number of registered drones just from hobbyists totaled 1.2 million units [1]. Looking at the commercial sector, the number of drones registered totaled roughly 510 700 units in the US in 2019. This is not surprising considering the numerous applications that they have. For instance, Amazon has slowly expanded their delivery services by using drones [2].

Drones are usually controlled by an operator and can often require a great deal of expertise, especially with drones that are used in more professional settings [30]. A lot of research is also dedicated towards creating better control systems for the operators [17]. However, on the other end, we have people who strive to automate tasks, including drone control. Lately, this trend has also been growing, with several studies being conducted on autonomous drone control [18]. Many studies have also applied AI in some form or another within drone projects [9, 31].

Among the applications of AI within drones, object detection and recognition is often used. One issue that needs to be considered, is that object detection usually takes time and considerable computing power: Object recognition algorithms such as R-CNN are often used because of their high accuracy and there are studies conducted to improve the speed at which they can recognize objects [25]. But even these algorithms may function too slowly when working with real-time detection, but several studies have attempted to improve the application of object detection within drones [14, 24].

3 Theory

3.1 DJI Tello



1. Propellers
2. Motors
3. Aircraft Status Indicator
4. Camera
5. Power Button
6. Antennas
7. Vision Positioning System
8. Flight Battery
9. Micro USB Port
10. Propeller Guards

Figure 1: Tello drone (source)

The DJI Tello is a quadcopter manufactured by RYZE and features a Vision Positioning System and an onboard camera [26, 28]. Some of the specs include:

- 13 min flight time
- 100 m flight distance
- 720p 30fps video transmission
- SDK for software development
- 13 min flight time

It also has an app that allows for manual control and built in maneuvers such as bounce, throw and go, and 8D flips.The most important point for this project is that it can easily be programmed through their software development kit (SDK) [27]. It allows for easy connection and communication between the drone and a control station such as a computer.

3.2 Object Recognition

The drone is able to send various types of data. The most important data that we have to collect in order to track an object is the frames received by the video transmitter. The video frames by itself does not tell the algorithm where the object is and has to be processed to locate the coordinates of the object in question. There are several ways to do this and usually requires the use of AI specifically within image processing.

Object detection is a general term relating to the identification of objects in digital photographs [6]. This goes hand in hand with image classification, which is the process of labelling an image based on the patterns that the algorithm can identify [5]. The labels depend on the data that the algorithm has been given. An example is that it will only identify a horse if it has been taught to find horses within an image.

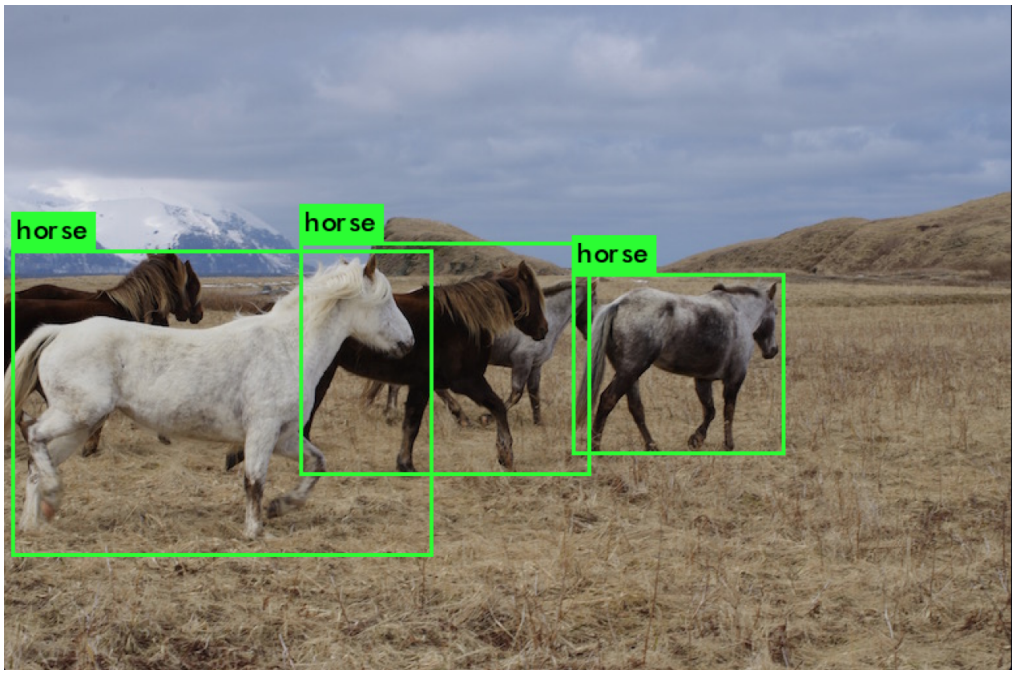


Figure 2: Example of object detection with YOLO classifier algorithm. (source)

There are several open source frameworks for training an AI algorithm to identify and find the desired object within an image. In this project we have tested Haar Cascade and YOLO. We have also tested a third method which does not use AI, but instead relies on copmuter vision techniques that creates a mask of the object of interest and uses that for detection. A brief explanation of these methods will be given below, but the implementation of these three methods will be elaborated later in this report.

