A machine learning approach to fast natural obstacle detection for drone flight using Tiny Yolov3

Lolézio Viora-Marquet

Vienna International School

Table of contents

Abstract 2

Introduction **3**

Methodology **4**

Physical constraints 4

Machine learning method 5

How does YOLOv3-tiny work? 6

Datasets 7

Training 7

Results **8**

Demonstration 10

Conclusion **12**

Steps forward **13**

References **13**

YOLO 13

ColANet: 13

LabelImg 13

Bibliography **14**

Abstract

Fast flying drones able to autonomously travel, while avoiding obstacles, can have many applications in industry. Numerous methods have been developed to tackle this problem, using a wide range of sensors, but I present a stereo-based approach, using a convolutional neural network to detect natural obstacles, in real-time, for fast and accurate obstacle detection. The model was tested on a test dataset representative of a coniferous forest environment, and outputted a mAP of 0.1209, indicating that it currently has a low success rate, however, that the approach could function with more training and a larger training dataset. My approach to obstacle detection is a valid solution to the problem, however, it needs to be significantly improved in order to be considered a reliable method for its real-world applications

Introduction

I have always been fascinated by technology, visiting museums such as the Technisches Museum Wien, and learning how to code in Scratch at around 7 years old. When I was offered the opportunity, I joined my school's robotics club, in order to learn about the mechanics of robotics, and how coding is applied to the physical world. I completed a number of projects, such as an Arduino-controlled autonomous greenhouse, or a GENIE controlled cleaning robot.

When I participated in the Summer STEM Institutes in 2020, a summer program focused on data science, I was taught how to conduct a data science research project, as well as the necessary Python programming skills. As part of the program, I had to start a project. The requirements were to read existing scientific papers on my chosen field of interest, find a research question, and find possible existing dataset that I could use to answer the question. The program approved my research proposal, and once the program came to an end in August 2020, I took on the task of researching more, designing and completing the project, fully independently.

Before finding my idea, I read a number of robotics papers, such as "*Limits to Compliance and the Role of Tactile Sensing in Grasping"* [1], a paper by a research team at the Harvard Robotics Lab. However, conducting my project in the field of anthropomorphic underactuated robotics hands, and helping in their development would be costly and unrealistic (as I do not have a lab). One paper that particularly interested me was "*High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo*" [2], written by researchers at the Computer Science and Artificial Intelligence Lab at MIT. Their research focused on obstacle avoidance for small aircraft, using an uncommon approach of a "pushbroom", which consists of detecting obstacles at a single *d* distance, meaning that only one variable had to be calculated, thus speeding up the processing speed of the aircraft, before using previous data (another *d* distance) to map out the surroundings left and right.

Unmanned aerial vehicles (UAVs), more commonly known as drones, have always been interesting to me, due to their ability to fly at high speeds for a cheap cost. As drone flying is one of my hobbies, I understand the difficulty of flying them, and the hours of training necessary to master the flight, without crashing in often constrained environments. Therefore, after reading about the current research on UAV obstacle detection and avoidance, I decided to conduct my project on this as well.

My initial research question was "How can we make collision avoidance faster?". My goal was to ensure a drone, which would embark GPS solely for the purpose of knowing its location, could travel from a point A to a point B, located using GPS, as fast as possible, with no prior knowledge of the terrain. This means it would have to travel at high speeds without crashing into obstacles, and so a fast obstacle avoidance algorithm must be developed.

---

[1]Robert D. Howe, Leif P. Jentoft and Qian Wan, "Limits to Compliance and the Role of Tactile Sensing in Grasping", *Proceedings of IEEE International Conference on Robotics and Automation*, *Harvard University*, 2014

[2]Andrew J. Barry, Peter R. Florence and Russ Tedrake, "High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo", *Computer Science and Artificial Intelligence Lab Massachusetts Institute of Technology,* 2017

Methodology

Physical constraints

Drones possess a number of physical constraints, due to their small size which ensures minimal energy use for their flight (as a heavier drone requires more energy to fly, which requires larger batteries, which are heavier, etc). For this reason, the computing power of a drone is limited, while it must be done at high speeds, to ensure the drone is able to detect the obstacles early enough to avoid them. A possible solution to this is cloud-based data processing, which consists of the drone sending the data from its sensors to a nearby computer with much larger computing power, which then sends it back to the drone. This possesses many limitations, however, such as latency in information transfer, especially for video data which requires large bandwidth. Due to the high speed of the drone, this could cause it to crash if the data transfer is sub optimal.

