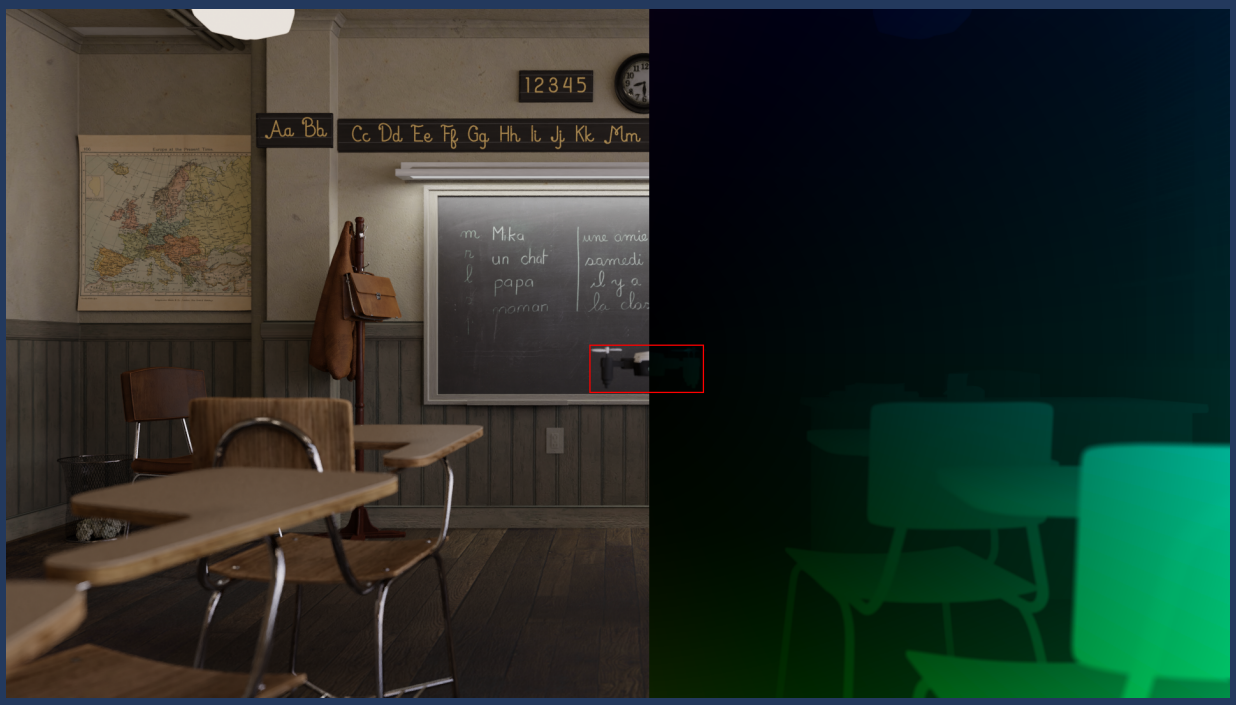
Deep Vision-based Relative Localization by Monocular Drones

R.A.A. Mink



TU Delft

Deep Vision-based Relative Localization by Monocular Drones

Thesis report

by

R.A.A. Mink

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on February 3, 2023 at 09:30

Thesis committee:	
	Prof. dr. G.C.H.E. de Croon (chair)
	Dr. ir. C. de Wagter
	Dr. ir. D. Zarouchas
Place:	Faculty of Aerospace Engineering, Delft
Project Duration:	March, 2020 - January, 2023
Student number:	4207491

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Faculty of Aerospace Engineering · Delft University of Technology

Contents

Nomenclature

List of Abbreviations

2D	Two dimensional
3D	Three dimensional
BW	Black-White
CMOS	Complementary Metal–Oxide–Semiconductor, a type of sensor
CNN	Convolutional Neural Network
CV	Computer Vision
DNN	Deep Neural Network
EASA	EU Aviation Safety Agency
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HSV	Hue, Saturation, Value
IR	Infrared
LED	Light Emiting Diode
MAV	Micro Air Vehicle
MOD	Multiple Object Detection
OD	Object Detection
R-CNN	Regions with CNN features
RGB	Red, Green, Blue
RGB-D	Red, Green, Blue, Depth
SLAM	Simultaneous Location and Mapping
UV	Ultraviolet
YOLO	You Only Look Once

List of Symbols

$\cap$	Intersect
$\cup$	Union
AP	Average Precision
IoU	Intersection over Union
mAP	Mean Average Precision
u	Horizontal velocity
v	Vertical velocity

List of Figures

List of Tables

1 Introduction

With the increasing prevalence of drones, and their lower costs, research has focused more on ways to reliably detect, and position drones within the airspace. With companies such as Amazon investing in drone deliveries[1], and regulatory institutions such as the EASA looking into managing the drones within the air[2], the likelihood for different swarms to fly through each other increases as well. Getting these drones to cooperate is one way to avoid mid-air collisions, but not a catch-all solution.

Within a drone swarm, drones can deduce each others relative states from broadcast positions obtained via GPS/GNSS systems, or other means of positioning. This functions well in places which are not GPS-denied, and when drones are cooperating. As the Gatwick Drone Incident of December 2018 has shown, not all drones in an air-space will cooperate.

For cooperative drone swarms, communication-based relative state estimation carry the extra burden of limiting scalability. It requires drones to either filter out neighbours based on distance, or to compute the relative positions of all drones within the swarm. While this can be done centrally for small swarms, for large swarms the amount of computations grows quadratically.

The article in Part I investigates the inputs used for vision based relative localisation. In Part II, the preliminary work done before the thesis is shown.

[1] Prime Air, launched in 2013

[2] `https://www.easa.europa.eu/sites/default/files/dfu/what_is_u-space.pdf`

Part I

Scientific Article

Deep Vision-based Relative Localisation by Monocular Drones

Student: R.A.A. Mink
Supervisors: Dr. S. Li & Prof. dr. G.C.H.E. de Croon,
Control & Simulation Section, Faculty of Aerospace Engineering
Delft University of Technology, Delft, The Netherlands

***Abstract*—Decentralised drone swarms need real time collision avoidance, thus requiring efficient, real time relative localisation. This paper explores different data inputs for vision based relative localisation. It introduces a novel dataset generated in *Blender*, providing ground truth optic flow and depth. Comparisons to *MPI Sintel*, an industry/research standard optic flow dataset, show it to be a challenging and realistic dataset. Two Deep Neural Network (DNN) architectures (YOLOv3 & U-Net) were trained on this data, comparing optic flow to colour images for relative positioning. The results indicate that using optic flow provides a significant advantage in relative localisation. The flow based YOLOv3 had an mAP of 48%, 9% better than the RGB based YOLOv3, and 23% better than its equivalent U-Net. Its $IoU_{0.5}$ of 63% was also 14% better than the RGB based YOLOv3, and 51% than its equivalent U-Net. As an input, it generalises better than RGB, as test clips with variant drones show. For these variants, the optical flow based networks outperformed the RGB based networks by a factor of 10.**

***Keywords*—Deep Neural Networks, Optic Flow, Relative localisation, Computer Vision**

I. Introduction

While ubiquitous use of autonomous drones and drone swarms in daily life have been the material of science fiction for decades, serious research is paving the way to turn that dream into reality. Applications ranging from drone shows to greenhouse monitoring or search-and-rescue are under active research or use, and all require different types of automation and coordination within drone swarms. Autonomous operation requires the drones to be able to avoid each other. This can be achieved by *relative state estimation*, which is the task of determining the position, and potentially orientation, of another body or object relative to ones self. Drones can make use of this to avoid mid-air collisions, navigate through difficult environments, and other tasks. One avenue for relative state estimation is through optical sensing.

Most drones have a CMOS camera equipped, due to its relative cheapness and availability. This is a very cost effective sensor, typically with high resolutions, but introduces limitations of frame rate and the depth problem inherent in monocular imagery. Different sensor types can improve relative localisation, by either introducing depth data such as with a depth [1] or stereo camera [2], or by not making use of a framerate such as event cameras [3]. However, this research focuses on the CMOS camera as it is a more widely available.

One drawback of communication based relative localisation within drone swarms is the scalability of the swarm. As drone swarms increase in size, the number of relative positions increases quadratically, with each additional drone adding $(n-1)$ new relative positions in the swarm. This imposes a limit on communication based collision avoidance, as larger swarms require increased bandwidth processing power to keep up [4].

One of the limitations of a vision based approach to relative localisation has been the processing power required. With recent advances in chip technology, such as the AI-deck [5] or Nvidia's Jetson Nano[1], implementation of DNNs onboard Micro Air Vehicles (MAVs) have become feasible. This opens the door to real-time, onboard Deep Learning based solutions.

Initial work into autonomous visual relative localisation used fiducial markers. One such example is the use of ArUco codes [6], where printer markers were placed on obstacles to aid in detection. Another example is [7]–[10], a set of papers exploiting different constellations of UV LED markers. In particular, [9] highlights how the markers can be leveraged for relative pose information, aside from localisation. However, this requires markers to be installed, making the approach less generic.

Other studies investigating relative state estimation of drones focus on using a single image, such as [11]–[14]. These methods do not depend on the drone having fiducials installed, but focus their experiments on simpler, uncluttered backgrounds such as empty skies or laboratory conditions. Research such as [11] push this to more complex backgrounds, increasing it's robustness for more difficult environments. However, none of those papers make use of the motion inherent in drones.

A more motion based approach is also looked at, based on the work in [15] which showed improved detections by using motion estimates to refine its appearance-based classification. However, it does not run in real-time, as the method used requires detections three time-steps into the future.

In the other extreme, [16] makes use of event cameras, exploring localisation through only motion, as the sensor used can only detects changes in the scene. However, it only detects drones by the rotation of its propellers. Thus is limited to detect drones only from a top down view. The combination of the sensor and the vantage point currently make this method less generalised.

[1] https://www.nvidia.com/jetson-nano/

Overall, no single paper found by the author presents a motion based, real-time, generic positioning system, needed for autonomous drone operation. Therefore, this research focuses on generic relative localisation with CMOS imagery, through optic flow features. Optic flow is the apparent motion of objects and features in a scene, caused by the relative motion of both the observer and observed.

Since it was previously shown that making use of the motion of drones improves detection in [15], this research focuses on exploiting that motion. It is expected that an optic flow based approach will perform better than appearance based data. Unlike in [15], no information from future time-steps will be required. Additionally, this research will make use of DNNs, as they have been shown to work in real time on MAVs [5].

By using optic flow, the networks should be more general, as they do not depend on appearance cues. Instead, they would recognise the shape of flow fields to make detections. This would allow the network the flexibility to detect drones of different shapes and sizes, without depending on its appearance.

To study this, a novel dataset was generated with ground truth data for Optic Flow, depth, segmentation, and bounding boxes in section II. In section III the method of this research is presented, identifying the neural network architectures used. These networks are then trained on the new dataset, and the performance and behaviour of motion and appearance based detection is explored in section IV. The conclusions of this research are presented in section V, along with recommended future work.

II. Dataset

A dataset was created in Blender for this research. There are several reasons Blender is suitable for this task: Firstly, Blender's rendering engine "Cycles", is capable of generating optic flow data, simulating motion blur, and is highly scriptable. Second, there is a large community of developers which release models under permissive licenses, from which datasets can be constructed. Thirdly, the Blender Foundation also maintains a python package which interfaces with the rendering engine, allowing for automated generation of the dataset. Lastly, Blender has also been used in the past to generate the MPI Sintel Motion dataset [17], which was deemed a representative motion dataset comparable to real-life motion. This combination results in a dataset that can expand easily, in terms of labels, extracted features, and variation.

The environment models were found with CC0 and CC-BY licenses, taken from the Blender Foundation website[2], and combined with a CC-BY licensed drone model designed by BlueMesh Studio[3]. This resulted in four different environments with different levels of occlusion and lighting. No environment was used featuring visual effects such as mist or glare. The drone was flown through different patterns, at varying distances, to generate the dataset, examples shown in Figure 1. To create an especially challenging test set, additional clips deviating from the rest of the dataset were generated featuring an alternate drone colouration, and an additional environment.

Figure 1: Samples from the dataset, featuring one indoor scene and three outdoor scenes with different lighting.