Haar Cascade

Haar Cascade is a machine learning based approach where a cascade function is trained based on a set of positive and negative images [16, 29]. The positive images contain the object that you want to be identified while the negative images do not contain the object. In its most basic form, it finds features in a picture based on the difference between the sum of the pixels within the white rectangle versus the sum of the pixels within the black rectangle (These rectangles are known as Haar features shown in the figure below). In other words, it finds edges and darker landmarks that stand out within the image.

This approach is very efficient to identifying objects within an image, as it focuses on the parts of an image that have a higher chance of containing the desired object. More details on how it works can be found within the references.

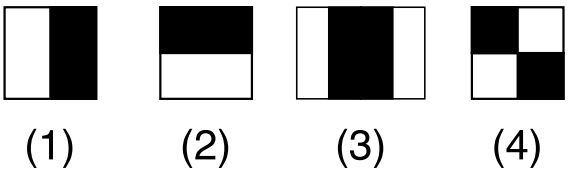


Figure 3: Examples of Haar Features (source)

YOLO

You Only Look Once (YOLO) object detection algorithm [19], and is one of the faster algorithms available as open source. The algorithm is fast enough for real-time applications and maintains a high accuracy. A benefit it has is the trade-off that can be done between speed and accuracy without having to retrain the model which otherwise would require a lot of time in training and computation power. In this project we use YOLO version 3 as it was the latest version published by its original authors [20, 21]. Later versions i.e. YoloV4 and YoloV5 are from different authors.

More conventional algorithms within object detection use classification to perform detection. YOLO differs in that it treats object detection as a linear regression problem, and applies a single neural network evaluation across the entire image. More detailed explanations can be found in their papers and website [19–21].

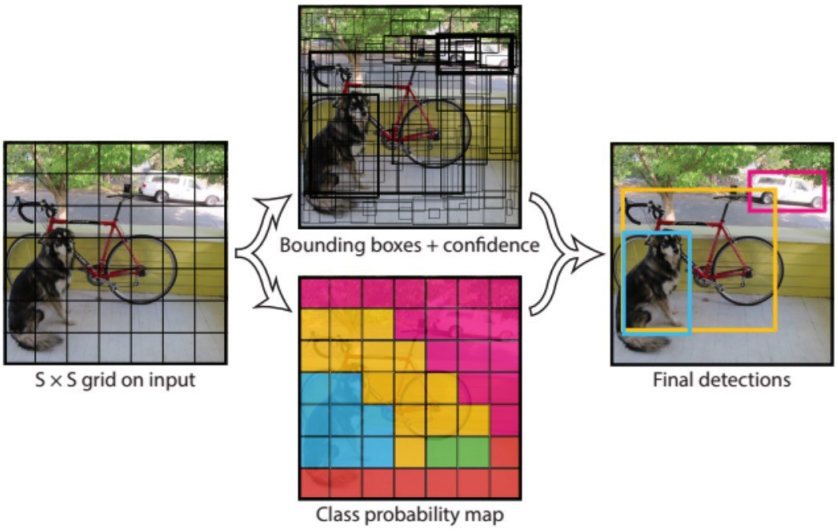


Figure 4: [YOLO] An input image is divided into grids, each grid cell predicts bounding boxes and confidence for these boxes. A class probability map is made and final detections is made. (source)

In comparison to YOLO, many Convolutional network (CNN) algorithms require the image to pass through evaluations several times to reliably detect the desired objects. This approach can be too slow for real-time usage, which is why we have decided upon using YOLO as one of our AI algorithms.

Computer Vision Techniques

The technique being described here detects the presence of an object by looking at its colors. This is done by adjusting the threshold values for the colors to create a mask like the one shown below [22].

Figure 5: Example of masking.

First, the input frame is preprocessed by applying some blur to it. This will make it easier to create a good mask. Afterwards, you localize the object by supplying a lower and upper HSV color boundary to a function which returns the mask as described. The last step is to convert the frame into the HSV color space and apply a series of erosions and dilations to remove any residual blobs that may be left in the mask.

A point worth considering when trying to get the best results from masking is to have the desired object be homogeneous in color. That specific color should preferably be unique as well within the frame to ensure that similarly colored objects do not show.

3.3 Kalman Filter

The measurements for a given instrument have the tendency to pick up noise and fluctuate in its readings. This is especially true for the data handled within this project. The measurements are based on predictions from the AI algorithms, which already have limited accuracy. Add to the fact that the drone is moving mid-air and the data will pick up a lot of noise and fluctuations in its readings. There are several ways to improve this, one of which is the Kalman Filter.

Kalman filtering is an algorithm that predicts the values of some unknown variables given the measurements observed over time [13], and are used on linear dynamical systems. Kalman filters have a broad range of applications and are often included in some form within navigation, control and measurements of land, water, and air vehicles, and even spacecrafts.

A brief explanation of how it works can be described as follows: You create a prediction based on data you have from before. This prediction is compared with the current measurement. These 2 values are weighed up against each other to see which one the algorithm will take into greater consideration. If the previous measurements gave a more accurate representation of the actual measurement, it will calculate a value closer to the current estimate.

On the other hand, if the previous predictions were more accurate, it will calculate a value closer to the current prediction. This value is defined as the updated state, and is the output of the Kalman Filter, which in turn will be used again as the input in the next iteration of the Kalman filter. The weighing of the values between prediction and measurement in each iteration is what defines the Kalman Filter and its numerous applications.