To answer the question, I first had to have an idea of what the hardware of my drone would look like. I compared 4 main existing techniques, namely LIDAR, ultrasounds, infrared and stereo image. All possess their drawbacks, and the table below compares them.

Table 1: Comparison between obstacle detection techniques

Technique	Advantages	Disadvantages
LIDAR[3]	Fast Works in low-light environments Provides a 3D image Measures very precisely	Only works in a predetermined range Low adaptability
Ultrasound[4]	Work in low-light environments Cheap Great accuracy	Slow Small field of view Accuracy affected by soft objects and temperature Limited range
Infrared[5]	Low energy consumption Work in low-light environment Fast Accurate	Accuracy affected by dust and other naturally occurring particles Small field of view Limited range Outdoor applications are limited
Stereo (RGB) image[6]	Large field of view High resolution Neural network can be used more easily Cheap Fast Large range	Does not work in low-light environments No depth sensing Only a 2D map of the surroundings

[3] “What Is LiDAR and Why LiDAR.” *LeddarTech*, leddartech.com/why-lidar/.
[4] Gross, Kristin. “Ultrasonic Sensors: Advantages and Limitations.” *MaxBotix Inc.*, Publisher Name MaxBotix Inc.Publisher Logo, 28 Oct. 2020, www.maxbotix.com/articles/advantages-limitations-ultrasonic-sensors.htm/.
[5] “Advantages and Disadvantages of Infrared Sensor.” *GeeksforGeeks*, 21 Dec. 2020, www.geeksforgeeks.org/advantages-and-disadvantages-of-infrared-sensor/.
[6]Shunguan Wu, “A sketch showing the pros and cons of both stereo vision and radar sensors”, *Intelligent Vehicles Symposium, 2008 IEEE*, 2008

As can be seen in the table, all methods possess their advantages and disadvantages, however, I decided to use stereo imagery due to the facilitated data collection (there are a number of available datasets online, and my drone possesses a stereo camera), and its other advantages which severely outweigh the disadvantages. Most research also uses stereo vision, and so this route would help me the most in terms of finding information and learning enough to be able to complete the project.

Machine learning method

After determining how data would be collected, I had to determine how it would be analysed. I quickly found a number of existing convolutional neural networks, which are currently the fastest and most accurate networks that exist in computer vision.

Object detection consists of two parts: image classification, and object localization[7]. The first adds a label to an image, identifying that there is an object of type *person*, for example, in the image. The second, however, draws a bounding box around the object, and so therefore localizes it within the given image. More commonly, however, both these techniques are combined into object detection, which draws a bounding box around the object, and then gives it a label. Such an approach is what I decided to do, so that the drone could use its stereo image input to detect obstacles by drawing bounding boxes around them, before identifying that they were in fact obstacles, and so that they must be avoided.

All apply a convolution on an image, which corresponds to applying a "filter" on the input, giving an idea of where an object the model was trained to detect is. Applying multiple convolutions, through the use of convolutional layers outputs a feature map, which indicates the location and strength of possible detections[8]. The more convolutional layers there are, the more accurate the model is, generally.

The two main families of convolutional neural networks that perform object detection are R-CNNs, which stands for Region-Based Convolutional Neural Network, and YOLO, which stands for You Only Look Once. R-CNNs are generally more accurate, however much slower, and so cannot be used in real time. For this reason, I started to research the existing YOLO networks, and how they work.

YOLO was developed by Joseph Redmon, et al., and presented in the paper "*You Only Look Once: Unified, Real-Time Object Detection"* [8]. There are 3 generations of YOLOs, Yolov1, Yolov2 and Yolov3. Yolov3 is the most recent, as the name suggests, and is also the fastest and most accurate. There are two Yolov3 model, the standard Yolov3, and Yolov3-tiny, which is much smaller and faster than the original, however, it is much less accurate[9].

I therefore decided to use Yolov3-tiny, due to its small size (perfect for the small microcontroller a drone can carry), speed (as it can run at a speed up to 244fps, compared to 155fps for Yolov3) and low GPU requirements (it only needs 611mb of GPU)[10].