Collected features for the dataset were PNG images, encoding the RGB images, usable as a sequence of frames. Several labels are also generated per frame, namely the Optic Flow, drone relative position vectors, ground truth bounding boxes, pixel-wise depth data, and ground truth segmentation masks.

In Figure 2, some sample flow fields of the dataset are shown. The drones are visible against the rest of the scene, due to the difference in motion. However, also visible are some of the more challenging parts, where the drone is difficult to detect in the context of larger magnitude motions surrounding it.

To more clearly represent the dataset, several of its statistics are highlighted. In fig. 3 the variation of luminance within each sequence is shown, also showing the averages for only the drone pixels. Visible is that within some sequences very little changes, while within others a large change is visible. Additionally, the drones show an almost identical variation when compared to the background. It shows how there is a lot of variety, both in the width of the variation, as well as the entire clip averages. This is one of the strengths of the dataset, as it contains both indoor, outdoor, and mixed footage, at multiple times of day.

The dataset generated is compared to another similar dataset, called MPI Sintel [17]. That is an optic flow dataset compiled from sequences and scenes of the Blender Foundation's

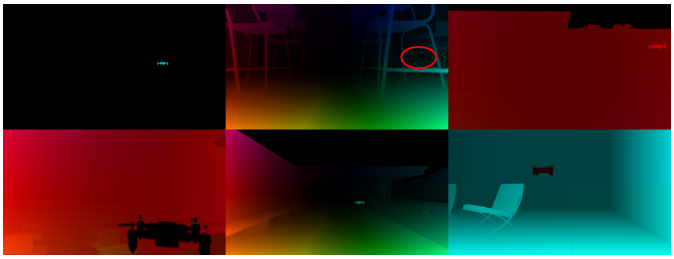


Figure 2: Six sample Optic FLow fields from the dataset, showing different movement rates and scales of the drones. The direction of flow is indicated by hue, the magnitude by saturation. The drone is circled in red when difficult to spot.

[2]https://www.blender.org/download/demo-files/
[3]https://www.sketchfab.com/VapTor

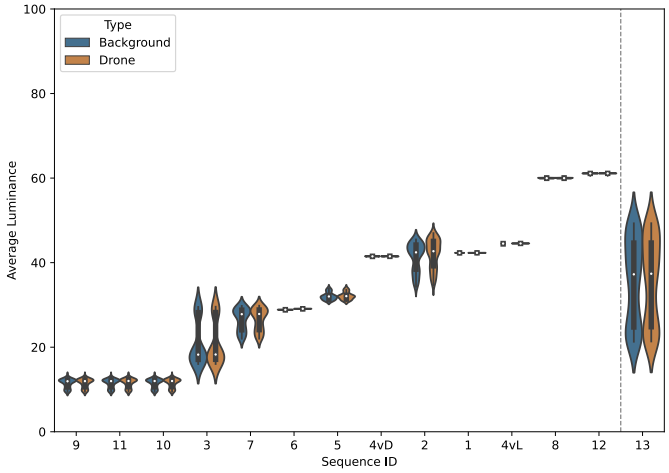


Figure 3: Average luminance of each frame on a scale of 0-100, collected per sequence for both the drone pixels and entire image. It it sorted on ascending luminosity, where the dotted line separates the general dataset from a clip used only in section IV-C

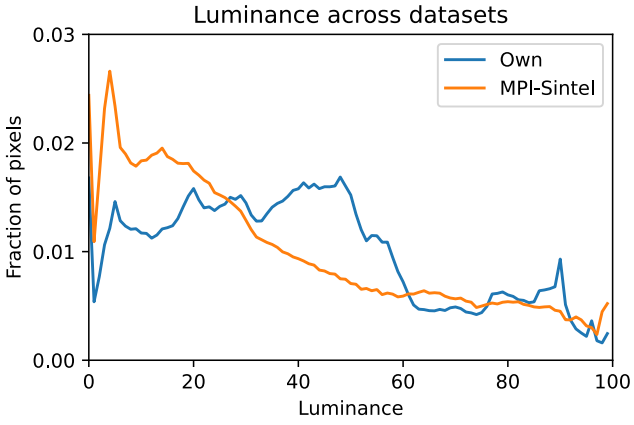


Figure 4: Histogram of the dataset, showing light intensity per pixel in the dataset, on a scale of 0-100. A comparison to the MPI Sintel dataset [17] shows that the new dataset has a similar amount of darker frames, but more low level light.

Sintel Open Movie Project. It features a great deal of motion, and is validated against data mined from other movies. The purpose of comparing to MPI Sintel is to compare the optic flow data. However, MPI Sintel is not a drone based dataset, and features a great deal of optic flow that is representative of movement near to the camera, which drones tend not to do. This last point is a factor to consider in the comparisons.

In fig. 4 a histogram of the light intensity of the dataset is shown. Up to near the 50 L mark, the luminosity of the drone dataset is rather flat. After, this settles at a lower plateau, indicating that the dataset is on the whole a very dark dataset. This is probably due to the large quantity of indoor scenes or outdoor scenes with little illumination. With less light available, drones are harder to distinguish, making this a challenging dataset.

Compared to MPI Sintel [17] dataset our own dataset has less darker scenes, but more scenes of median light. Our dataset shares the challenging nature of the MPI dataset of having less brightly lit scenes, likely due to being mainly indoor, or in dark or dusk environments.

In fig. 5 the similarity between the two datasets is also apparent. The new dataset has a similar distribution of motion along the horizontal axis (u), but a marked difference when comparing the vertical motion v. This is likely due to the higher amount of close-up shots in the MPI Sintel dataset. Even with that difference, the absolute velocities in the entire dataset (bottom left) is still comparable. This supports the idea that the dataset generated is a challenging and realistic dataset, as it contains a wide range of motions.

The ground truth depth maps in this dataset were generated using Blenders Cycles engine, which outputs the distance to the camera. The dataset was created to ensure that these distances were expressed in meters. Due to the dimensions of the available scenes, the furthest a drone was placed from the camera was 25m. At the same time, due to the size of the

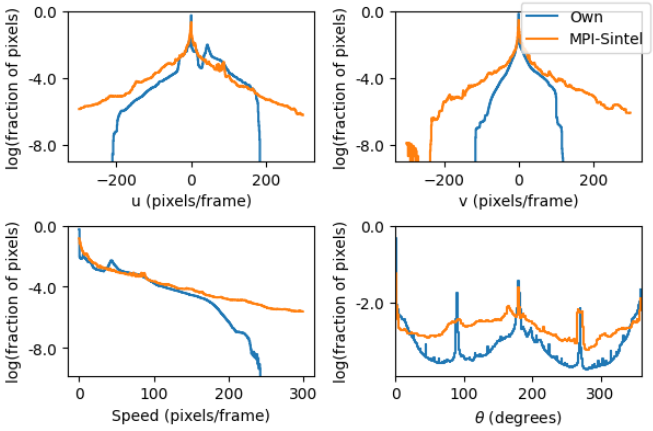


Figure 5: Ground truth optic flow statistics of the entire dataset. The top two figures show histograms of the horizontal (u) and vertical (v) flow, the bottom two the magnitude and direction. An angle of 0° is horizontally to the right. The orange line is a comparable optic flow dataset called MPI-Sintel [17], while the blue line is the dataset described in this paper.

drone, the closest it was placed was 80cm, to simulate close formation flying. This created a variety of sequences where the drone generally occupies a very small fraction of the image, with a few sequences with drones blocking out large swaths of it. Some examples are shown in fig. 6.

The ground truth segmentation masks were also output from the Cycles engine. This engine determined which object was under the pixel that it was generating, allowing it to create a bit-mask per drone. This allows not only for sequences of single drones to have bit-masks, but also to distinguish between nearby or overlapping drones. These bit-masks were used later to generate the bounding boxes through a python script.

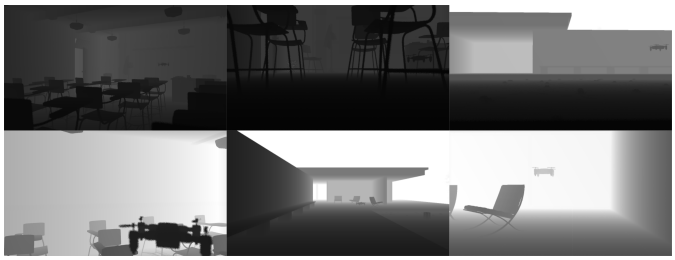


Figure 6: Six sample depth maps from the dataset. The lightness of a pixel is proportional to the distance to the camera, with white pixels representing a distance of 30m or farther. These samples correspond to the samples in fig. 2.

III. Method

Two neural network architectures were trained using Tensorflow 2.4.1 and the dataset presented above. The first network trained is the U-Net architecture, first presented in [18], and reproduced in fig. 7. The network was trained with inputs scaled to a resolution of 512×512 during preprocessing. This was done without cropping, preserving the aspect ratio and ensuring the shape of the drone did not change. The dataset itself was split into a training and test set, with 80% of the sequences being used for training. It was trained against two channel flow data, representing the horizontal and vertical pixel flows in units of pixels/s. This network was trained for a total of 200 epochs, using batch normalisation.

The second network architecture used was YOLOv3, presented in [19]. A diagram of this network is shown in fig. 8. The network was trained using inputs scaled to a resolution of 512×512, using the same preprocessing steps as for the U-Net layer, in batches of 16. Two versions of the network were trained, one drawing bounding boxes from RGB image data, and the other from flow data. These were evaluated against ground-truth bounding boxes. The networks were trained for a total of 120 epochs, using batch normalisation.

The data was preprocessed in two ways, depending on whether it was flow data or image data that was being processed.

For the image data, the colour space used was RGB, and all colour images were converted to that. This format was chosen as a common colour format, which matches the output of CMOS cameras. The HSV colour format was considered, but not used due to the extra processing step required for use onboard drones. The original images, sized at either 1920×1080 or 1280×720, were resized to 512×512 pixels, again preserving aspect ratio through padding. The same resizing ratios were applied to the ground truth boxes, to ensure they aligned with the image. This resulted in an input tensor of shape (512,512,3) that was given to the networks.

The flow data preprocessing requires a few more steps. Each frame contains four channels of data; $u_{n-1,n}, v_{n-1,n}$, the flow w.r.t. the previous frame; and $u_{n,n+1}, v_{n,n+1}$, flow w.r.t. the next frame. For this research, only the first two channels were used, as the second two are 'future' information not known to a real-time use-case. Following the same steps as the image data above, the flow data was resized into a (512,512,2) tensor that could be given as an input to the network.

No further data augmentation methods were used, such as cropping, mirroring or hue shifting. Due to flow data not containing hues, this was not applicable. Cropping was not used in order to avoid cutting out drones from near the edges, nor to change the shape of the bounding boxes.

IV. Results and Discussion

A. Qualitative results

Figure 9 shows the sample predictions of the networks described in section III side-by-side. The three networks were evaluated against the same testset. From the inputs to the networks, five were sampled to illustrate different behaviours noted. The first two columns display the inputs to the network, with the subsequent columns showing the predictions of each network for that input. From this figure, several things can be seen.

In the first two rows, a) & b), when drones are the only thing moving in view, all networks detect drones correctly, with only RGB YOLOv3 having a false negative. This supports the intuition that moving sensors are more difficult for drones, which is a disadvantage for onboard detection.

An interesting thing to see in row b) is the presence of a halo around the two drones detected by the UNet network trained on flow data. This is possibly concealed under the larger-than-necessary bounding box detections of both YOLOv3 networks.

In row c), it can be seen that occlusions hinder the flow based detection, as both the network types perform worse than the RGB network. This could be due to the shape of the drone's flow being broken, causing the network to have difficulty recognising it. In both flow networks this is visible in the mislabelling of the background flow.

Row d) shows a clear example of a detection failure of U-Net. While both YOLOv3 networks are able to detect the drone, the U-Net column does not show any detection for the given input, aside from the background. Looking at the input flow, it is clear that the drone does not experience any occlusions, nor has any similar flow nearby, which should make it detectable for U-Net. Perhaps the magnitude of the flow is part of the problem, as the similarly clear flow field of row e) does not cause as much difficulty.

Also visible in row e) is that the RGB YOLOv3 network has a false positive on the right side of the image. All networks perform well on row e) with this exception, reinforcing the idea that flow inputs works better as an input.

