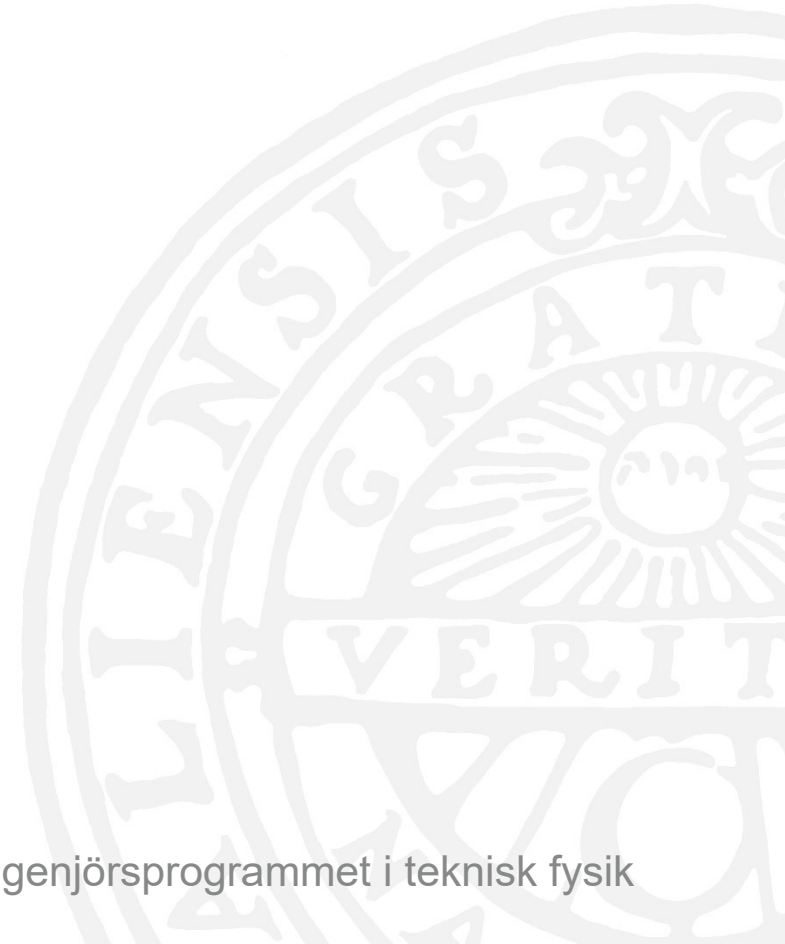
UPPSALA
UNIVERSITET

UPTEC F 22034
Examensarbete 30 hp
Juni 2022

Target Recognition and Following in Small Scale UAVs

Ellen Lindgren



Civilingenjörsprogrammet i teknisk fysik

UPPSALA
UNIVERSITET

Target Recognition and Following in Small Scale UAVs

Ellen Lindgren

Abstract

The industry of UAVs has experienced a boost in recent years, and developments on both the hardware and algorithmic side have enabled smaller and more accessible drones with increased functionality. This thesis investigates the possibilities of autonomous target recognition and tracking in small, low-cost drones that are commercially available today. The design and deployment of an object recognition and tracking algorithm on a Crazyflie 2.1, a palm-sized quadcopter with a weight of a few tens of grams, is presented. The hardware is extended with an expansion board called the AI-deck featuring a fixed, front-facing camera and a GAP8 processor for machine learning inference. The aim is to create a vision-based autonomous control system for target recognition and following, with all computations being executed onboard and without any dependence on external input. A MobileNet-SSD object detector trained for detecting human bodies is used for detecting a person in images from the onboard camera. Proportional controllers are implemented for motion control of the Crazyflie, that process the output from the detection algorithm to move the drone to the desired position. The final implementation is tested indoors and proved to be able to detect a target and follow simple movements of a human moving in front of the drone. However, the reliability and speed of the detection need to be improved to achieve a satisfactory result.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala

Handledare: Lars Forssell Ämnesgranskare: Roland Hostettler

Examinator: Tomas Nyberg

Popular science summary

An unmanned areal vehicle (UAV) is an aircraft without any pilot or passenger on board and controlled either remotely or with various degrees of autonomy, also referred to as a drone. This thesis aims to investigate the possibilities of autonomous control of small drones, specifically for the use case of object recognition and following. The main idea is to use an object detection model based on convolutional neural networks to detect a target of interest, and then use the result to control the movements of the drone to achieve object following.