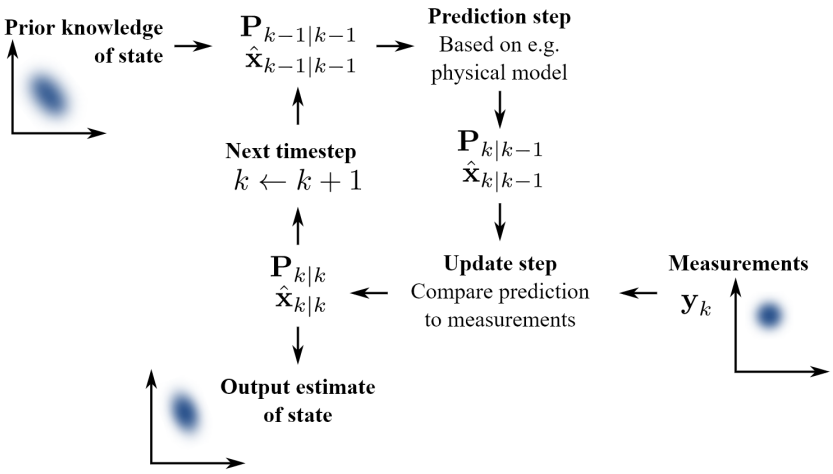


Figure 6: Kalman Filter representation. (source)

The algorithm can be divided into two stages, prediction and update:

$$\text{Predict cycle} = \begin{cases} \hat{x}_{k+1} = A\hat{x}_k^+ + Bu_k \\ P_{k+1} = AP_k^+A^T + LQL^T \end{cases}$$

$$\text{Update cycle} = \begin{cases} P_{k+1}^+ = P_k - P_kC^T(CP_kC^T + R)^{-1}CP_k \\ K_k = P_k^+C^TR^{-1} \\ \hat{y}_k = C\hat{x}_k \\ \hat{x}_k^+ = \hat{x}_k + K_k(y_k - \hat{y}_k) \end{cases}$$

Where:

x is the state vector.

u is the control input.

P is the state error covariance.

A, B, C, L show the relation of the different states with each other. (Transition matrices).

Q is the process noise covariance.

R is the measurement noise covariance.

y is the current measurement.

The hat operator ˆ, refers to the estimate for that given variable.

Q and R function as tuning parameters to adjust the performance and output of the filter. Estimates for these values can be calculated but are often not able to reflect how much noise is actually present.

Initial estimates for x and p also need to be given to the system.

For this project, the state vector X is composed of 3 outputs from the object detection algorithm. This includes the center coordinates of the object and its bounding box dimensions. The initial values also need to be filled out for $\hat{X}$ and $\hat{P}$. All these equations will then look like this:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \hat{X}_i = \begin{bmatrix} x_{1i} \\ x_{2i} \\ x_{3i} \end{bmatrix} \quad \hat{P}_i = \begin{bmatrix} p_{11} & 0 & 0 \\ 0 & p_{22} & 0 \\ 0 & 0 & p_{33} \end{bmatrix}$$

$\hat{P}_i$ is set to a large value based on the recommendation of "initial ignorance" [13]. The values for Q and R need to be tuned depending on the results. However, it is possible to get a rough idea on what the values should be. The assumptions are that the variables of the state vector are independent of each other with known variances [7]. Hence, the matrices Q and R look like this:

$$Q = \begin{bmatrix} q_{11} & 0 & 0 \\ 0 & q_{22} & 0 \\ 0 & 0 & q_{33} \end{bmatrix} \quad R = \begin{bmatrix} r_{11} & 0 & 0 \\ 0 & r_{22} & 0 \\ 0 & 0 & r_{33} \end{bmatrix}$$

Where all the values of the diagonal are variances. The variances in the diagonal of R matrix can be computed by measuring the data while the object is held in one place. You can then find the variance for each of the required variables. The Q diagonal is harder to compute, and is easier to adjust based on how well the kalman filter performs.

Once the initial values are in place, the Kalman Filter feeds the values for $\hat{X}$ and $\hat{P}$ back to itself. However, it also needs to be fed the current measurements, which follows the form of the state matrix (note that $[X_m]$ corresponds to y_k from the equations with the update cycle). It will then look like this:

$$X_m = \begin{bmatrix} x_{1m} \\ x_{2m} \\ x_{3m} \end{bmatrix}$$

After the desired values have been passed through the Kalman Filter, the resulting $\hat{X}$ matrix (the new predicted values) is used in the next section of the program, which is the PID controller.

3.4 PID

Proportional-Integral-Derivative (PID) control is one of the most widely used control algorithms within the industry [3, 4]. The idea behind how it works is that you decide a setpoint for a given process, and control the output in such a way that it moves toward the setpoint. The controls are given by the PID, which sums up the proportional, integral and derivative responses of the system to calculate the output. This forms a loop where the system feeds the error back to be able to update the values in real time.

The proportional factor K_p is the product of a certain gain and the measured error ε. This means the system reacts directly according to its error. Too high K_p will also cause it to overshoot and can lead to oscillation.

When the error becomes small, the output also becomes small and introduces a steady-state error where the output becomes stable but does not actually match the setpoint. The integral factor K_i fixes this by summing up the error over time, and adjusts the output based on this sum. This reduces the steady state error to zero as the output matches the setpoint.