B. Quantitative results

To evaluate the networks, four different metrics were used. These are the IoU scores, two implementations of the Average Precision, and the execution time.

IoU is a good metric to show how well two bounding boxes overlap, as a value of 1 indicates a perfect match, and 0 no overlap between boxes. It encodes the relative sizes of the bounding boxes, as well as the overlap. It is defined as $\cap/\cup$.

In contrast, Average Precision reflects how trustworthy predictions are. It evaluates how well a prediction correctly finds a

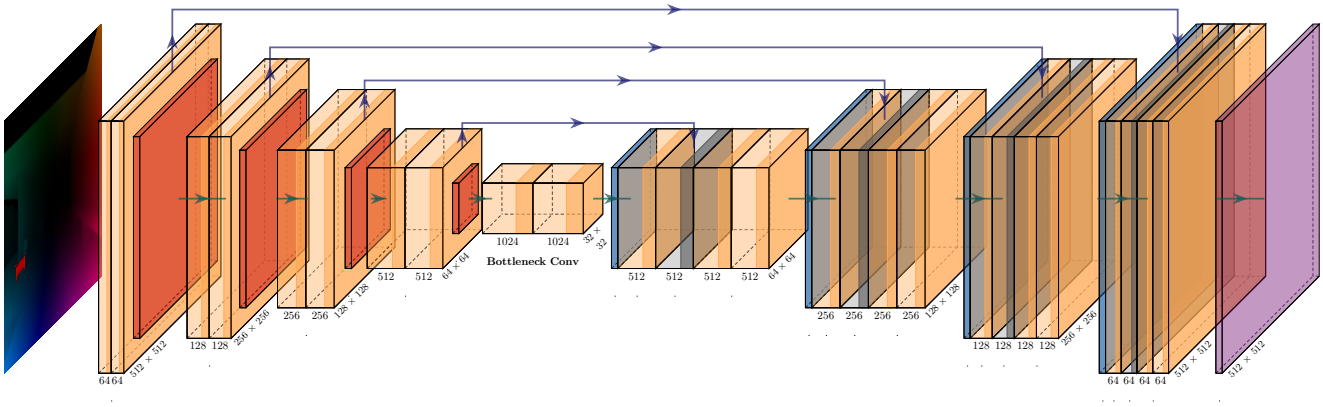


Figure 7: U-net network diagram. One frame of flow data is shown as example input, on the left. During experiments, 2 channel flow data and 3 channel colour data were separately trained. The network starts with blocks of orange convolutions with a pooling step in darker red. Two convolutions at the smallest image resolution of 32×32 pixels preceed the the upsampling arm, which returns to a 512×512 image detection mask.

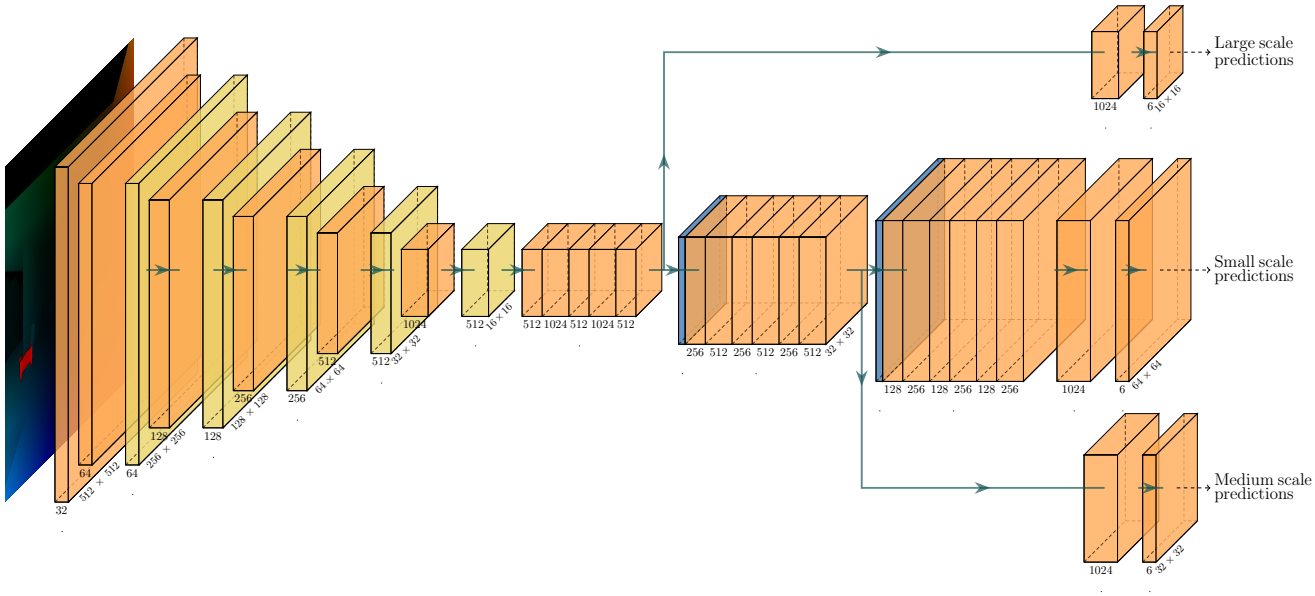


Figure 8: YOLOv3 network diagram used in this paper. A sample flow input is shown on the left. The network contains a mix of convulational and residual blocks, outputting predictions at three scales. These are then turned into corresponding bounding box predictions, shown as dotted arrows pointing right. In the figure, the yellow boxes indicate residual blocks, the orange convolutional layers, and blue upsampling layers.

true label. This is done via the IoU score of a prediction, where a threshold is used to weed out false positives. Two ways of weeding out false predictions were used, one allowing for multiple detections of the same ground truth. The first, $AP_{0.5}$, scores only the best bounding box for a given ground truth that meets the threshold as a true positive, and any subsequent as a false positive. The second, $AP^2_{0.5}$ considers any bounding box that meets the threshold a true positive, and any that does not a false positive, irrespective of whether the ground truth has already been detected. Both $AP_{0.5}$ and $AP^2_{0.5}$ use a threshold of 0.5.

The mAP expands this to also look at how well the network collects all targets. It calculates the recall, defined as the number of true positives over the total number of targets, and the precision, for all thresholds ranging from 0 to 1 to construct a Precision-Recall curve. The mean Average Precision is the area under this curve.

To calculate the average execution time, all forward passes of the network, excluding the data pre- and post-processing steps, were timed across the testset. This was done to isolate the execution time of the networks themselves, which average times are included in the table.

These evaluation metrics for the networks are given in table I.

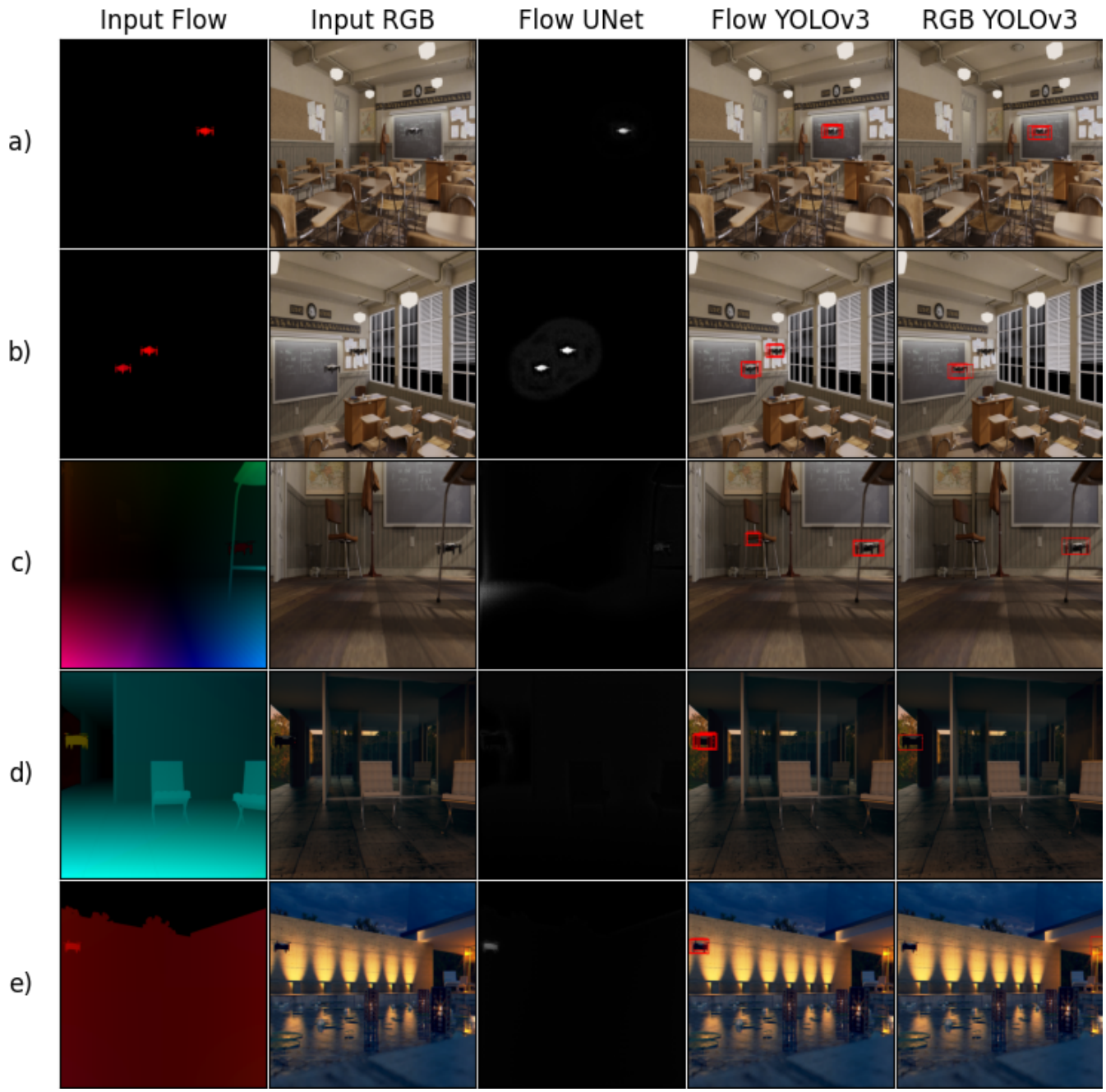


Figure 9: Different inputs for the three different networks trained. All inputs and outputs were resized to a square aspect ratio to fit into the grid. First two columns display the possible inputs, while the others identify which input was given to which network architecture. Flow YOLOv3 performs best, only having a false positive in row c). In comparison, RGB YOLOv3 fails to detect drones in rows b) and e), and has a false positive in row e). Flow UNet, in contrast, only confidently detects the drones in the first two and last rows.

	U-Net	YOLOv3	
Input	Flow	Flow	RGB
$IoU_{0.5}$ (↑)	0.1143	**0.6280**	0.4875
$AP_{0.5}$ (↑)	0.1808	**0.2233**	0.1520
$AP^2_{0.5}$ (↑)	0.1808	**0.7902**	0.4597
mAP (↑)	0.2536	**0.4847**	0.3862
Execution time (s) (↓)	0.1241	0.0629	**0.0598**

Table I: Run-time statistics of the different networks over the test set. Arrows indicate which value is better, and the best value of each row is indicated in bold.

Comparing the three remaining networks it is evident that YOLOv3 networks score better on the $IoU_{0.5}$ metric than the U-Net network.

Looking at the metrics of $AP_{0.5}$ and $AP^2_{0.5}$, one clear difference is the effect of penalising multiple detections. The YOLOv3 networks both detect individual drones multiple times, which could mask multiple overlapping drones as a single drone. The difference between the $AP_{0.5}$ and $AP^2_{0.5}$ is one way to show how the YOLOv3 networks cluster predictions around drones. This is something that can be corrected in postprocessing steps. Another noticable change between the networks is the better performance of the Optic Flow data. Looking at the difference between YOLOv3-Flow and -RGB, there is almost a 0.33 difference between the two. This supports the hypothesis that Optic Flow data works better for detection of drones.