The hardware used in this thesis is a quadcopter, an aircraft with four rotors, called Crazyflie 2.1. The drone weighs around 30g and can fit in your palm. The vehicle is equipped with a grayscale front-facing camera and a low-power processor specifically designed for image processing and machine learning workloads. Quadcopters are generally easy to fly and control, have high maneuverability, use vertical takeoff and landing, and can hover in the air.

UAVs with the ability to autonomously fly after and follow moving object is not completely new, but the systems commercially available today are either larger than the Crazyflie 2.1 and equipped with more powerful processors and sensors, or rely on some type of external communication, either for positioning or to send over data to a computer that executes complex computations before sending back commands to the drone. This thesis aims to implement a system independent of any external communication, running all computation on board the palm-sized Crazyflie 2.1. By doing this, any latency from external communication is avoided, as well as the dependency on signals such as GPS or WiFi, or being close to a base station.

An advantage of small drones such as the Crazyflie 2.1 is that they can be used safely indoors with only minimal safety measures and can fly in environments unavailable for larger counterparts. However, the small form factor of the drone becomes a challenge when creating a fully autonomous system. Memory, processing power, battery capacity, and the type of sensors that can be used are all limited. Computations need to be within the strict requirements on memory usage and stay within the power envelope to not drain the battery.

Object detection builds on image classification, a common problem in computer vision where you want to associate an image with a class that describes what the image represents; this is an image of a dog, a house, a plane, etc. Bounding box object detection takes this one step further by instead of only saying what the image represents, it also tells where in the image an object is localized by outputting a rectangular box that surrounds the object. During the last ten years, models built on convolutional neural networks (CNNs) have become popular for object detection. CNNs are a type of neural network that makes use of the two-dimensional structure in an image when analyzing it and have been proven to be very effective for image processing. This thesis is not focused on designing, training, and evaluating different models. Instead, a CNN model with pre-trained weights for detecting human bodies is selected. The chosen model

architecture, SSD-MobileNet is a fast and effective model for bounding box object detection, partly designed for embedded devices, that does not compromise the accuracy too much.

Coordinates of a bounding box surrounding a detected object are used by proportional controllers to compute commands for desired movements of the drone, to keep the target in the center of the camera image frame at the desired distance. The implementation is tested for forward and lateral motion of the drone with a step response in both directions respectively. The results show that the drone can follow simple movements. However, the SSD-MobileNet model sometimes fails in detecting a target, even if it appears in the image captured by the camera. The system does not quite reach real-time object detection, taking over a second to analyze each image.

This work can be improved with a faster and more accurate detection, which also would enable the design of better controllers. The algorithm could also be extended with a traditional tracking algorithm to ensure that the drone follows the same person, even if two people appear in the images captured by the onboard camera.

Acknowledgments

I want to give special thanks to my supervisor, Lars Forssell, for providing me the opportunity to work on this project at the Swedish Defense Research Agency (FOI), and for his enthusiasm and encouragement during this process. I also want to thank Mattias Vikgren, Erik Branzen, Robin Lilja, Johan Markdahl, Jan Markborg, and all others at FOI for their support and help during the time I've worked on this project. I would also like to thank Roland Hostettler, my subject reader, for our rewarding discussion and his valuable feedback on this report.

Contents

Abbreviations

Abbreviation	Meaning
CNN	Convolutional Neural Network
FPS	Frames Per Second
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LiDAR	Light detection and ranging
MCU	MicroController Unit
PULP	Parallel Ultra-Low-Power
QVGA	Quarter Video Graphics Array
ReLu	Rectified linear unit
RPN	Region Proposal Network
SIMD	Single Instruction Multiple Data
SLAM	Simultaneous Localization And Mapping
SSD	Single Shot Multibox Detector
SoC	System-on-Chip
UAV	Unmanned Aerial Vehicle
YOLO	You Only Look Once