The derivative factor K_d looks at the rate of change of the error ε. The more the error changes, the larger this derivative factor becomes and the faster the overall control system responds. It is however, highly sensitive to noise, and can cause the system to be unstable.

A general equation of a system with a PID controller looks like this [15]:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_p \frac{de}{dt}$$

Where $u(t)$ is the output of the system and $e(t)$ is the error between the desired output (set point) and the actual output.

Choosing the right parameters for the PID controller is crucial to get the desired result of the system. There are several ways to choose these parameters, also called tuning, and the ones discussed here include trial and error and Ziegler Nichols tuning.

The trial and error method can be done by following the steps below:

- Set all PID values to 0.
- Increase K_p until the desired response time. Too high K_p will cause oscillations.
- Increase K_i to stop oscillations and minimize steady-state errors. Too high K_i increases overshoot.
- Increase K_d until the loop achieves desired speed to move towards setpoint. Too high K_d will increase sensitivity to noise.

The Ziegler Nichols method works in a similar way as the trial and error method described above. However, K_p, K_i and K_d are calculated based on a critical gain K_c. K_c refers to the point where the system oscillates. A table with the calculations is shown below.

Figure 7: Ziegler-Nichols tuning

$Control$	P	T_i	T_d
P	$0.5K_c$	$-$	$-$
PI	$0.45K_c$	$\frac{P_c}{1.2}$	$-$
PID	$0.60K_c$	$0.5P_c$	$\frac{P_c}{8}$

There are other ways to tune the PID parameters more precisely using more extensive calculations to analyse the system. It is for example possible to see how the system performs by analysing its transfer function and its step response [8]. Key elements that can be analyzed include poles and zeros, overshoot, time constant etc. More detail regarding this will not be discussed as these methods have not been applied for this project.

4 Implementation and results

The implementation of the system for controlling the drone to track an object has been done solely through programming in Python. A flowchart of the system is shown below to get a quick overview and aid with the explanations.

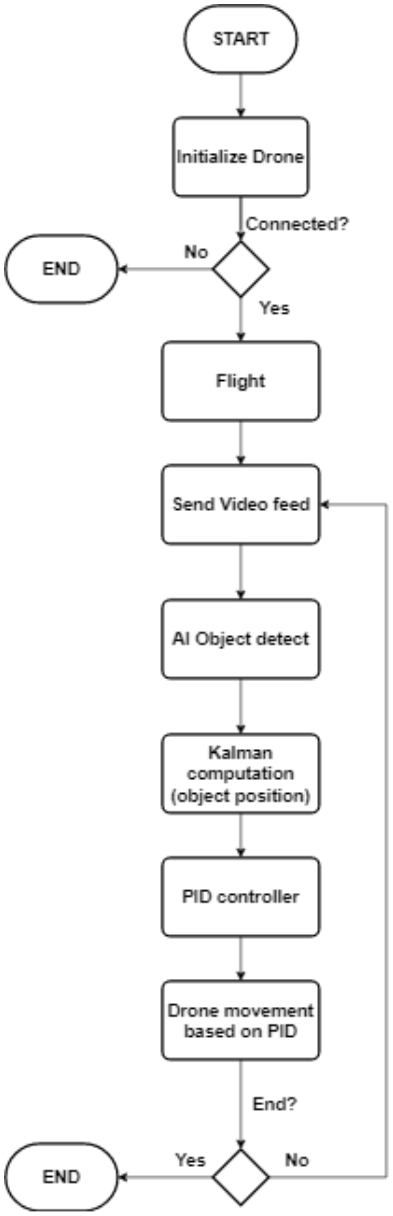


Figure 8: Simple flow chart of system

4.1 Connection and Control of Tello

As mentioned early in this report, Tello has an SDK that we use to establish a connection between the drone and our PC through wi-fi. The connection between these instances utilizes a communication protocol by the name UDP which stands for User Datagram Protocol. This protocol is primarily used for establishing low-latency and loss-tolerating connections and it speeds up transmissions by enabling data transfers, bypassing some checks [23].

Setting up the socket information, client and server ports and IP addresses can be a bit awkward if you are not familiar with sockets but can be done easily through DJI TelloPys application programming interface (API) which as the name implies, functions as an interface between the official Tello SDK and python to allow for easier programming [10, 11].

We mainly use the API to initialize a connection and send control commands which tells the drone which direction to move and at what speed and to take off and land.

In our system there are two ways of controlling the drone. One is by manually controlling it by giving inputs through the computer keyboard and the other is by activating the track function, giving the drone autonomous control based on the tracked object.

Figure 9: [Our manual controls] Blue controls roll pitch yaw, green for take off and land, dark red for switching between manual control and tracking, yellow to enable/disable OSD, bright red to end program, purple to switch between translation and rotation mode

4.2 Data from Tello

The drone has many different sensors that give a lot of information on everything from the drone's temperature, battery percentage, attitude, etc. All the information is serial data which is decoded, gathered and presented in a way that makes it easier to perceive what is happening to the drone which is crucial in debugging and can also aid in tuning parameters. Another type of data is the image (frame) transmitted from the drone and received by the computer program. In the figure below we can see a single frame received from the drone and displayed in a window. All the data available from the drone is drawn on this frame.