The mAP scores also support this, as the YOLOv3-Flow network is the best performing one. Combined with observations during training, this indicates that the shape of the drones motion is what the Flow YOLOv3 network learned to detect, which is more robust than the colour based detections. The execution times of the networks follow the expected behaviour of the YOLOv3 networks completing the forward pass much faster. That YOLOv3 outperforms U-Net was to be expected, as within the literature it was promoted as the best network for object detection tasks [19]. As there was no study found detailing the effects of input type, it is interesting to see the superior performance between the Flow and RGB networks. Similar behaviour can be seen in [15], where a motion based input outperforms a normal static input.

C. Response to variant drone design

One hypothesis of this study was that Optic Flow based networks would more reliably detect unknown drones that do not fit the shape or colouration expected than RGB based networks. This continues in the vein of [15], as it demands that the network leverages the motion instead of appearance. To test this, one sequence was developed using a dark blue and green drone.

In fig. 10, a similar figure to fig. 9 is shown. From these samples, flow based YOLOv3 is the best performing network, showing correct detections in rows a) and c). Comparatively, RGB based YOLOv3 only accurately identifies the drone in row b), with a bounding box far larger than necessary.

This difference in amount of detections is not just in these samples; across the 160 frames of the sequence, the YOLOv3 networks makes bounding box predictions for 27 of the RGB inputs, and for 129 of the optic flow inputs. Combined with the fact that no samples were found where the OF-based UNet network and OF-based YOLOv3 network both labelled the drone, the author concludes that this sequence was a very difficult one for all networks.

In rows b)-c) it is also visible that the UNet network labels a large portion of the wall as 'drone'. This suggests that the UNet network is more concerned with the magnitude of motion.

The IoUs and mAPs for this test are presented in table II, which surprisingly show a significant hit in performance of the Optic Flow based networks. As flow is not dependent on the colour of a pixel but its displacement, it was expected that the Optic Flow based networks would be unaffected by changing the colour of a drone. This suggests the test sequence was far more challenging than expected. However, even faced with an unusually challenging test sequence, the RGB network still performed far worse than the Flow networks, as expected. This underscores one of the benefits of Optic Flow, as it is more generalisable than RGB.

	U-Net	YOLOv3	
Input	Flow	Flow	RGB
$IoU_{0.5}$	0.0460 (40%)	**0.1484** (24%)	0.0260 (5%)
mAP	0.0899 (35%)	**0.1457** (30%)	0.0135 (3%)

Table II: Performance of the dataset on a clip featuring a different drone colouration. Also given is a comparison to the statistics in table I, where the value in brackets indicates the ratio between the two tables. A higher value is desirable for both metrics, and the best value is marked in bold.

Another hypothesis is that the flow based U-Net network identifies the drones by finding the maximum flow in the field of view. Snapshots during the training process corroborates this theory, as initially obstacles (such as the chair in fig. 9, row b) that move faster than the drone are also identified as drones. This behaviour was not present in the YOLOv3 networks, suggesting that those networks did not depend on the absolute flow velocity.

Further research can be built upon the above, with a few questions remaining open. The dataset used is entirely synthetic, so a real-life test set can be generated using onboard cameras and position trackers such as OptiTrack, as a challenging secondary dataset. This would evaluate the applicability of flow based networks in real-world scenarios, bridging the Simulation-to-Reality divide.

A second question would be to test the difference between two successive RGB frames. This would not be as information rich as Optic Flow data, but could prove a less computationally intense source of information.

V. Conclusions

This paper has presented a novel drone detection dataset with multiple environments, motion regimes, and light conditions, containing ground truth values for Optic Flow, Depth, segmentation, and bounding boxes. A comparison to MPI Sintel has shown it to be a comparable and challenging

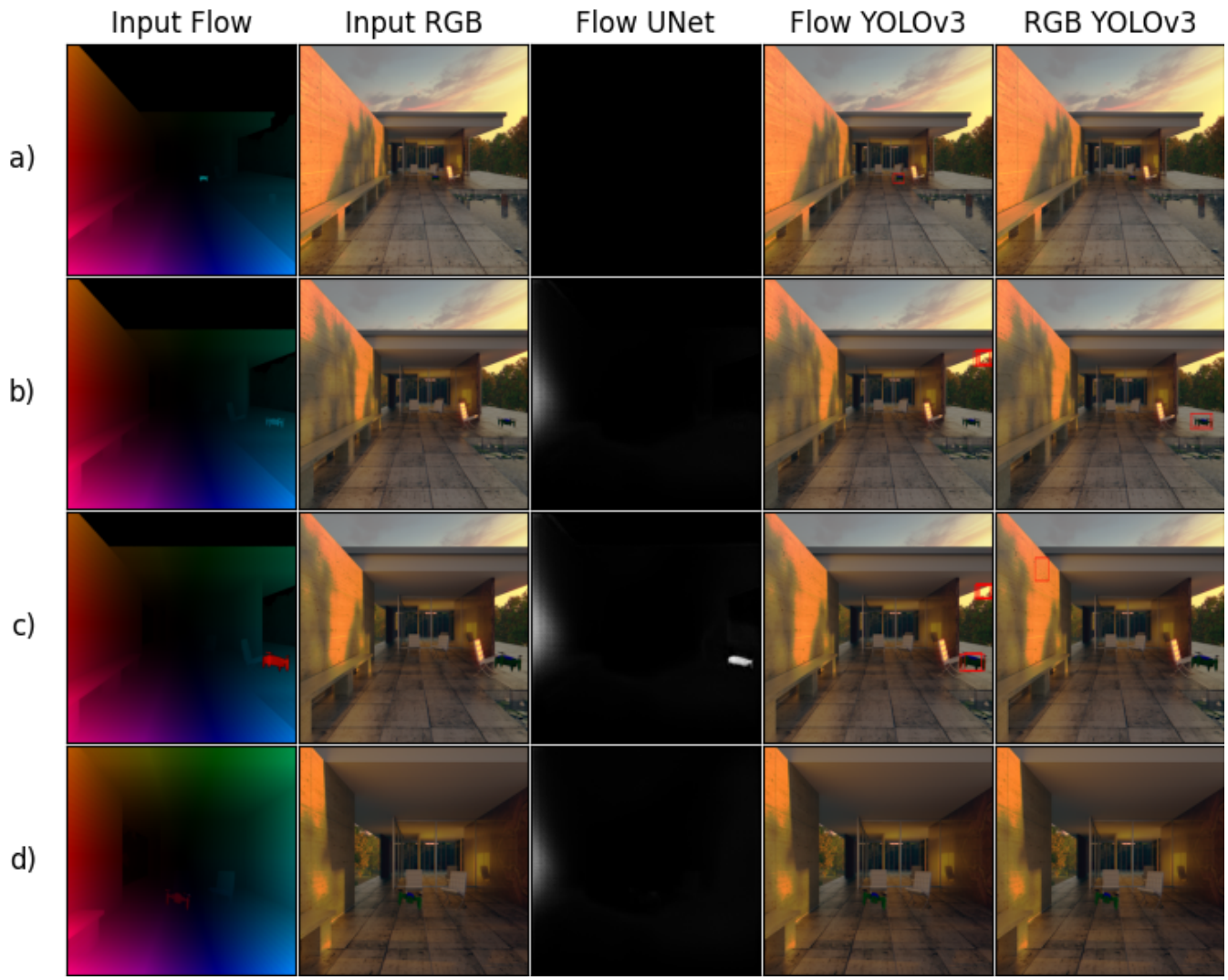


Figure 10: A similar plot to fig. 9, but for the sequence with the variant drone design.

dataset. We then used this dataset to train two competing deep network architectures, YOLOv3 and U-Net, to compare the responsiveness to different input types, optic flow and normal images.

Visible in table I, the YOLOv3 networks both outperform the U-Net network, with the optic flow based YOLOv3 network having an mAP of 48%. This is 9% higher than the RGB YOLOv3, and 23% better than the Flow U-Net. When comparing the performance against the variant drone in section IV-C, the benefit of using optic flow is clear. For both flow networks, the score decreased less than for the RGB network, which only retained 3-5% of it's original metric. Due to this, we concluded that making use of optic flow as input data for drone localisation problems allowed for better predictions, and was more generalisable. This is a promising path for onboard relative localisation in drone swarms.

Several avenues of further work remain. In the first, the dataset could be improved upon to bridge the gap between simulation and reality. The flight paths of the drones within the dataset were generated on the computer, and follow smooth curves. By using recorded flight paths, such as through motion-capture, the motion of drones can be made more realistic.

The second avenue would be to implement the work from this research on board real drones. This would show how well suited for on board usage it would be.

A third potential avenue would be to expand the networks to make use of the depth data in the dataset. Combining it with the segmentation masks or bounding boxes would give a "true 3D position" label to use during training. While especially challenging for the CMOS cameras, the optic flow networks may be able to use the encoded motion to create a 3D relative position estimate.

VI. Acknowledgements

I'm deeply grateful for the support of my supervisors, Dr. G.C.H.E. de Croon and Dr. S. Li, my family, and friends during the writing and research of this thesis project. The patience and understanding as I worked through this thesis project during and after COVID-19 lockdowns were invaluable to me.

REFERENCES

[1] R. Takaoka and N. Hashimoto, "Depth map super-resolution for cost-effective rgb-d camera," in *2015 International Conference on Cyberworlds (CW)*, 2015, pp. 133–136.

[2] Y. Mito, M. Morimoto, and K. Fujii, "An object detection and extraction method using stereo camera," in *2006 World Automation Congress*, 2006, pp. 1–6.

[3] D. Falanga, K. Kleber, and D. Scaramuzza, "Dynamic obstacle avoidance for quadrotors with event cameras," *Science Robotics*, vol. 5, no. 40, 2020. [Online]. Available: https://robotics.sciencemag.org/content/5/40/eaaz9712

[4] M. Coppola, K. N. McGuire, C. De Wagter, and G. C. H. E. de Croon, "A survey on swarming with micro air vehicles: Fundamental challenges and constraints," *Frontiers in Robotics and AI*, vol. 7, p. 18, 2020. [Online]. Available: https://www.frontiersin.org/article/10.3389/frobt.2020.00018

[5] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "Ultra low power deep-learning-powered autonomous nano drones," *CoRR*, vol. abs/1805.01831, 2018. [Online]. Available: http://arxiv.org/abs/1805.01831

[6] J. Pestana, J. L. Sanchez-Lopez, P. de la Puente, A. Carrio, and P. Campoy, "A vision-based quadrotor swarm for the participation in the 2013 international micro air vehicle competition," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2014, pp. 617–622.

[7] V. Walter, N. Staub, A. Franchi, and M. Saska, "Uvdar system for visual relative localization with application to leader–follower formations of multirotor uavs," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2637–2644, 2019.

[8] M. Vrba, D. Heřt, and M. Saska, "Onboard marker-less detection and localization of non-cooperating drones for their safe interception by an autonomous aerial system," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3402–3409, Oct 2019.

[9] V. Matouš, "Relative localization of helicopters from an onboard camera image using neural networks," Master's thesis, CZECH TECHNICAL UNIVERSITY IN PRAGUE, 2018.

[10] T. Baca, P. Stepan, and M. Saska, "Autonomous landing on a moving car with unmanned aerial vehicle," in *2017 European Conference on Mobile Robots (ECMR)*, 2017, pp. 1–6.

[11] S. Li, C. D. Wagter, and G. C. H. E. de Croon, "Self-supervised monocular multi-robot relative localization with efficient deep neural networks," *CoRR*, vol. abs/2105.12797, 2021. [Online]. Available: https://arxiv.org/abs/2105.12797

[12] M. Cognetti, P. Stegagno, A. Franchi, G. Oriolo, and H. H. Bülthoff, "3-d mutual localization with anonymous bearing measurements," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 791–798.

[13] M. W. Ashraf, W. Sultani, and M. Shah, "Dogfight: Detecting drones from drones videos," 2021.

[14] J. Li, D. H. Ye, T. Chung, M. Kolsch, J. Wachs, and C. Bouman, "Multi-target detection and tracking from a single camera in unmanned aerial vehicles (uavs)," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 4992–4997.

[15] A. Rozantsev, V. Lepetit, and P. Fua, "Flying objects detection from a single moving camera," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 4128–4136.