---

[7] Brownlee, Jason. "A Gentle Introduction to Object Recognition With Deep Learning." *Machine Learning Mastery*, 26 Jan. 2021, machinelearningmastery.com/object-recognition-with-deep-learning/.

[8]Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", Cornell University, 2015

[9]Brownlee, Jason. "How to Perform Object Detection With YOLOv3 in Keras." *Machine Learning Mastery*, 7 Oct. 2019, machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/.

[10]Rachel Huang, Jonathan Pedoeem and Cuixian Chen, "YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers" *Georgia Institute of Technology*, 2018

How does YOLOv3-tiny work?

The reason YOLO is faster than other models is due to the fact that it only looks at the input image once. First, it divides it into a sxs grid, where s is a value determined based on the wanted speed or accuracy (a larger s value gives more accuracy but less speed)[11]. Each grid cell created then predicts a bounding box with its center in the grid cell, creating a number of bounding boxes all over the image. A class probability map is also created, and both are combined into giving the final result, with bounding boxes around the object, and a class label on it. Figure 1, taken from the original YOLO paper "*You Only Look Once: Unified, Real-Time Object Detection*" [8], represents how YOLO works.

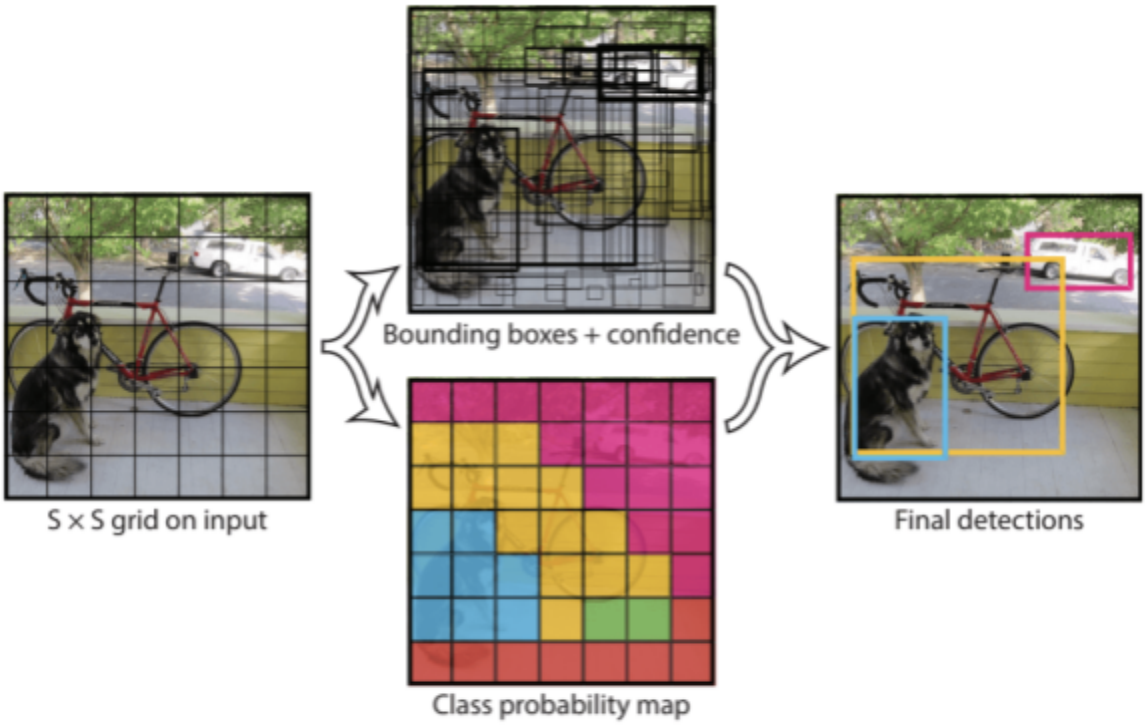

Figure 1: visual representation of YOLO's detection of an object

YOLOv3-tiny works in the same way, except it has many less convolutional layers, allowing it to be much smaller.

The "Darknet" framework, found at the Github repository https://github.com/AlexeyAB/darknet, is an easy way to use the YOLO model, and so I chose to use it to train, and later run my model. It requires a number of files to run, but once they are provided, it can be trained and used for any datasets.