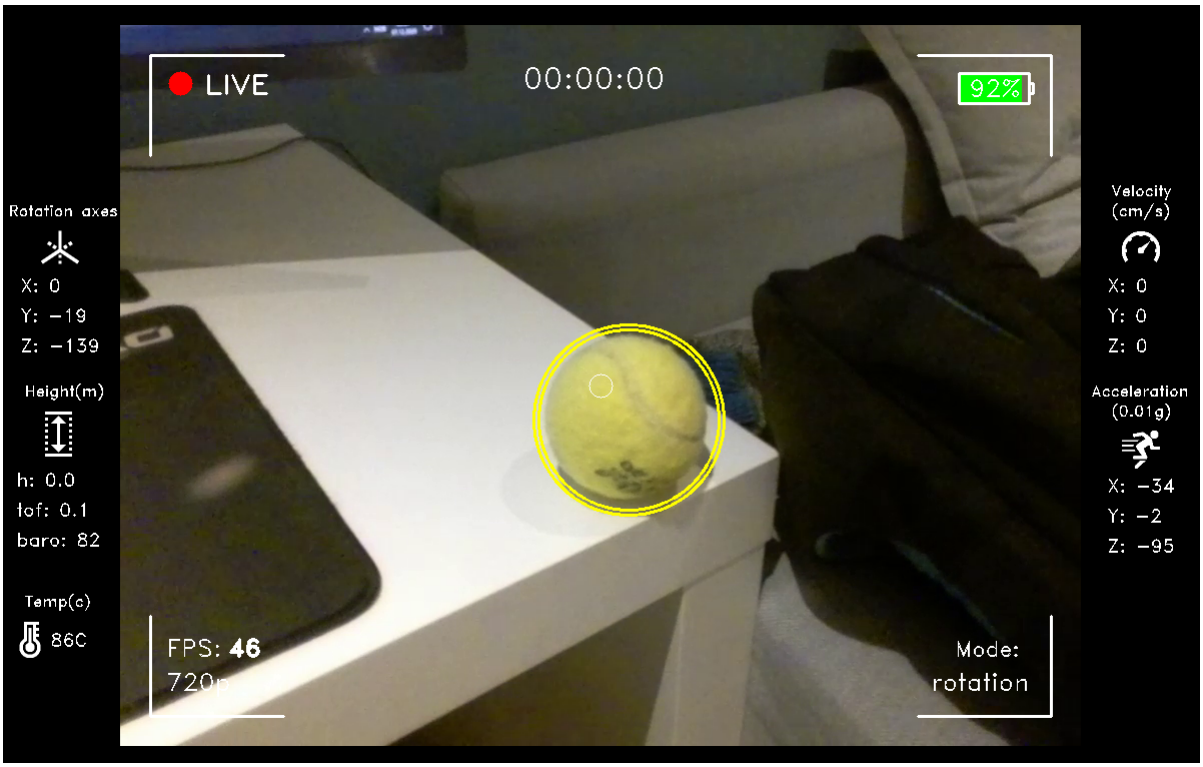


Figure 10: On screen display of a frame from Tello

Through this data we have noticed that the drone performs worse when its battery level is below a certain percentage and if the internal temperature is too high. The drone also stops operating normally and needs to be cooled down. This have been an issue in testing and tuning, but was partially solved by having multiple drones on hand so each drone can rest between testing.

4.3 Detection algorithms

As mentioned earlier in the paper, we have implemented and tested three methods for detection. Tthe reasons for trying out several detection methods was to see how they performed compared to each other and see which one could showcase the effects of the Kalman Filter and PID controller. In this section we will explain our findings and explain the pros and cons of each method.

Haar Cascade is provided by the computer vision package OpenCV which can be implemented with only a few lines of code and in terms of implementation is just as easy as the computer vision technique tested. In this report, the computer vision technique used will now be referred to as **HSV**. Haar Cascade is also not too computationally heavy and runs perfectly on the central processing unit (CPU). The method gives us acceptables frames per second (FPS), although one big concern is that it does not provide good enough detection. This often makes the drone confused as to what it should track. The algorithm often places boundary boxes in random places. An example image is provided below for clarification.

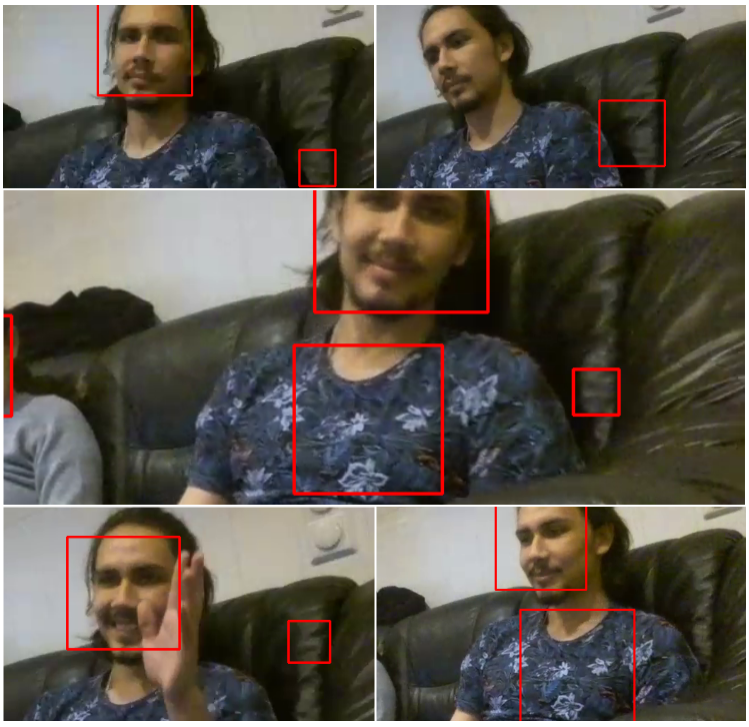


Figure 11: Boundary boxes from Haar Cascade algorithm.