[16] N. J. Sanket, C. D. Singh, C. M. Parameshwara, C. Fermüller, G. C. H. E. de Croon, and Y. Aloimonos, "Evpropnet: Detecting drones by finding propellers for mid-air landing and following," *CoRR*, vol. abs/2106.15045, 2021. [Online]. Available: https://arxiv.org/abs/2106.15045

[17] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *European Conf. on Computer Vision (ECCV)*, ser. Part IV, LNCS 7577, A. Fitzgibbon et al. (Eds.), Ed. Springer-Verlag, Oct. 2012, pp. 611–625.

[18] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

[19] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

Part II

Preliminary Analysis

*This part has been assessed for the course AE4020 Literature Study.

3

Literature Review

3.1. Introduction

With the increasing prevalence of drones, and their lower costs, research has focused more on ways to reliably detect, and position drones within the airspace. With companies such as Amazon investing in drone deliveries[1], and regulatory institutions such as the EASA looking into managing the drones within the air[2], the likelihood for different swarms to fly through each other increases as well. Getting these drones to cooperate is one way to avoid mid-air collisions, but not a catch-all solution.

Within a drone swarm, drones can deduce each others relative states from broadcast positions obtained via GPS/GNSS systems, or other means of positioning. This functions well in places which are not GPS-denied, and when drones are cooperating. As the Gatwick Drone Incident of December 2018 has shown, not all drones in an air-space will cooperate.

For cooperative drone swarms, communication-based relative state estimation carry the extra burden of limiting scalability. It requires drones to either filter out neighbours based on distance, or to compute the relative positions of all drones within the swarm. While this can be done centrally for small swarms, for large swarms the amount of computations grows quadratically.[3]

As most modern drones are equipped with cameras, those sensors can be leveraged for relative state estimation. In this literature study, the following question is used as a guide:

> What are current techniques within the broader Computer Vision community that can be applied on board drones to generate real-time relative state estimates of other drones?

To answer this question, is is broken up in the following sub-questions:

1. What techniques within Computer Vision are applicable to relative state estimation?
2. What is the state of of the art within drone relative state estimation?

Section 3.2 starts tackling these questions by delving into the details of how drones could be equipped to see, as well as identifying some of the more competitive algorithms, some of which see little application in drone theory. In continuation, Section 3.3 starts by discusses the state of the art within drone relative state estimation. It highlights the existing solutions and discusses the limitations thereof. Finally, Section 3.4 summarises the preceding discussions, and identifies directions of research.

[1] Prime Air, launched in 2013

[2] `https://www.easa.europa.eu/sites/default/files/dfu/what_is_u-space.pdf`

[3] In a swarm of n drones, each new drone added would introduce n new relative positions that could be calculated. This results in a total of $n(n-1)/2$ relative positions.

3.2. Computer Vision

Computer vision, hereafter referred to as CV, is a complex field which sees applications far beyond the scope of this research. This chapter identifies the parts of the CV community that are directly related to Micro Air Vehicles, hereafter referred to as MAVs, and the adjacencies that could expand CV within MAV research. The chapter also identifies some of the challenges inherent in computer vision, as well as some of the mitigation strategies.

3.2.1. Visual Sensors

There are several types of cameras available to drone designers, the simplest which is an ordinary camera. Each has it's own advantages and disadvantages, while many share the problem of depth, as described in Subsection 3.2.2.

- BW camera. A Converts the visual spectrum as a Black-White image.
- RGB/HSV camera. Known to most smartphone users, these take colour images.
- IR/UV camera. Detect light from either the Infrared or Ultraviolet spectrum.
- RGB-D camera [1], also known as depth camera. Combines the image of an RGB sensor with a per pixel depth map.
- Stereo camera, also known as two cameras. Analysis of the difference between images leads to depth information.
- Event-camera, also known as a Direct Vision Sensor [2]. Each pixel updates individually and asynchronously, leading to faster detection rates.

Within drones, the most common sensors are the first two; these are easily available, generally cheap, and typically light [3]. Especially for MAVs, the weight can be important due to stringent mass budgets. Combined with the smartphone industry pushing for cheaper, lighter and higher quality colour sensors, this makes them the go-to choice for many drone designers. In applications that require data outside the visual spectrum, ultraviolet or infrared sensors can be used. However, these sensors introduce limitations for tasks requiring 3D positioning, as they only return 2D images, offloading the generation of depth information to other components, either hardware or software.

Some cameras do generate depth information; those are stereo cameras and RGB-D cameras. They do so by combining different sensors. The stereo camera combines two normal colour cameras, spaced at a known distance. The two images, one from each camera, combined with the known distance between cameras, allow for a good depth estimates to be made. In contrast, RGB-D cameras project an IR pattern into the field of view, and convert the detected pattern to a depth map. Both of these ways alleviate localisation problems, as they introduce depth information, albeit with some error margin.

Finally, event cameras are a relative newcomer which mimic biological eyes by treating each pixel as a photoreceptor. When pixels detect a change in brightness, they send a signal out asynchronously, which results in a very low latency. As a result event cameras do not return frames, but rather these must be compiled from the pixel data. Due to this, motions that in conventional cameras are hard to detect due to the frame rate can be detected quickly.

Essentially, this results into three clusters of available sensors: CMOS cameras, returning BW, colour, IR, or UV data; Depth cameras, containing Stereo and RGB-D cameras which additionally return depth image per pixel; and Event cameras, which output pixel data asynchronously, unbound by frame rates.

3.2.2. The depth problem

The problem of depth is inherent in monocular sensors of any kind, due to the way pixels are defined. Each pixel looks at a sliver of the world projecting outwards, as shown in Fig. 3.1. All the sensors in Subsection 3.2.1, with the exception of the stereo and RGB-D cameras, reduce the information space from (X, Y, Z) to (u, v). This conversion is not reversible, as converting (u, v) to three dimensions resolves to the red line shown in Fig. 3.1. Stereo cameras and RGB-D sensors, on the hand, have two different values for P and (u, v), due to making use of two discrete sensors.

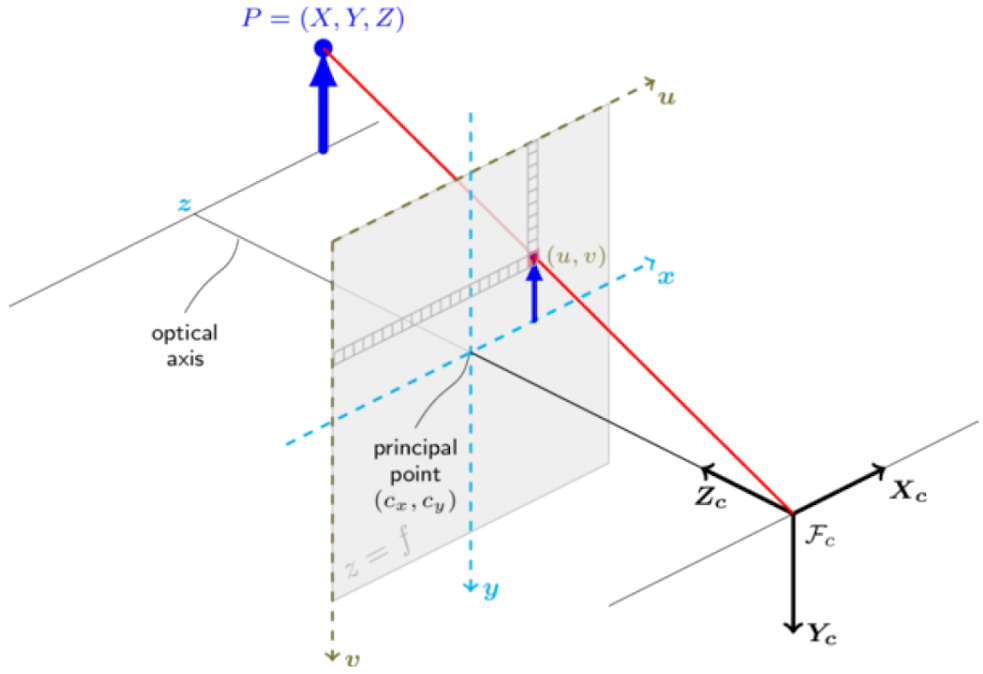


Figure 3.1: Simple model of a pinhole camera, omitting distortion effects from lenses. A pixel at position (x, y) is used to represent an object at point P at position (X, Y, Z). While the conversion of P to (x, y) is trivial, the inverse requires knowledge of z, which cannot be determined from the information of a single pixel. Figure taken from [4].

When focusing on monocular cameras, there are other ways to generate depth information from the images. For cases were the size of an observed object is known [5], it's depth can be deduced from the observed size. This works by identifying multiple points P in the image and combining their relative positions to generate a depth. By making many observations, a map can eventually be built, is typically done in SLAM[4] methods. Alternatively, machine learning networks are leveraged to estimate the depth maps, such as via [6].

Generating, estimating, or measuring depth information is a vital for tasks which require 3-D information, as without it only a bearing can be determined. For some use cases, a bearing will be enough, but for tasks such as SLAM and collision avoidance, the distance to an object is desirable. The depth cameras directly create per pixel depth maps, and generally make this problem a simple one.

However, by keeping the focus on MAVs, depth cameras and event cameras both are constrained by the mass budget, and are not guaranteed to be used. Due to this, the rest of the report will focus on monocular cases, where acquiring depth information is a non-trivial part of the problem.

3.2.3. Features of interest: What computers see

When a computer processes an image, it must find features and identify regions of interest. Provided below is a short list of features that can be extracted from a single image:

- Corners, or interest Points, are specific 2-D points within an image, for example a corner or a bright spot on a dark background.
- Edges can be found by looking at the gradient between different regions, as a sharp contrast. An example would be a silhouette.
- Lines are either composites of edge and corner detection, or detected directly as ridges. They are used to detect elongated objects, such as roads in satellite images.
- Blobs are larger regions that would be missed by corner detectors, due to the absence of sharp gradients. An example would be a large field, or a wall.

[4] Simultaneous Location and Mapping

Typically, these features are found via convolutions. By using specific convolution filters, the image returns a feature map, indicating which pixels contain which features. These low level features can then be combined to generate more complex objects, as in [7]. By combining certain features, complex objects can be identified, and by embedding such features in targets [8, 5], tracking tasks can be simplified.

In addition to these features, when using multiple images extra features can be found. These images could be linked via time, as in videos, or in space, as different angles to the same scene. Additional features that can be derived from the differences between images are as follows:

- The displacement in detected features. Motion can be derived from the changes in locations of edges, corners, and other features.
- Occlusions: Parts of objects are visible in only a subset of the images. Differences in depths can be found with occlusions, object boundaries can be determined from them.
- Optic flow fields. These are vector fields describing motion, and can reveal information about the motion of the camera, as well as objects within the scene. Within optic flow fields, several modes can be used to find different features of the camera's ego-motion. When areas in the image diverge in behaviour from these modes, then other moving objects in the scene can be identified.
  - Foci of expansion and contraction. Due to a cameras motion towards a point, objects seem to move away from the focus of expansion, and towards the focus of contraction.
  - Centre of rotation. The axis of rotation of the camera can be discovered from the points around which everything in the image rotates.

These features can be found via different methods; Either the difference in the outputs to feature detectors per images can be compared, or a detector can be applied to the differential image. Different features in input images, such as occlusions, corners and edges can be linked together to decode displacements. To keep computational efficiency, traditional methods typically do this as sparse methods, only linking features of interest.

3.2.4. Machine Learning

With the ascendancy of computational power, machine learning has also found it's way into Computer Vision tasks. These networks are capable of complex distinctions, such as between breeds of dogs, based on the quality of the data that it is trained with. To do this, networks create their own criteria based on the data they are exposed to during training, such as convolution filters or feature weighting. This also allows them to find more complex features, such as 'eyes' or 'faces'[5], within an image. Due to this, it can be difficult to determine exactly what occurs within a machine learning network, as some of these networks operate as a black box system, and only reveal their workings after careful analysis.

These machine learning networks are typically computationally intensive, precluding it's use in MAV platforms. Historically, this is due to it's applications, all allowing the use of a large quantity of processing power (desktop PCs, automobiles, servers, etc). However, recent innovations have brought AI functionality to even nanodrones [9]. Due to this, the more lightweight architectures from the CV community become accessible to the MAV community.