[11]Brownlee, Jason. "A Gentle Introduction to Object Recognition With Deep Learning." *Machine Learning Mastery*, 26 Jan. 2021, machinelearningmastery.com/object-recognition-with-deep-learning/.

Datasets

To train the model, I had to create my own dataset, which would consist of images where I had drawn the bounding boxes of obstacles, which the model could then use to train at detecting. During the early stages of the project, and as part of the SSI program, I researched and found a number of open source datasets which I could use to answer my research question. The ColANet dataset[12], created by Dário Pedro, et al., contains 96 videos of drones colliding into obstacles, as well as frame by frame images of the collisions. The UZH-FPV Drone racing dataset[13] contains a number of RGB stereo images of a drone flying around a racing track, piloted by a professional pilot, avoiding the obstacles. Both these datasets provide examples of obstacles drones face, in a number of different environments, light settings etc. Originally, I planned to use the first to "teach" the drone what not to do, and the second to "teach" the drone what it should do when facing obstacles. However, after inquiring how I would do this, I found out this was not a realistic nor possible approach. Therefore, I settled to simply create my own dataset with images from both of these, then draw bounding boxes around the obstacles, before training Yolov3-tiny on such a dataset. YOLO requires a very specific dataset format, which requires .jpg images, which each have a .txt file with the x and y coordinates of the bounding boxes. The most common tool, which supports the YOLO format, is LabelImg, https://github.com/tzutalin/labelImg, which I was able to run in Python using VS Code on my computer.

Drawing the bounding boxes was harder than expected, as I had to consider a number of things, such as whether the drone should consider a far away object an obstacle. As stereo cameras provide no information about the depth of the environment, I decided to only highlight obstacles very close to the drone, in an effort to make the model "biased" towards obstacles closer to the drone, in order for them to be detected. In short, when preparing my dataset, I had to make a number of decisions on what exactly I wanted my drone to detect, as each small decision would be amplified by the model iterating through the images millions of times.

Training

After annotating 100 images, which I had taken from the ColANet dataset, I decided to train my model, and then test it. I had to download a number of environments on my computer, and Python libraries which facilitate training such as OpenCV were difficult to install, and so I was not able to do this. Originally, a large number of bugs occurred after installing Darknet, and trying to run it on VS Code, but eventually I was able to start training, after having modified the setup files to be able to train on my dataset with my single object class "Obstacle", on Yolov3-tiny, and for maximum speed. After repeatedly crashing, I was able to launch a good training session, which however predicted a training time of 2 months. This was due to the fact that my computer does not have a GPU, and CPU is considerably slower for using and training the model. Therefore, I had to research what my other options were.

As part of SSI, I had learned to use Google Collaboratory, in order to complete some Python and machine learning exercises. It also provides 60gb of temporary hard drive storage, as well as 12gb of GPU, for free. It can also be connected to a google drive account, where I could store my dataset and configuration files.

Even on Google Collab, the training was slow. Another problem I faced was that Google Colab only allows you to connect to an environment for 12 hours, which for my model represented only about 20,000 iterations through the dataset (which corresponds to iterating through a million images), leaving me with an insufficient model, with a mAP of 3.57 (100 being a perfect model which succeeds everytime). After researching, I found

[12] "A UAV Collision Avoidance Dataset." *COLANET*, colanet.qa.pdmfc.com/.
[13] "Department of Informatics." *The UZH-FPV Drone Racing Dataset*, rpg.ifi.uzh.ch/uzh-fpv.html.

out it was possible to start training again from a checkpoint, which allowed me to significantly increase the training time, allowing me to complete over 50,000 iterations through the dataset to get a satisfying model.

Results

In order to test my model on a realistic dataset, I flew my drone in the Austrian alps, in a forest environment, in order to collect images of a possible environment the drone could have to fly through, with space out obstacles. I then annotated these images using LabelImg, and tested the model on whether it could detect obstacles on the images.

In computer vision, there are a few metrics which are used to determine how good the model is[14]:
Precision is simply a measure of how many predictions are correct, as a percentage (0 to 1). It is calculated by the formula $Precision = \frac{TP}{TP+FP}$, where TP is the number of true positives (obstacles correctly detected), and FP is the number of false positives (the model detected an obstacle that does not exist). In other words, $Precision = \frac{TP}{total\ positive\ results}$.