The HSV technique on the other hand performs even better in terms of FPS and detection accuracy with the caveat of being the only unique color in the frame as explained in the theory section.

The last method is **YOLO** which was the hardest to implement, but allowed for more flexibility in terms of what objects to detect and had the most stable results. It was however, too computationally heavy that our hardware could barely keep up, giving us frames in the single digits. Further research was made to alleviate this problem by enabling the algorithm to utilize the graphical processing unit (GPU) which works better at computing image frames. This improved our FPS by 3-4 times which made the method viable.

One of the downsides is that retraining the model takes time. The original authors trained the model on 80 different classes which are commonly found in images. We wanted to track a person by their face, and had to retrain the model. This required many attempts and extensive time put into labeling custom images and training the model due to limitations in hardware. In the end we decided to train the model to detect six different classes which are: tomatoes, bananas, coffee cups, bottles, human face and tennis ball. The many attempts yielded a bit more than mediocre results but were still acceptable to be used for testing. The image below show examples of the results from our trained model.

Figure 12: Boundary boxes from YOLO algorithm.

Between Haar Cascade and YOLO, if the computer has been set up such that YOLO can use the GPU, it is the better choice. The lightweight model provided by YOLO is also good, if the object that you want to track is included in its list of classified objects.

In terms of pure FPS performance, the HSV method wins by a significant margin, but has its limitations in that it does not directly recognize an object in the frame, and instead filters colors to try to locate the object in question inside the frame.

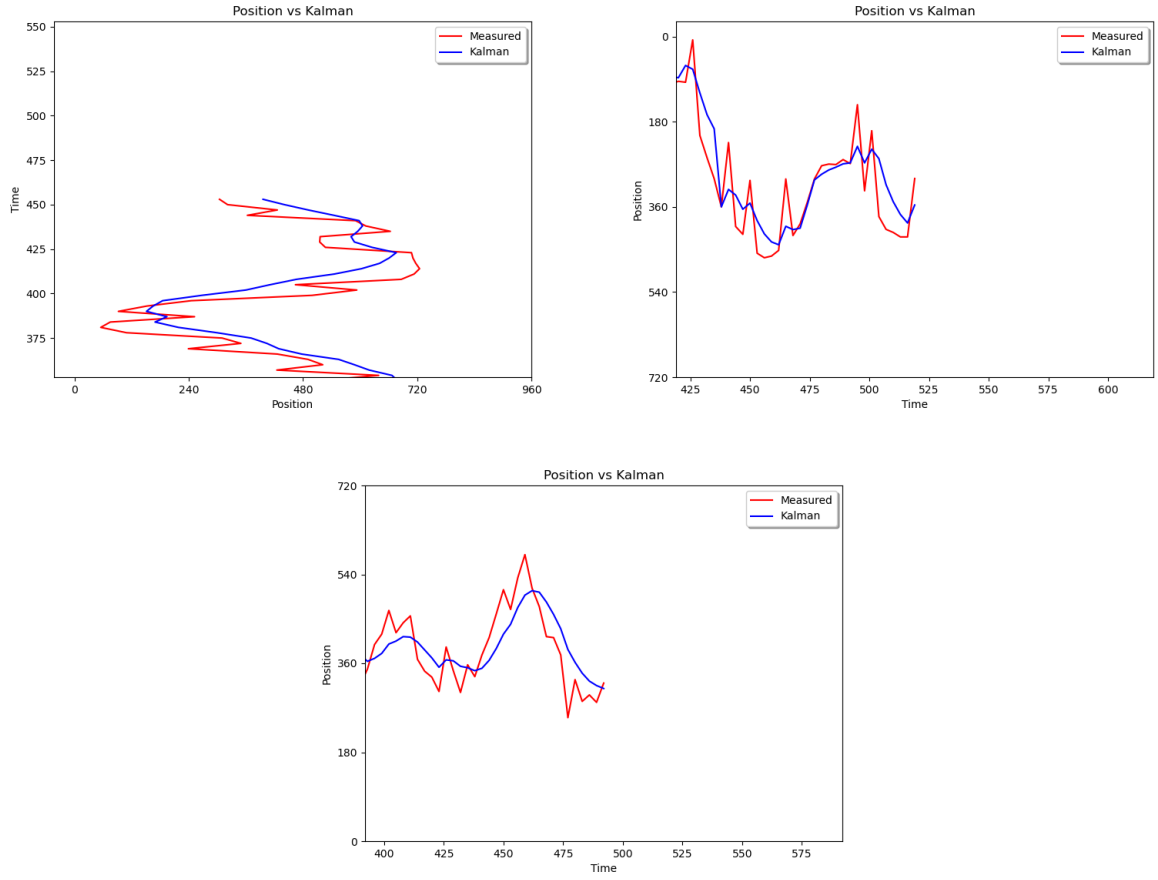
4.4 Kalman implementation

The values sent from the tracking algorithms contain the location and size of the boundary box which we pass through a Kalman filter to gain a smoother signal. This is because our detection method has quite a bit of noise. The algorithms for detecting have a tendency to lose signal every once in a while. The other point is that it for each frame constantly calculates the objects boundary box and its location. Therefore, the locations sporadically jump around even if the object is stationary. This is the reasoning behind the use of the Kalman Filter.