1 Introduction

1.1 Motivation

The utilization of unmanned aerial vehicles (UAVs) has gained popularity in both research and commercial applications over the last decades. Autonomous drones can be used in a wide range of domains such as surveillance, monitoring, aerial photography, search and rescue operations, or product delivery. There is an advancing trend towards smaller, more intelligent, and autonomous UAVs due to the development of more efficient computational hardware and software. Smaller drones, particularly the four-rotor quadcopter, have become popular due to their size, maneuverability, low cost, and ability to fly safely indoors and operate near humans. This opens up for applications in environments previously inaccessible due to size and safety constraints, or environments that are hard to access or hazardous for humans [1]. UAVs can be an effective tool to substitute or support human operators in dangerous environments. Small drones can, for example, be used to perform gas sensing tasks in indoor environments [2], air quality monitoring [3], or assist in warehouse inventory management [4].

For a safe and effective autonomous flight, drones require some ability to sense and navigate their environment and therefore somehow need to gain knowledge and understanding of the surroundings. Humans and animals can learn how to move through an unknown environment by processing the information that is constantly perceived by our vision system, but providing the same capabilities to a drone is a challenging problem. The system must simultaneously be able to solve tasks in perception, positioning, and control. It becomes even more challenging if the use-case also includes interaction with other agents [5].

One common approach for acquiring a drone's position in the context of autonomous navigation is to use data from a global navigation satellite system (GNSS) such as the US global positioning system (GPS), European Galileo, Russia's globalnaya navigatsionnaya sputnikovaya sistema, which translates to global navigation satellite system (GLONASS), or China's BeiDou. GNSS systems might work well in some outdoor applications but it is not always reliable or available in all situations and is not suitable to use indoors since the signal can be blocked by buildings. Using GNSS can also pose a safety issue in applications with high-security demands, and relying on external signals can make the system vulnerable to the risk of signal jamming.

An alternative to GNSS for positioning is to use simultaneous localization and mapping (SLAM) algorithms, where sensor data is used to construct a local map of the environment that can be used for localization, navigation, and planning of a safe trajectory. Such algorithms can be based on different types of sensor data including light detection and ranging (LiDAR) and other distance-based sensors, or on visual information from different types of cameras. SLAM algorithms are however rather complex in terms of computations and memory intensive and can therefore be too resource-demanding for real-time execution on smaller drones

with limited computational power and memory. Real-time execution is crucial for a safe autonomous flight, where a fast reaction time is needed to react to what is happening around it and, for example, avoid collisions with dynamic obstacles.

The challenge with onboard computations on resource-constrained platforms such as small drones can be addressed both at an algorithmic level as well as on a hardware level. Methods based on convolutional neural networks can serve as an alternative to traditional approaches for perception and have been shown to enable navigation without creating a map of the environment [5]. But this type of algorithm has until recently been considered out of reach for execution on palm-sized drones [6, 7]. Over the last few years, energy-efficient neural network models specially adapted for usage on embedded and mobile devices, focusing on additionally reducing the memory footprint and the number of computations have been presented [8–10]. The developments of energy-efficient and computationally capable hardware devoted to low-power deep learning inference together with optimized network architectures have made it possible to deploy deep learning models with real-time inference on drones with a weight of only a few tens of grams [11].

1.2 Problem Formulation

This thesis considers the problem of controlling a palm-sized drone with limited resources in such a way that a target of interest is detected and followed. The aim is to explore the possibilities of implementing a fully autonomous system that only relies on data from onboard sensors as feedback to the control system, performing all computations onboard, without a connection to a base station, and without relying on any external positioning system such as GNSS.

The first task is to deploy an object detection algorithm based on convolutional neural networks onboard a small drone. The second task is to implement an algorithm for object following by controlling the movements of the drone, based on the result of the detection. The drone is equipped with a camera that captures images that are fed to the object detection model that outputs a bounding box surrounding a target if an object of interest is detected in the captured image. The bounding box coordinates are used to compute instructions for how the drone should fly to follow the detected object.

A pre-trained model for detecting human bodies is used to create a system for following a person, but the problem can be generalized to other types of objects by changing the detection model. The objective is simplified by only considering movements in the horizontal plane, where the drone is flying at a fixed altitude while following a human target moving on the ground.