Recall is an indication of how many of the positives the model identifies, once again measured as a percentage (0 to 1). It is calculated by the formula $Recall = \frac{TP}{TP+FN}$, where FN is the number of false negatives (the model did not detect those obstacles). In other words, in the case of my model, $Recall = \frac{obstacles\ correctly\ detected}{total\ number\ of\ obstacles\ on\ the\ image}$.

IoU, which stands for Intersection over Union, is simply a measure of how much the bounding box predicted by the model intersects over the ground truth (the real bounding box of the obstacle, determined by me). It is calculated by the formula $IoU = \frac{area\ of\ overlap}{area\ of\ union}$, where the overlap is the intersection between the area of the prediction and ground truth, and the union is the combined area of both of these. A perfect IoU, when the bounding box predicted is exactly the same as the ground truth, therefore has a value of 1.

AP, which stands for average precision, is found by integrating the graph of recall (on the x axis) and precision, on the y axis, giving a value between 0 and 1 as well. This value then gives a good general idea of the accuracy of the model, between the number of correctly identified obstacles, and how many false positives it declares.

mAP, which stands for mean Average Precision, is simply the average AP for each class of the model. Since my model only has 1 class, “obstacle”, it will be the same as the AP.

[14] Hui, Jonathan. “MAP (Mean Average Precision) for Object Detection.” *Medium*, Medium, 3 Apr. 2019, jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173.

Table 2: mAP, precision and recall for a confidence threshold of 0.25, and IoU threshold of 50% tested on 5 images.

Model	Precision	Recall	mAP	Average IoU
Earliest version of the model (10,000 iterations)	0.45	0.34	0.0208	21.5%
20,000 iterations	0.55	0.47	0.0357	25.7%
40,000 iterations	0.56	0.58	0.0795	29.2%
Final (50,000 iterations)	0.73	0.64	0.1023	32.4%

These results are however presented with a confidence threshold of 0.25, which means if the model is 25% sure that there is an obstacle, it will count it as a prediction. However, in the case of the drone, it is better to falsely detect an obstacle, go around it, and lose a bit of time (even if the goal is for the drone to go to point B as fast as possible) than to crash in an obstacle, rendering the drone unoperational, which is costly and time consuming. Therefore, I tested the drone with a lower confidence threshold, of 0.1, to see if this impacted the mAP, and resulted in a more reliable model.

Table 3: mAP, precision and recall for a confidence threshold of 0.1, and IoU threshold of 50% tested on 5 images.

Model	Precision	Recall	mAP	Average IoU
10,000 iterations	0.52	0.33	0.0310	24.9%
20,000 iterations	0.52	0.46	0.0456	26.3%
40,000 iterations	0.67	0.64	0.1004	29.5%
Final (50,000 iterations)	0.84	0.76	0.1209	33.2%

From the data, a trend is clearly noticeable: as the number of iterations increase, the precision, recall and mAP of the model increase, indicating that the training is successful in making the model better. When the confidence threshold is decreased, increasing the chances that an obstacle is detected by the model, the mAP also increases. This indicates that the model declares too many false negatives, and that predicting less probably obstacles helps the model miss less obstacles. The average IoU of the model also follows this trend, indicating that the bounding boxes become closer to the ground truth when more are predicted.

Overall, however, these values remain quite low, the mAP only at 0.1209 or 12.09% when the threshold is decreased. This is not satisfactory, as the drone is likely to have many false positives (making it avoid an obstacle that does not exist and so lose time) and false negatives (not detect an obstacle, crashing, possibly breaking).

Such low numbers can however be explained by a number of factors. The first main reason is that my dataset is quite small (only 109 images), compared to datasets such models usually get trained on with thousands of

images, such as COCO with 5,000 images for each class. The images of the dataset are not very varied, and most are taken from high heights, with a clear obstacle, that the model can detect. However, I tested the dataset with more of a forest environment, with large trees, which the model sometimes failed to detect. Another reason why the model performed poorly is that the training time was rather short, and I was unable to finish training. Due to not possessing any GPU on my computer, and having to use Google Colab, the training had to be cut off after 12 hours, sometimes earlier when the system crashed. This means training was rather difficult, and so I was only able to train the model for about 15 hours, resulting in only 50,000 iterations. The average IoU when I had to stop training was only at around 90%, which means that training was not finished when I had to stop it. This explains the mAP of 0.9949 when testing the model on its training dataset, which indicates that more training time would have helped the performance of the model. These reasons explain the low success rate of the model, and they must be fixed in order to improve it.