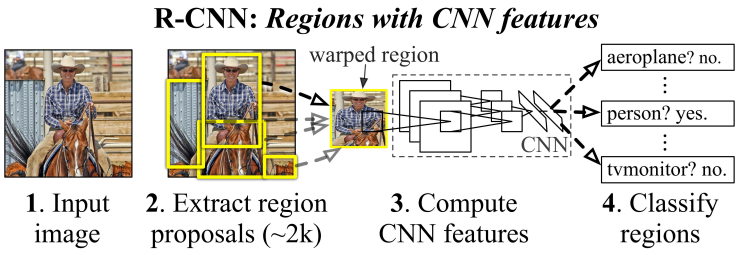
Within the taxonomy of machine learning solutions there are several possible ways that machines can learn to see. In the first place, machines can learn to see, and detect objects, from the information contained within in an individual image. This will be treated in more detail in Subsection 3.2.4 This treats every frame in a video feed as separate, and detection in multiple frames can then be linked across frames. This is called Appearance based detection, and tends to make use of Convolutional Neural Networks (hereafter referred to as CNNs), a relatively fast and powerful network type.

In contrast, Motion Analysis makes use of the differences between sequential inputs, in a time sensitive manner. This requires more processing power and memory, as it must keep the previous input. Depending on the architecture, it can keep the processed results in memory, or simply process the previous input in a second pipeline, before using those results with the current input. A more detailed overview is given in Subsection 3.2.4.

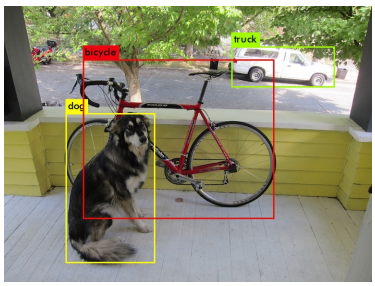
[5] Google's DeepDream vividly shows how a Deep Neural Network can find eyes in unexpected locations.

Appearance based detection
Computer vision has several subsets when it comes to finding objects within scenes, these are typically called Object Detection tasks. Variants hereof exist, aimed at tackling different subproblems, such as Multiple Object Detection (MOD), which distinguishes between objects of the same type; and Category-specific Object Detection, which only detects a objects of relevant categories.

Networks are trained with use of a few detection architectures, such as the YOLO [10], R-CNN [11, 12], or SSD [13] detectors. These are based on CNNs, making use of convolutional layers to create bounding boxes to localise objects within the scene. An example of this is Fig. 3.2a, which gives a high-level illustration of how the R-CNN network architecture detects and classifies objects in the scene. While this example specifically shows the R-CNN network, it is generalisable example. Some outputs are delivered as bounding boxes, and others as pixel maps.



(a) The R-CNN pipeline. Taken from [11]. First regions of interest are identified, and based on the CNN features found within them they are classified or discarded.



(b) Example output with detections. Taken from [14]

Figure 3.2: Examples of appearance based detection. In Fig. 3.2a the pipeline of R-CNN is shown, serving as an example of how the appearance within a region is used to identify what is in the image. In contrast, Fig. 3.2b shows the output of a YOLO network. This shows that complex objects (such as the dog), or partially obscured objects (such as the bicycle) can be classified by such a network.

While these detections can work in real time, they are not necessarily linked to the time series data: each classification counts for that particular image. This means that over a series of images, the classification of a region can change, disappear, or be consistently detected across all frames. Due to this, any tracking problem must also include a part that learns to detect and adjust for changes in the detections.

Motion analysis
There are also detection methods that make use of motion cues to create detections. While this seems intuitive in a scene where only one object moves, it becomes more complex when everything moves. Motion can be detected through a set of different techniques.

One way of identifying moving objects is background subtraction. When the background is known, this is removed from the image, and everything left within it are the objects that move across the field of view. This method, while effective, holds as an assumption that a model of the background exists to enough accuracy. This is difficult to guarantee for drones, as the background changes with the drones motion.

A second method is to follow interesting features. This requires features to have been found, by use of the detectors mentioned in Subsection 3.2.4 or features from Subsection 3.2.3, and found again in subsequent images. The benefit of this method is that not the entire image needs to be searched for features; methods as meanshift and camshift [15] search nearby areas in images to find the features again.

Another method is to use the optical flow present in an scene. Sparse techniques determine the velocity of individual pixels across the screen, where dense techniques determine the velocity of all pixels within an image. Machine learning algorithms, such as DeepFlow [16], determine the flow for every pixel in the input image, in what is called dense flow.

One feature of optical flow is that an approximate depth can be extracted. While a single pixel will not contain enough information to solve for depth, a group of pixels can be used to create depth estimates. This is explored in [6], which uses two consecutive images to estimate both the flow and depth of the image, treating it as two linked properties.

3.3. Drone Based Applications

In this chapter the focus lies on drone based applications for relative state estimation. Current solutions are moving away from purely centralised planning, where central, more powerful, processing units coordinate swarms. The benefits listed for this include from fault tolerance, adaptive behaviour, and scalability of swarms.

This study will focus on methods performed on-board drones. It will include methods in its discussion that include auxiliary systems, as well as methods that require specific on board components. All methods discussed will use vision in some capacity.

In Subsection 3.3.1 localization systems making use of vision systems in combination with secondary systems, such as GPS. In contrast, Subsection 3.3.2 highlights methods that do not make use of secondary systems. Finally, some remarks about the use-cases and applicability of these methods will be discussed in Subsection 3.3.3.

3.3.1. Vision based systems

In the visual spectrum, local positioning systems can be developed from visual cues via SLAM. These methods create maps of the local area, and use a history of locations to determine the camera's location. The more detailed a SLAM map is, the more accurate the representation of the drone location is. Often, SLAM is run in a centralised fashion, taking advantage of more powerful computers off board drones. This combines the images from multiple drones to create detailed maps quickly, but requires all drones to be connected.

Due to the bandwidth requirement of sharing entire images, research has been done into decentralised SLAM methods. The Distributed Online Outlier Resistant SLAM (DOOR-SLAM) [17] is an example of this, which starts by identifying three limitations affecting other methods of SLAM positioning;

- The need for offline centralised post-processing,
- The assumption that all drones are always in communication range, or
- The assumption that sensor data is pre-processed centrally.

DOOR-SLAM is an approach to avoid these assumptions, which focuses on sharing only processed data to drones within range. This allows for efficient and sparse communications. Essentially, each drone runs it's own SLAM, creating a sparse feature map by which it navigates. Instead of updating a central repository, drones only update each other when near enough. When within communications range, drones also use a distributed SLAM instead of singular SLAM. This allows for the drones to determine each others relative positions based on their positions within the shared map.

Clearly, this still requires some form of shared reference system and knowledge of each others identity. This is not a trivial assumption, as the absolute system is used to increase the accuracy and reliability of the position estimates. For flexibility in deployment, some researchers do away with global positioning, and focus on the relative positioning systems. In [18], the authors argue that the identity of a detected drone is not needed in a decentralised swarm. They show that by sharing the odometry within the swarm, each drone can confidently compute each others relative position, lowering the bar for relative positioning.

Another way drones can cooperate for relative localization is by fusing uncertain measurements. This idea is explored in [19], where multiple drones combined uncertain pose estimates of an object, and used that to both determine their own relative positions in a way similar to SLAM, but also refine the accuracy of their respective pose estimates.

In [20], the authors used the motion across time as a feature, identifying areas of interest as "Spatial-Temporal Cubes". By leveraging the detected motion of drones, regions of interested are identified for scrutiny and false positives are removed. This research showed that methods combining both appearance and motion cues outperformed those depending on just one of the two. This research is continued in [21], where Ashraf et. al. demonstrate a two-stage algorithm based on ST-cubes. Unfortunately, these methods are not real-time, and also look to future frames to make detections.

Although RGB-cameras are the staple sensor for drones due to their low cost and easy availability, in [1] Carrio et. al. demonstrate how stereo imagery simplify the drone detection pipeline. By looking for 'floating' patches, namely patches disconnected from the ground, airborne objects are detected, and then

further classified. This reduces the problem to a classification one, where drones need to be separated from other airborne objects such as balloons or birds, after which pose estimation can be applied.

In contrast, depth information can be deduced from the changes in the camera position and motion in the frame, as Hur and Roth show in [6]. This leverages the optical flow of a sequence to make predictions of future frames. To do so, a depth map of the scene is created, and refined by the differences between predicted and actual frame.

The last visual sensor that could be used is the Dynamic Vision Sensor, or Event Camera [2]. As mentioned in Subsection 3.2.1, this changes the interaction

Another more intrusive change is by making use of a Dynamic Vision Sensor or Event Camera [2], which does not follow the traditional paradigm of changes in frames used to detect motion. While still a new technology, these cameras update individual pixels as things change, resulting in a very clear feed of motion. A moving object shows up in these images as an outline changing pixels, which a network could train to classify. By aggregating changes over the last n seconds[6], shapes in the changes would start appearing that can be recognised by the human eye.

3.3.2. Drone Augmentations

In contrast to the previous section, these following methods require more more than just a a sensor on the drone.

In the first case, Pestana et. al. show in [5] how the presence of ArUco markers simplifies the pose estimation problem considerably, making the detection of friendly drones a simple thing for real-time applications. By using the orientation and size of the detected marker, the relative pose can be calculated. This is however dependent on the markers being correctly detected, and limits the use case of the methods to friendly drones and good lighting conditions.

The design of these fiducial markers is something that can vary, with ArUco codes being just one contender. For applications where visible lighting isn't guaranteed, patterns of LEDs can be applied, such as the UV LEDs applied in [8, 22].

3.3.3. Applicability

The methods presented in Subsection 3.3.2 at times outperform the methods in Subsection 3.3.1, but at the cost of the flexibility of it's implementation. For use-cases such as night time drone-light shows[7], passive markers such as ArUco codes are less than ideal, while some LED colours can interfere with the desired look of the show. On the other hand, cooperating drones such as in [8] do widen the operational window, as the presence of the UV LEDs enables function independent of lighting.

While drones in drone shows and other commercial swarms can be expected to cooperate with each other, there is no such expectation between differing swarms or lone drones. In the case of failures, or outright antagonism, the methods from Subsection 3.3.2 do not perform as well, and robust detection algorithms are preferred. In places such as airports and theme parks[8], ground installations can augment detection through use of sensors like stereo cameras as proposed in [1].

With the prevalence of RGB cameras in drones, collision avoidance systems largely depend on on the drone-to-drone detections using the standard RGB camera. While this can be done through cooperation, such as in DOOR-SLAM [17], robust methods such as the Spatio-Temporal Cubes [20, 21], an approach not yet running in real-time, are capable enough that they do not require the need for cooperation. These more independent methods ensure that drones can decentrally find position estimates of other nearby drones.

To more easily compare the current state of research, Table 3.1 scores the different papers cited above, as well as other related literature, on the points discussed in the preceding text. The first three directly relate to how well a proposed solution could operate in a decentralised swarm of MAVs. The fourth column, "No Comms", assesses whether the proposed solutions require drones to be in communication; this filters

[6] Or fractions thereof

[7] Such as during over Rotterdam, for Liberation Day in may 2020.

[8] Efteling Drone Interception System

out solutions that require cooperation between drones. The next column also questions the cooperative nature of drones, by verifying that the proposed solution does not use predefined markers, be they LEDs, colours, or icons. The rest of the columns all deal with the visual data requested. The first of these looks at whether the sensor used is a monocular CMOS camera, as this is the most common type of sensor on drones due to their inexpensive nature. While it would be tempting to combine the ”Motion Analysis” and ”Optic Flow” columns, they have been kept separate to help differentiate similar research. Finally, the ”Birds” column indicates whether the proposed detection algorithms distinguishes between drones and other flying objects, such as birds. While not a critical column, it gives an indication of how the papers work; for example, in [1] drones are detected by being objects disconnected from the ground, a state that could also apply to baseballs, birds, and clouds.

From the table, we can see that some papers propose solutions that almost meet all the challenges discussed previously, but no paper tackles them all. The closest solution is EVProp [23], a event camera based solution that used the detected optic flow to find propellers. This paper highlights how the motion information can be used to identify MAVs. However, it is not applicable to the large majority of drones, due to its choice of sensor. That a specialised sensor is required to achieve all the other criteria, identifies a gap in the research.