The platform used is the quadcopter Crazyflie 2.1 from Bitcraze, further presented in Section 4.1, coupled with an onboard low-power camera, and a GAP8 System-on-Chip (SoC) from GreenWaves Technologies that is designed for running machine learning algorithms on low-power devices described in Section 4.2.

The two main goals in this thesis are the deployment of an object detection model based on convolutional neural networks on the selected hardware platform, and the development of a control algorithm for achieving object following based on inputs from the detection model. Training and evaluation of object detectors and convolutional neural networks are not considered, as well as evaluation of model performance.

2 Background

2.1 Quadcopters

The terms *UAV* and *drone* are used in general for aerial vehicles without an onboard human operator. The vehicles may be controlled remotely by a human or with various degrees of autonomy and can have different numbers of motors and wing configurations. Drones can be classified using different criteria such as weight, size, flight altitude operation, configuration, and degree of autonomy. Drones with a diameter on the centimeter scale and a weight of a few tens of grams are classified to be in the nano-scale [7]. One common type of drone is the quadrotor multi-copter, or quadcopter, shown in Figure 1 which is a type of UAV with four rotors that are used to lift and maneuver the vehicle. The quadcopter has capabilities such as hoovering and vertical take-off and landing which makes them easy to use and contributes to their popularity. In this thesis, the terms *drone*, *UAV* and *quadcopter* will be used interchangeable, where the terms *drone* and *UAV* most often will imply a quadcopter but also could refer to a general aerial autonomous vehicle.

Drones are often equipped with an inertial measurement unit (IMU) including a tri-axis accelerometer and a gyroscope. It is also common that drones to have a camera mounted on them, either fixed looking forward or down or on a gimbal that can rotate. They may also have support for positioning systems such as GNSS and communication over WiFi, Bluetooth, or other radio systems.

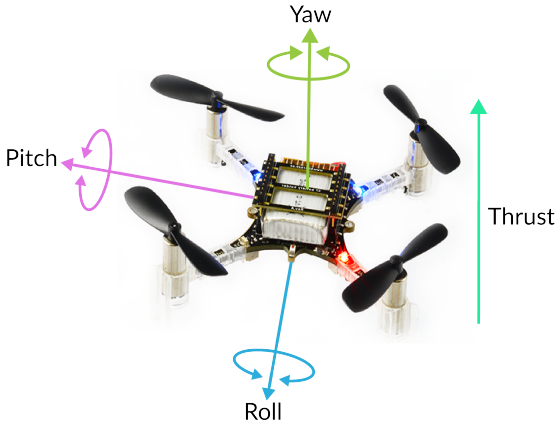


Figure 1: Dimensions of control for a quadcopter. Image source: Bitcraze.

There are four main dimensions for controlling a quadcopter, illustrated in Figure 1: roll, pitch, yaw, and thrust. Roll refers to the rotation around a horizontal axis going through the drone from back to front and is responsible for the horizontal movement to the left or right, which is achieved by slightly tilting the drone in the direction that it is supposed to fly. The horizontal forward and backward movements are controlled by pitch, which is the rotation around a

horizontal axis going from left to right through the drone. Similarly to roll, pitch will slightly tilt the drone up or down for moving backward or forward, respectively. Yaw is the rotation around a vertical axis through the drone and is used to change direction in the horizontal plane. Finally, thrust is responsible for the upward or downward motion.

2.2 Neural Networks

2.2.1 Introduction

In terms of machine learning, neural networks are flexible models with the ability to describe complicated relationships between inputs and outputs, and can be used for both regression and classification problems [12]. Neural networks were originally inspired by how the brain works, modeling the biological neural systems with neurons connected to other neurons, where one can send signals to activate another. Artificial neural networks are constructed similarly, with sets of neurons organized in layers connected to each other, where each connection between two neurons has an associated weight. A neural network can mathematically be described as several linear regression models combined with non-linear activation functions. The machine learning counterpart of a biological neuron performs a dot product between input signals from neurons in the previous layer and the weights associated with each connection, add a bias term, and applies an activation function.

Linear regression describes a linear relationship between input variables $x_1, x_2, ...,$ x_n and an output variable y by multiplying each input variable with weight and adding a bias term,

$$\hat{y} = W_1 x_1 + W_2 x_2 + ... + W_n x_n + b.$$

The parameters of a linear regression model are the weights $W_1, W_2, ..., W_n$ and the offset term b.