Demonstration

For demonstration purposes, I also tested the model on a video I filmed with my drone. On 12gb of GPU, the model was able to run at 35fps, which is much slower than the 244fps Redmon, et al. are able to achieve, most likely due to the comparably low computing power of Google Colab. However, this speed was sufficient to demonstrate that my model can be used in real time, to detect obstacles. This video, as well as this one demonstrate that the drone was able to detect all the obstacles I made it avoid when filming, and this suggests the model can be used to fly the drone without crashing in an obstacle.

I found that the model performed better in open spaces with little obstacles, as can be seen in Figure 2, where the model was able to detect two obstacles, with which the drone would have collided. This obstacle is sufficient for a path planning algorithm to steer the drone in between two obstacles, even if the bounding boxes are quite inaccurate.

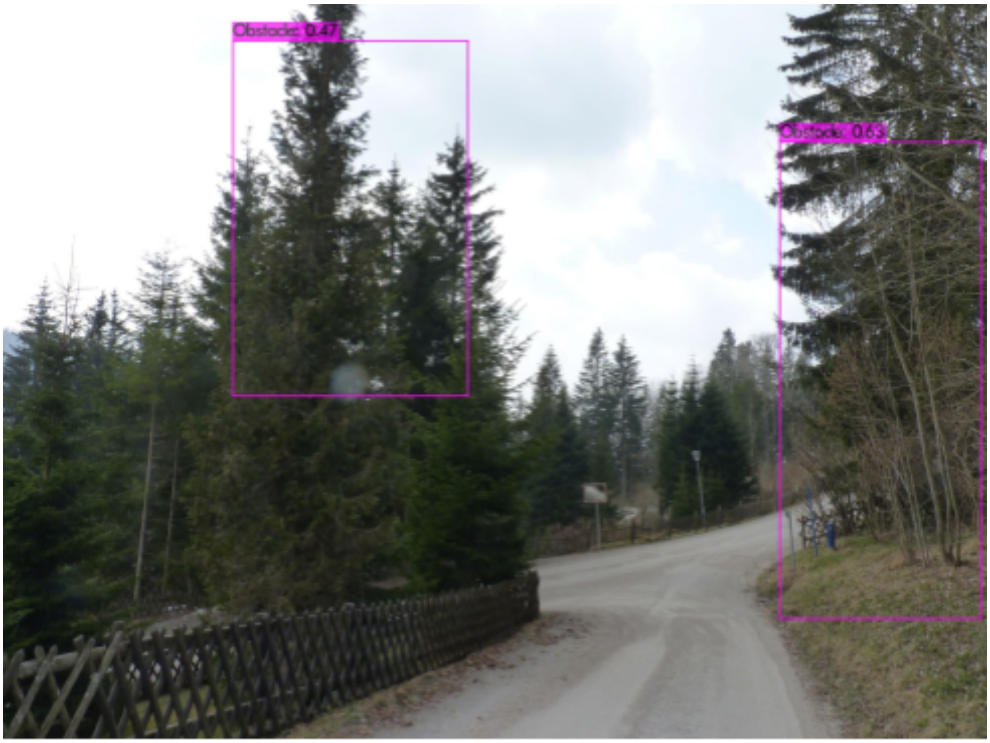

Figure 2: Image taken by my drone, where the model detected 2 obstacles, with a minimum threshold of 0.25

Another successful example can be seen in Figure 3, where the model detected an obstacle, however, its bounding boxes (left) were different from those I had originally drawn on the image (right). The model was able to draw correctly by different bounding boxes, as a drone avoiding those boxes (by flying up for example) would effectively avoid the obstacle, even if originally this was not my idea of the boxes for this image.