The center points x,y are self explanatory. They provide information on where the object is located with respect to the drone's camera. This allows us to move the drone left and right, up and down, and even rotate. However, x and y is not enough information to decide if the drone should move forward and backward. This is where the boundary box dimensions come in. When the object is closer, the boundary box becomes larger. The inverse also holds true. Note that the object being tracked needs to be uniform in size. Also, the bounding box will change in size depending on the object, which means separate calculations have to be performed such as when tracking a small ball compared to tracking a human face. To simplify calculations, only the height of the boundary box is used in calculating the distance and will be referred to as h_{bb}.

The values set in vector X, (x, y, h_{bb}) are the center coordinates of the camera given in pixels. h_{bb} is calculated based on the desired distance and size of the object in question. This is done by measuring the distance and height of the object from the drone camera point of view. The idea behind the values is that the object will most likely be placed in the center of the camera when turning on tracking.

The parameters Q and R required a little bit of testing to figure out the values that worked best. R was relatively easy to find, since the diagonal could be computed based on variance. Q on the other hand, required more time and had to be tested several times to find out a good tradeoff between speed and resistance against noise. The final results from tuning can be seen in the figures in the next page.



When comparing the values between the measurements and the predicted values from the Kalman filter, we see that the predicted values lag behind slightly compared to the measurements. This has been reduced as much as possible while still maintaining resistance against noise.

There is a noticeable difference in the performance of the drone when using the Kalman Filter. The movements are cleaner since the values sent to the PID controller are smoother with respect to time. There is still some lag present, but is still worth the tradeoff. Adjustments can instead be made in the PID controller for the drone to respond faster.

4.5 PID implementation

The Tello drone receives its control inputs in the form of velocity given in cm/s. Hence the output given by the system's PID is v(t). The error ε corresponds to the difference between the center point of the object c_o and the center point of the drone's camera c_c. Hence, we get $\varepsilon = c_o - c_c$. This calculation has to be performed for x, y, h_{bb}.

After finding ε for each point, we get a working equation that looks like this:

$v(t) = K_p E(t) + K_i \int E(t)dt + K_d \frac{dE}{dt}$, where E is a matrix containing: $E = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_{hbb} \end{bmatrix}$

A lot of time was spent in tuning the PID parameters, as every variable in the state vector X needs to be adjusted separately. Rotation could be tuned to react relatively quickly since the drone was still able to stabilize itself. Translation up and down could also be tweaked to be fairly reactive, which is important since the camera has less height compared to width. Translation going left/right and forward/back had to be tuned conservatively, as the drone did not stabilize itself fast enough.

Another issue that made tuning even harder was the performance of the drone. When the battery gets low or the temperatures go up, the motors react slower than usual.

The trial and error method has been used the most. The values acquired from Ziegler Nichols do not function well in our given system, but were still useful in getting an idea on how to tune the parameters.

Once the velocity has been calculated through the PID, the values get sent to the drone which then performs the desired movements.

5 Conclusion/Discussion

Implementing object tracking in drones present several challenges. There are many points in the process that need to work well to optimize the results. The first challenge is getting a reliable detection algorithm. Haar Cascade, YOLO and HSV performed relatively well during testing. Even with a good algorithm, there will still be some noise and inaccuracies in the measurements, and need to be filtered in some way. The implementation of the Kalman filter satisfied this requirement. Once you can accomplish that, you finally have data that can be used for calculating the movements and instructions to be sent to the drone.

The project has been successful in accomplishing its goal. However, there are many points that could still be improved. The detection algorithm is one of the weaker points. Preferably it should be more stable in its predictions and readings. The drone was also not able to react quick enough for sudden movements. This could also be partly due to limitations in hardware, for both the drone and the computer that performs the calculations for the algorithm.

References

[1] Federal Aviation Administration. *UAS by the Numbers*. 2020. URL: https://www.faa.gov/uas/resources/by_the_numbers/ (visited on 12/05/2020).

[2] Amazon. *Amazon Prime Air*. URL: https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011 (visited on 12/05/2020).

[3] Engineer Ambitiously. *PID Theory Explained*. Apr. 2020. URL: https://www.ni.com/en-no/innovations/white-papers/06/pid-theory-explained.html (visited on 12/05/2020).

[4] Paul Avery. *Introduction to PID control*. July 2015. URL: https://www.machinedesign.com/automation-iiot/sensors/article/21831887/introduction-to-pid-control (visited on 12/06/2020).

[5] Jason Brownlee. *4 Types of Classification Tasks in Machine Learning*. 2020. URL: https://machinelearningmastery.com/types-of-classification-in-machine-learning/ (visited on 12/01/2020).

[6] Jason Brownlee. *A Gentle Introduction to Object Recognition With Deep Learning*. 2019. URL: https://machinelearningmastery.com/object-recognition-with-deep-learning/ (visited on 12/01/2020).