Paper	Source	Real-Time	Online	Decentralised	No Comms	Markerless	CMOS	Motion Analysis	Optic Flow	Birds
Collaborative Stereo	[24]	Yes	Yes	Yes	No	Yes	Yes	No	No	No
Deep Neural Network-Based Cooperative Visual Tracking Through Multiple Micro Aerial Vehicles	[2]	Yes	Yes	No	No	Yes	Yes	No	No	Yes
DogFight: Detecting Drones from Drone Videos	[25]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Drone Detection Using Depth Maps	[20]	No	No	Yes	Yes	Yes	No	No	No	No
Dynamic obstacle avoidance for quadcopters with event cameras	[26]	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	No
EVPropNet: Detecting Drones By Finding Propellers For Mid-Air Landing And Following	[22]	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
Flying Objects Detection from a Single Moving Camera	[1]	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Multi-Target Detection and Tracking from a Single Camera in Unmanned Aerial Vehicles (UAVs)	[27]	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Onboard Marker-Less Detection and Localization of Non-Cooperating Drones for Their Safe Interception by an Autonomous Aerial System	[28]	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No
Real-Time Object Tracking on a Drone With Multi-Inertial Sensing Data	[29]	Yes	Yes	Yes	Yes	Yes	No	No	No	No
Relative localization of helicopters from an onboardcamera image using neural networks	[6]	Yes	Yes	Yes	Yes	No	Yes	No	No	No
Self supervised Monocular Scene Flow Estimation	[23]	No	No	Yes	Yes	Yes	No	Yes	Yes	No
UVDAR System for Visual Relative Localization With Application to Leader–Follower Formations of Multirotor UAVs	[21]	Yes	Yes	Yes	Yes	No	No	No	No	No
Vision-Based State Estimation for Autonomous Micro Air Vehicles	[8]	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No

Table 3.1: Collection of papers, with columns denoting different aspects. As visible, no paper was found by the author that has only "Yes's" across all columns.

3.4. Conclusion

In Section 3.1 the following research question is formulated:

> What current techniques within the broader Computer Vision community can be applied on board drones to create a real-time state estimation of the relative positions of other drones?

In Section 3.2, a summary of the basics involved in computer vision is given. From this, we see that within the selection of hardware there is already a limitation on what commonly gets placed on drones, due to the costs and weights associated to the different components. Additionally, we see what features and parts of an image are interesting for computer analysis, and how images are decoded.

This reveals a problem with monocular vision, due to the loss of depth information. As an important hurdle for monocular drones, its presence is something that influences proposed solutions, and research is carried out to perform monocular depth perception.

Finally, the chapter ends distinguishing between two different approaches to seeing motion within machine learning. In this, it acknowledges the differences in processing power and memory between them.

Section 3.3 looks at the research performed on drone applications for state estimation and positioning, and how they are applied. It starts by exploring methods of visual positioning, through collaborative approaches, sharing both data and sensors. Then methods that expand or simplify the problem space by augmenting drones are touched upon, before the merits and drawbacks of the methods are discussed.

While some of these methods are found to operate on-board drones, others, such as [21], do not operate within the real-time constraints required for online use. One of the drivers of this distinction is the computational cost of methods, which in the past barred machine learning solutions from MAV applications. Increases in computational power such as [9] have opened this area of research for autonomous MAVs.

At the end of the chapter, a table is compiled to identify the research gap. From it, the thesis research will focus on the machine learning methods that perform real-time state estimation via monocular drones with RGB cameras as they are the most common sensor on drones. This means other sensors fall outside the scope of the thesis research. It will further focus on non-collaborative approaches due to the flexibility in use-case this offers, and greater scope of potential detections.

References

[1] A. Carrio et al. “Drone Detection Using Depth Maps”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 1034–1037. DOI: 10.1109/IROS.2018.8593405.

[2] Davide Falanga et al. “Dynamic obstacle avoidance for quadrotors with event cameras”. In: *Science Robotics* 5.40 (2020). DOI: 10.1126/scirobotics.aaz9712. eprint: https://robotics.sciencemag.org/content/5/40/eaaz9712.full.pdf. URL: https://robotics.sciencemag.org/content/5/40/eaaz9712.

[3] Mario Coppola et al. “A Survey on Swarming With Micro Air Vehicles: Fundamental Challenges and Constraints”. In: *Frontiers in Robotics and AI* 7 (2020), p. 18. DOI: 10.3389/frobt.2020.00018. URL: https://www.frontiersin.org/article/10.3389/frobt.2020.00018.

[4] OpenCV. *The OpenCV Reference Manual*. Release 2.4.13.7. OpenCV, 2019.

[5] J. Pestana et al. “A Vision-based Quadrotor Swarm for the participation in the 2013 International Micro Air Vehicle Competition”. In: *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2014, pp. 617–622.

[6] Junhwa Hur et al. *Self-Supervised Monocular Scene Flow Estimation*. 2020. arXiv: 2004.04143 [cs.CV].

[7] T. Baca et al. “Autonomous landing on a moving car with unmanned aerial vehicle”. In: *2017 European Conference on Mobile Robots (ECMR)*. 2017, pp. 1–6.

[8] V. Walter et al. “UVDAR System for Visual Relative Localization With Application to Leader–Follower Formations of Multirotor UAVs”. In: *IEEE Robotics and Automation Letters* 4.3 (2019), pp. 2637–2644.

[9] Daniele Palossi et al. “Ultra Low Power Deep-Learning-powered Autonomous Nano Drones”. In: *CoRR* abs/1805.01831 (2018). arXiv: 1805.01831. URL: http://arxiv.org/abs/1805.01831.

[10] Joseph Redmon et al. “YOLOv3: An Incremental Improvement”. In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[11] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: 1311.2524 [cs.CV].

[12] Kaiming He et al. “Mask R-CNN”. In: *CoRR* abs/1703.06870 (2017). arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870.

[13] Wei Liu et al. “SSD: Single Shot MultiBox Detector”. In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

[14] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/. 2016.

[15] G. R. Bradski. “Real time face and object tracking as a component of a perceptual user interface”. In: *Proceedings Fourth IEEE Workshop on Applications of Computer Vision. WACV’98 (Cat. No.98EX201)*. 1998, pp. 214–219. DOI: 10.1109/ACV.1998.732882.

[16] P. Weinzaepfel et al. “DeepFlow: Large Displacement Optical Flow with Deep Matching”. In: *2013 IEEE International Conference on Computer Vision*. 2013, pp. 1385–1392. DOI: 10.1109/ICCV.2013.175.

[17] P. Lajoie et al. “DOOR-SLAM: Distributed, Online, and Outlier Resilient SLAM for Robotic Teams”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1656–1663.

[18] M. Cognetti et al. “3-D mutual localization with anonymous bearing measurements”. In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 791–798. DOI: 10.1109/ICRA.2012.6225288.

[19] Eric Price et al. “Deep Neural Network-Based Cooperative Visual Tracking Through Multiple Micro Aerial Vehicles”. In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3193–3200. DOI: 10.1109/LRA.2018.2850224.

[20] A. Rozantsev et al. “Flying objects detection from a single moving camera”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 4128–4136.

[21] Muhammad Waseem Ashraf et al. *Dogfight: Detecting Drones from Drones Videos*. 2021. arXiv: 2103.17242 [cs.CV].

[22] Vrba Matouš. “Relative localization of helicopters from an onboard camera image using neural networks”. Master’s thesis. CZECH TECHNICAL UNIVERSITY IN PRAGUE, 2018.

[23] Nitin J. Sanket et al. “EVPropNet: Detecting Drones By Finding Propellers For Mid-Air Landing And Following”. In: *CoRR* abs/2106.15045 (2021). arXiv: 2106.15045. URL: https://arxiv.org/abs/2106.15045.

[24] M. W. Achtelik et al. “Collaborative stereo”. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 2242–2248. DOI: 10.1109/IROS.2011.6094866.

[25] Jing Li et al. “Multi-target detection and tracking from a single camera in Unmanned Aerial Vehicles (UAVs)”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 4992–4997. DOI: 10.1109/IROS.2016.7759733.

[26] Matouš Vrba et al. “Onboard Marker-Less Detection and Localization of Non-Cooperating Drones for Their Safe Interception by an Autonomous Aerial System”. In: *IEEE Robotics and Automation Letters* 4.4 (Oct. 2019), pp. 3402–3409. DOI: 10.1109/LRA.2019.2927130.

[27] Peng Chen et al. “Real-Time Object Tracking on a Drone With Multi-Inertial Sensing Data”. In: *IEEE Transactions on Intelligent Transportation Systems* 19.1 (Jan. 2018), pp. 131–139. DOI: 10.1109/TITS.2017.2750091.

[28] Eric Price et al. “Deep Neural Network-Based Cooperative Visual Tracking Through Multiple Micro Aerial Vehicles”. In: *IEEE Robotics and Automation Letters* 3.4 (Oct. 2018), pp. 3193–3200. DOI: 10.1109/LRA.2018.2850224.

[29] Thomas Webb et al. “Vision-Based State Estimation for Autonomous Micro-Air Vehicles”. In: *Journal of Guidance Control and Dynamics* 30 (May 2007). DOI: 10.2514/1.22398.

A

Dataset generation

The datset was developed by combining several files from different sources. The environments used were taken from `https://www.blender.org/download/demo-files/`, specifically the ”Barcelona Pavillion” and ”Classroom” files. In addition to that, on SketchFab a CC BY 4.0 licensed model of a S9 Mini Drone[1] was found. This appendix gives details on how these files were combined to create the dataset. To expand the dataset using new environments or new drones, those files merely need to be replaced.

The current dataset can be found at in the 4TU data repository under the DOI number 10.4121/21904635. The rest of this appendix details how to generate a new sequence to expand the dataset.

System Setup

Due to the processing power required to generate the amount of data, a headless server with GPUs was used. This required some additional setup beyond the normal use case as described in Blender tutorials. It is assumed that the reader will have access to a similar machine for rendering purposes.

This appendix also further assumes a version of Blender more recent than 2.8, such as 3.1.2. While older versions of Blender can be used, there was a significant rework of the Cycles rendering engine between 2.7 and 2.8 which improve the performance of the engine considerably.

Further, it is assumed that sequences will be rendered via Blender’s command line interface.

Combining files into one

The first step required is to combine all files into one. This is best done in a copy of the environment file, such as the Pavillion. The environment files can contain multiple scenes, which are used to create different effects. For example, Pavillion contains 6: midday, sunset, night, assets, and two benchmark scenes. Scenes are used within blender to manage the data and share assets between scenes. More detail can be found in the Blender manual[2].

Once a scene is selected, such as the Pavillion’s ’1 - midday’, the drone needs to be imported. This requires the drone to already be prepared for import, as the scales between the different objects might vary. Using the S9 Mini Drone as an example in Fig. A.1, an inspection shows that the model was not created in meters. This would cause scale differences if imported, with the drone appearing gigantic, so must be modified before import. For this dataset, the different parts were merged, and the entire drone was scaled down to 21cm$\times$24cm$\times$10cm.

To add the drone to the environment, the menu options `File > Link` must be selected. From there, the `.blend` file containing the drone is opened, and from the directory `Objects` the data objects that make up the drone are selected[3]. This will create a non-editable link within the environment file to the ”data object” of the drone. This includes all animations, textures, and mesh geometries, and will update with changes to original file. At this point, the drone should be visible in the environment.