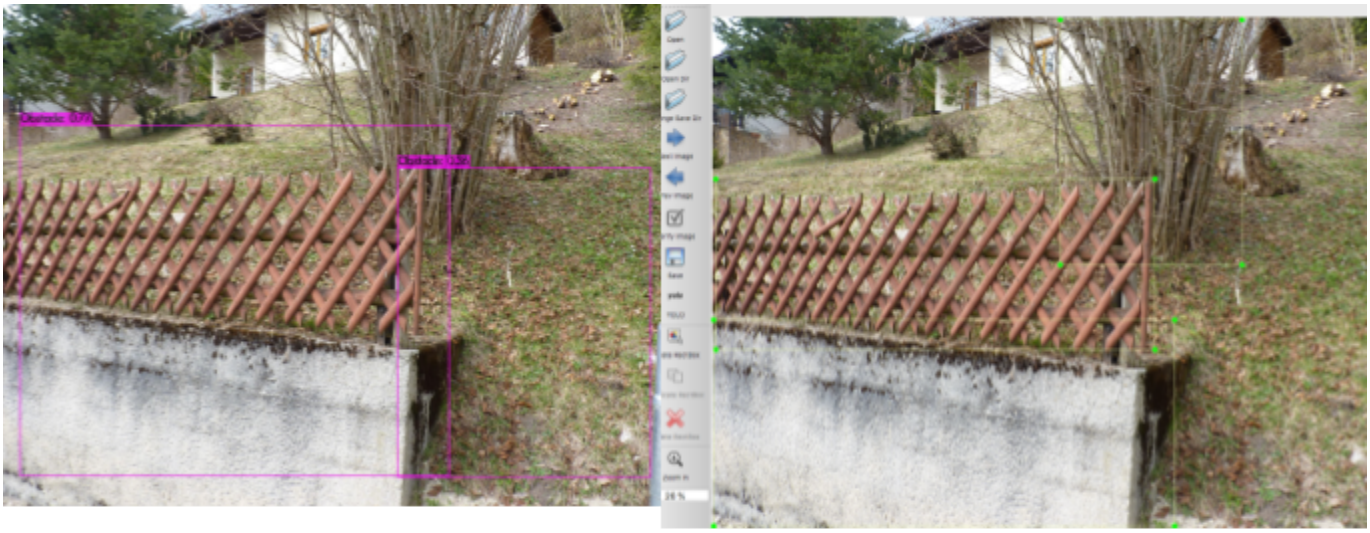

Figure 3: comparison of the bounding boxes drawn by the model (left), and those that I had originally drawn on the image (right) using LabelImg. The model was able to innovate and find different but correct obstacles.

When tested on the images it was trained on, the model was able to reach an mAP of 0.9949, indicating that the training was successful for those training images (though not complete, as it should have been exactly 1). This is demonstrated by this video, taken from the ColANet dataset, where my model is able to detect an obstacle in which the drone crashes.

When watching the videos, I noted that the obstacles were detected very late, almost when the drone crashes in them. Instinctively, my first reaction was that this means the drone will crash before detecting the obstacles. However, this was how I planned to counteract the fact that stereo cameras cannot measure depth, and so solely possessing a stereo camera, the drone would be unable to tell whether the obstacles were far or close, losing time to avoid obstacles which were still far, resulting in a time loss. For this reason, I had decided to only highlight obstacles close to the drone in the images I had from ColANet, to train the model to only detect obstacles close to it. This proved to be effective, as can be seen in the videos linked in this report, where the model detects the obstacles only a few frames before it crashes, allowing it to maneuver at the last second, gaining time, as travelling in a straight line is faster than going around an object (and so the drone should travel as straight as possible).

Conclusion

Despite the low mAP of the model, I believe my model can be used to make collision avoidance faster, and is therefore a valid solution to the question. Yolov3-tiny has a mAP of 0.331 on the COCO dataset[15], used to test various CNNs, which means it already has low precision and recall (though it is very fast), and so my result of 0.1209 is not very far from the original, considering my dataset is harder to use (as the obstacles have varying shapes, colors, and they are sometimes closely packed together), while the training time of the model could also be improved. My model is able to solve a number of problems when detecting obstacles that a drone could face when using other techniques such as LIDAR or ultrasound, whose accuracy depends on the conditions, quantity of dust, pre-set range, etc. Moreover, Yolov3-tiny is faster than any other current CNN, and though it is less accurate, it is able to fulfill its role of real time obstacle detection, as shown by my results. It is also able to detect obstacles only at close range, allowing the drone to spend less time avoiding obstacles, and more travelling in a straight line, overall making the obstacle avoidance faster. My model can however be greatly improved, by using a larger dataset, with more varied training images, and more training time, to be able to finish training (as the mAP when testing on the training dataset is 0.9949 instead of 1, indicating it has not yet finished training). This could then have a number of applications, to reach and film zones inaccessible to humans, without the need for a pilot which are costly, as piloting a drone at high speeds requires large amounts of training and good reflexes. This technology could therefore be used to save lives or decrease the environmental impact of drones, which is my original goal when working in the field of robotics.