[7] Math Works: Help Center. *State Estimation Using Time-Varying Kalman Filter*. 2014. URL: https://se.mathworks.com/help/ident/ug/estimating-states-of-time-varying-systems-using-kalman-filters.html (visited on 12/05/2020).

[8] Erik Cheever. *The Unit Step Response*. URL: https://lpsa.swarthmore.edu/Transient/TransInputs/TransStep.html (visited on 12/05/2020).

[9] Sam Daley. *Fighting fires and saving elephants: 12 companies are using AI drones to solve big problems*. 2019. URL: https://builtin.com/artificial-intelligence/drones-ai-companies (visited on 12/05/2020).

[10] Damia et al. Fuentes. *DJITelloPy*. 2018. URL: https://github.com/damiafuentes/DJITelloPy (visited on 12/01/2020).

[11] Damia et al. Fuentes. *DJITelloPy API Reference*. 2018. URL: https://djitellopy.readthedocs.io/en/latest/tello/#tello (visited on 12/01/2020).

[12] Business Insider Intelligence. *Drone market outlook: industry growth trends, market stats and forecast*. 2020. URL: https://www.businessinsider.com/drone-industry-analysis-market-trends-growth-forecasts?r=US&IR=T (visited on 12/05/2020).

[13] Youngjoo Kim and Hyochoong Bang. "Introduction to Kalman Filter and Its Applications". In: *Introduction and Implementations of the Kalman Filter*. Ed. by Felix Govaers. Rijeka: IntechOpen, 2019. Chap. 2. DOI: 10.5772/intechopen.80600. URL: https://doi.org/10.5772/intechopen.80600.

[14] J. Lee, J. Wang, D. Crandall, S. Šabanović, and G. Fox. "Real-Time, Cloud-Based Object Detection for Unmanned Aerial Vehicles". In: *2017 First IEEE International Conference on Robotic Computing (IRC)*. 2017, pp. 36–43. DOI: 10.1109/IRC.2017.77.

[15] Bill Messner and Dawn Tilbury et al. *Introduction: PID Controller Design*. URL: https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID (visited on 12/05/2020).

[16] Alexander Mordvintsev and Abid K. *Face Detection using Haar Cascades*. 2013. URL: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html (visited on 12/01/2020).

[17] Talya Porat, Tal Oron-Gilad, Michal Rottem-Hovev, and Jacob Silbiger. "Supervising and Controlling Unmanned Systems: A Multi-Phase Study with Subject Matter Experts". In: *Frontiers in Psychology* 7 (May 2016). DOI: 10.3389/fpsyg.2016.00568.

[18] Agung Prayitno, Veronica Indrawati, and Gabriel Utomo. "Trajectory Tracking of AR.Drone Quadrotor Using Fuzzy Logic Controller". In: *TELKOMNIKA (Telecommunication Computing Electronics and Control)* 12 (Nov. 2014), p. 819. DOI: 10.12928/telkomnika.v12i4.368.

[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. "You Only Look Once: Unified, Real-Time Object Detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[20] Joseph Redmon and Ali Farhadi. *YOLO: Real-Time Object Detection*. 2018. URL: https://pjreddie.com/darknet/yolo/ (visited on 12/01/2020).

[21] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *arXiv* (2018).

[22] Adrian Rosebrock. *Bacll Tracking with OpenCV*. 2015. URL: https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/ (visited on 12/07/2020).

[23] Margaret Rouse. *UDP (User Datagram Protocol*. URL: https://searchnetworking.techtarget.com/definition/UDP-User-Datagram-Protocol#:~:text=UDP%20(User%20Datagram%20Protocol)%20is,provided%20by%20the%20receiving%20party. (visited on 12/07/2020).

[24] Ulzhalgas Seidaliyeva, Daryn Akhmetov, Lyazzat Ilipbayeva, and Eric Matson. "Real-Time and Accurate Drone Detection in a Video with a Static Background". In: *Sensors* 20 (July 2020), p. 3856. DOI: 10.3390/s20143856.

[25] Xudong Sun, Pengcheng Wu, and Steven C.H. Hoi. "Face detection using deep learning: An improved faster RCNN approach". In: *Neurocomputing* 299 (2018), pp. 42–50. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.03.030. URL: http://www.sciencedirect.com/science/article/pii/S0925231218303229.

[26] Ryze Technology. *Tello*. 2018. URL: https://www.ryzerobotics.com/tello (visited on 11/21/2020).

[27] Ryze Technology. *Tello SDK 2.0*. English. Version Version 1.0. Ryze Technology. 2018. 7 pp.

[28] Ryze Technology. *User manual*. English. Version Version 1.3. Ryze Technology. 2018. 21 pp.

[29] P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features". In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 1. 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990517.

[30] Jin Zhou, Haibei Zhu, Minwoo Kim, and Mary L. Cummings. "The Impact of Different Levels of Autonomy and Training on Operators' Drone Control Strategies". In: *J. Hum.-Robot Interact.* 8.4 (Nov. 2019). DOI: 10.1145/3344276. URL: https://doi.org/10.1145/3344276.

[31] Pengfei Zhu, Longyin Wen, Xiao Bian, Haibin Ling, and Qinghua Hu. *Vision Meets Drones: A Challenge*. 2018. arXiv: 1804.07437 [cs.CV].