[1] `https://sketchfab.com/3d-models/s9-mini-drone-cf3ed83c1b87486d90435f54c074e16e`

[2] `https://docs.blender.org/manual/en/latest/scene_layout/scene/introduction.html`

[3] For more information on linking data from other `.blend` files, see `https://docs.blender.org/manual/en/latest/files/linked_libraries/link_append.html`

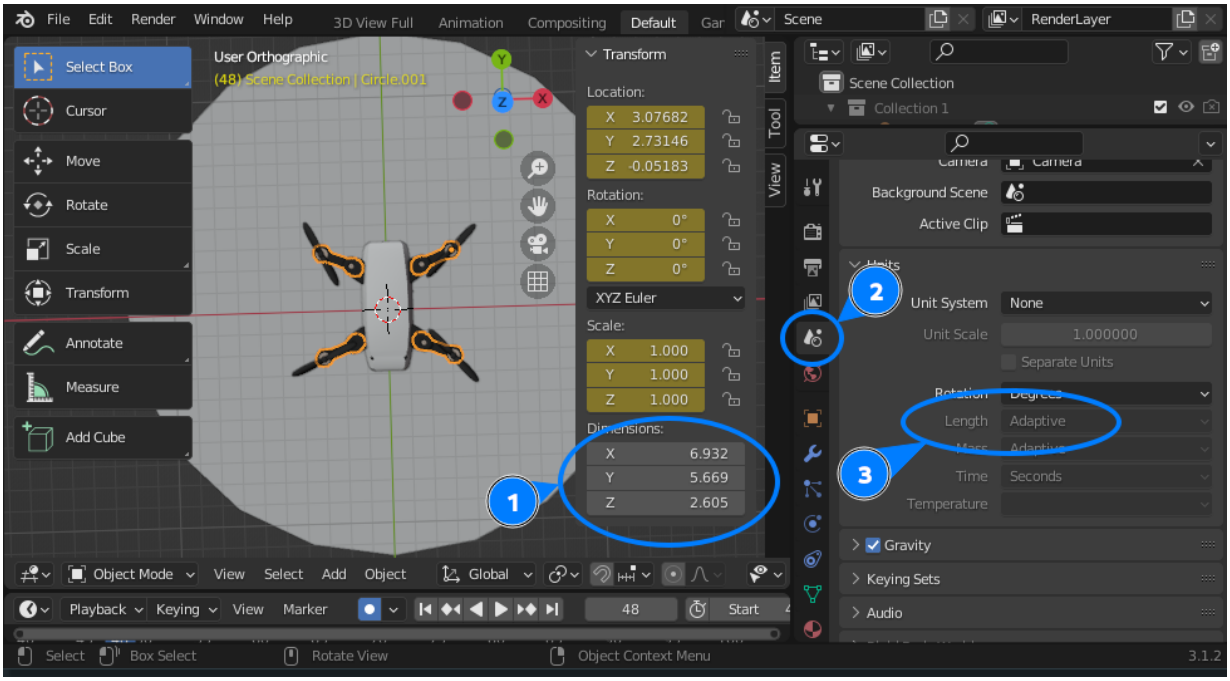


Figure A.1: Blender screenshot, showing part of the drone selected. Circled and marked at (1) are the dimensions. Selecting the menu option at (2) allows the user to set the units for the dimension at (3).

Animating the drone

The drone found in `S9_drone.blend` already includes animations for the rotors, but only for a period of 47 frames. These include animations for the rotation of the propellers, a side to side motion of the drone, and an small variation in height. While the motion of the propellers is useful for further use, to create longer sequences than 47 frames, the animation needs to be changed.

Animations in Blender can be created using f-curves. This gives animators a large amount of control in the types of motion they can create, as different kind of interpolation between keyframes are possible. As shown in Fig. A.2, it is possible to also set the f-curves to repeat themselves, with an offset. This allows for the drone, imported into any environments, to have it's propellers behave realistically. While doing this, it is important to remove any extra animations that the drone file has, as those would also be imported and potentially throw off the behaviour of the drone.

Treating the drone as one object, we can also animate it's motion to follow a flight path. The simplest way known to the author is by constraining it to follow a path. Having added any Curve object to the environment, the drone can be given an `Object Constraint`. The `Follow Path` constraint allows for the drone to follow any arbitrary path, without needing to keyframe the entire flight path. Figure A.3 shows how this is set up.

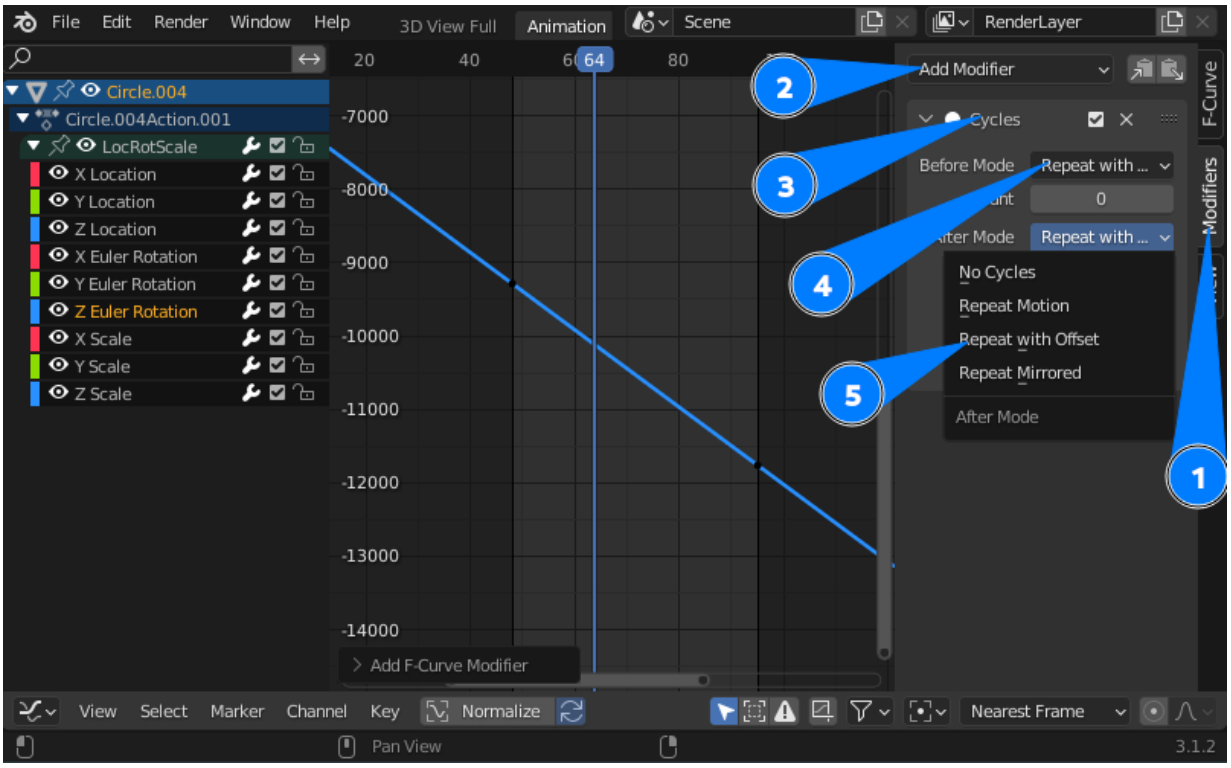


Figure A.2: A step by step guide on how to setup the cyclic animation for propeller motion. In the `Graph editor`, after selecting two consecutive nodes on the graph, one can add the `cycles` modifier (1-3) to the animation. By setting the `Repeat with offset` option to both before and after the selected nodes (4-5), the propeller will continuously spin up.

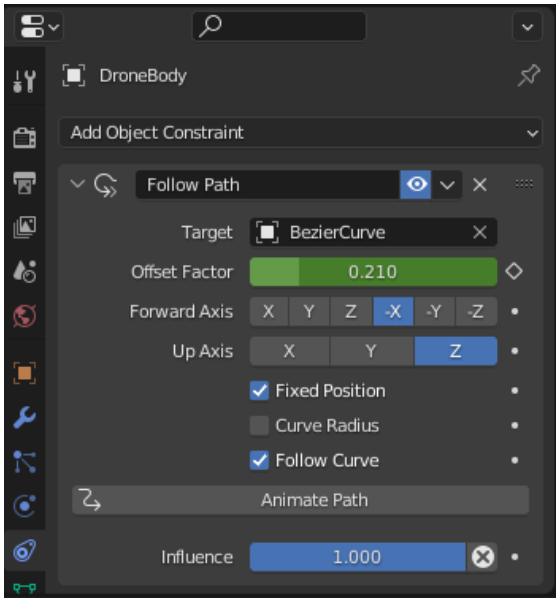


Figure A.3: A screenshot showing the menu where the object constraint is added, as well as the required settings. By keyframing the `Offset Factor` between 0 and 1, the position along the path can be set, with 0 being the start of the path, and 1 the end. Interpolation turns this into a smooth start and stop.

Exporting flow, depth, and segmentation

To export the flow, depth and segmentation data, attention must be turned to the Cycles engine. The first setting that must be enabled is in the `Render Passes`. There, as shown in Fig. A.4, there are four passes needed: `Combined`, which generates a colour image as seen by a normal camera; `Z`, which gives the depth in the image; `Vector`, which gives the optic flow; and `Object Index`, which is used to create segmentation masks.

The next step requires entering the `Compositor` view. The Pavillion file will already have many nodes set up for all the details of the final rendered image, but the flow, depth, and segmentation files will require their own nodes. This is done as in Fig. A.5. The file output node allows each output to be given a subpath, which can be used to sort them into folders by using "/" characters. Additionally, the file format, type, and colourspace of outputs can be defined there.

The segmentation masks require addition of the `ID Mask` node. This requires knowing which ID to input into the node, which can be found at `Object Properties > Relations > Pass Index`

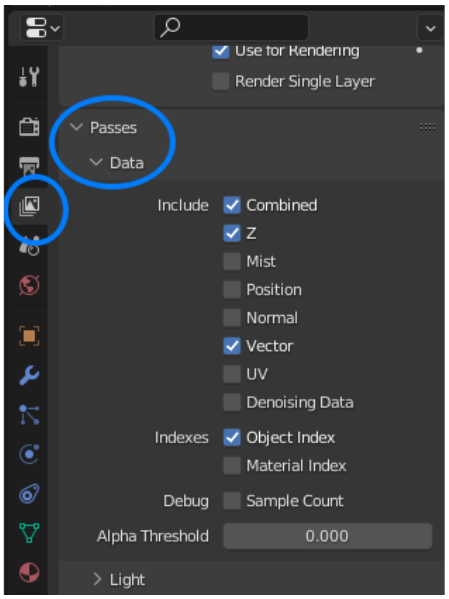


Figure A.4: The required render passes to be able to extract depth, optic flow, and segmentation data. The required menu tabs and heading are circled. Z is required for Depth data, Vector for Optic Flow, and Object Index for segmentation masks.

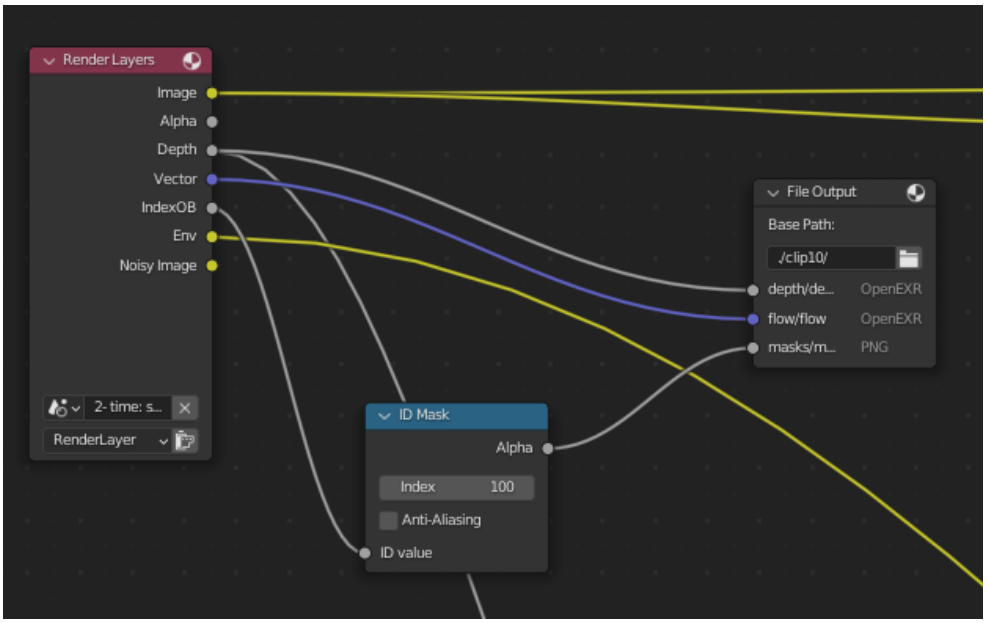


Figure A.5: This figure shows a small section of the compositing node tree used for clip10. On the left is the `Render Layers` node, which has as few outputs that can be saved directly. The `File Output` node on the right allows each input to it's node to be saved to a separate file, with it's own setting. The `ID Mask` node in the middle creates a black and white image based on the texture ID of an object. This is used to generate the segmentation masks.