Steps forward

The first step forward would be to increase the mAP of the model, using a larger and more varied training dataset, as well as more training time. However, once the model is more than satisfactory, and detects obstacles with great precision and recall, a path planning algorithm should be implemented, for the drone to find a path from Point A to Point B, determined by GPS, avoiding obstacles. A possible method for this could be to annotate images with the obstacles drawn out (by the model, for example) with directions on where the drone should go in each situation. This approach could result in the drone learning how to detect obstacles, and then how to avoid them, as fast as possible. Once this fully works, and can be trusted enough to fly a real drone, a drone should be built with the necessary GPU computation to run the model in real time, as well as a large angle camera. An emphasis should be put on building a fast drone, to ensure it is able to travel from Point A to B as fast as possible. This would create a drone ready for industrial development, able to travel fast between two points, helpful for possible reconnaissance missions or transporting payload to inaccessible areas, without the control of a human pilot, greatly reducing the cost of such an operation.

[15] Ultralytics. “Unable to Reproduce MAP with yolov3-Tiny.weights · Issue #188 · Ultralytics/yolov3.” *GitHub*, github.com/ultralytics/yolov3/issues/188.

References

YOLO

Paper: https://arxiv.org/abs/1804.02767
Website: https://pjreddie.com/darknet/yolo/
Github repository: https://github.com/AlexeyAB/darknet

ColANet:

Website: https://colanet.qa.pdmfc.com
Github repository: https://github.com/dario-pedro/uav-collision-avoidance

LabelImg

Github repository: https://github.com/tzutalin/labelImg

Bibliography

Andrew J. Barry, Peter R. Florence and Russ Tedrake, “High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo”, *Computer Science and Artificial Intelligence Lab Massachusetts Institute of Technology,* 2017

“Advantages and Disadvantages of Infrared Sensor.” *GeeksforGeeks*, 21 Dec. 2020, www.geeksforgeeks.org/advantages-and-disadvantages-of-infrared-sensor/.

Brownlee, Jason. “A Gentle Introduction to Object Recognition With Deep Learning.” *Machine Learning Mastery*, 26 Jan. 2021, machinelearningmastery.com/object-recognition-with-deep-learning/.

Brownlee, Jason. “How to Perform Object Detection With YOLOv3 in Keras.” *Machine Learning Mastery*, 7 Oct. 2019, machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/.

“Department of Informatics.” *The UZH-FPV Drone Racing Dataset*, rpg.ifi.uzh.ch/uzh-fpv.html.

Gross, Kristin. “Ultrasonic Sensors: Advantages and Limitations.” *MaxBotix Inc.*, Publisher Name MaxBotix Inc.Publisher Logo, 28 Oct. 2020, www.maxbotix.com/articles/advantages-limitations-ultrasonic-sensors.htm/.

Hui, Jonathan. “MAP (Mean Average Precision) for Object Detection.” *Medium*, Medium, 3 Apr. 2019, jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173.

Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi, “You Only Look Once: Unified, Real-Time Object Detection”, Cornell University, 2015

“A UAV Collision Avoidance Dataset.” *COLANET*, colanet.qa.pdmfc.com/.

Ultralytics. “Unable to Reproduce MAP with yolov3-Tiny.weights · Issue #188 · Ultralytics/yolov3.” *GitHub*, github.com/ultralytics/yolov3/issues/188.

“What Is LiDAR and Why LiDAR.” *LeddarTech*, leddartech.com/why-lidar/.

Rachel Huang, Jonathan Pedoeem and Cuixian Chen, “YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers” *Georgia Institute of Technology*, 2018

Robert D. Howe, Leif P. Jentoft and Qian Wan, “Limits to Compliance and the Role of Tactile Sensing in Grasping”, *Proceedings of IEEE International Conference on Robotics and Automation*, *Harvard University*, 2014

Shunguan Wu, “A sketch showing the pros and cons of both stereo vision and radar sensors”, *Intelligent Vehicles Symposium, 2008 IEEE*, 2008