Using the biological simile, the activation function decides whether a neuron should be activated or not based on a weighted sum of its input. The purpose of the activation function h is to add non-linearity to the network model. Every neuron performs a linear transformation of its input, but the combination of two or more linear functions is also a linear function. Without the activation function, the network would just be a linear model, making it hard to learn any complex tasks. The activation function transforms the summed weighted input to a neuron into an output value that is fed to the next layer. A linear regression model and the extension with an activation function are illustrated in Figure 2,

Two common choices for activation functions are the logistic, or sigmoid, function, and the rectified linear unit (ReLu). The sigmoid function,

$$h(z) = \frac{1}{1 + e^{-z}}$$

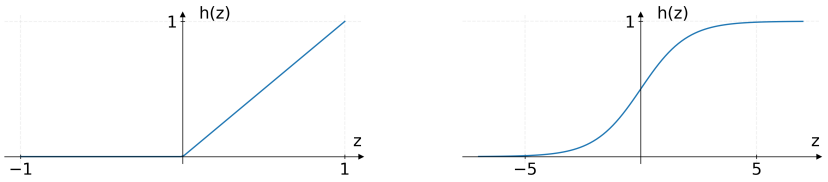
(a) (b)

Figure 2: Graphical illustration of a linear regression model (Figure 2(a)) and a linear regression model combined with an activation function denoted h (Figure 2(b)).

takes a real-valued input and squashes it into a range between 0 and 1. The function is linear close to $z = 0$, and large negative numbers become 0 and large positive numbers become 1. The simpler ReLu function,

$$h(z) = max(0, z)$$

is linear for positive inputs and equal to zero for negative inputs. The graphs for both activation functions are shown in Figure 3. The sigmoid function has been used frequently in the history of neural networks but has been replaced as the standard choice of activation function by the ReLu [12].



(a) (b)

Figure 3: Two activation functions commonly used in neural networks; ReLu (Figure 3(a)) and the sigmoid function (Figure 3(b))

In a fully connected network, each neuron of one layer is connected to all neurons in the previous layer but does not share any connections with neurons in the same layer. Figure 4 shows an example of a small neural network with two fully connected layers. Each connection, represented by blue arrows in the figure, has a weight associated with, where the bias term is represented as a neuron with a fixed value of 1. The parameters of the network are the weights and biases, and these can be learned from a training dataset using, for example, backpropagation.

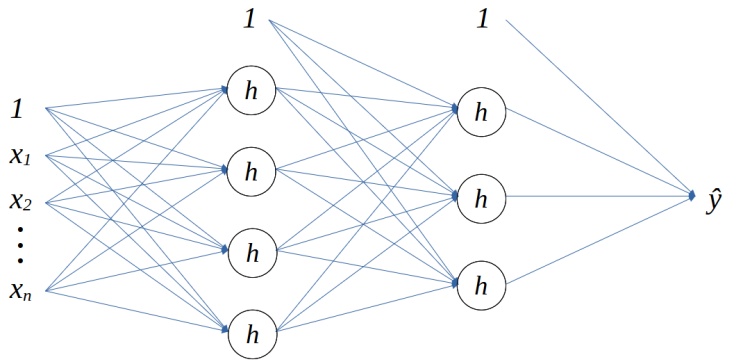


Figure 4: A two-layer fully connected neural network with four respectively three neurons in each layer.

2.2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a type of neural network with an architecture that makes them suitable for processing grid-like structured data such as time-series data, images, or 3D medical scans. CNNs are most commonly used for image processing, which is also the focus of this thesis. A grayscale image can be represented as a matrix where each pixel in the image corresponds to a cell in the matrix with a value from 0 to 1 for different shades of gray, 0 being total black and 1 representing white. Color images can, for example, be represented with three such matrices for representing the red, green, and blue color channels respectively, stacked together to create an input volume of depth three.

It is possible to put all the input variables or pixel values representing an image into a long one-dimensional vector and use a fully connected network to analyze the image. However, this approach does not take advantage of the structure typically present in an image. CNNs preserve this information by representing the input variables as well as the hidden layers as matrices. A convolutional neural network has at least one convolutional layer, a type of layer that can exploit the structure present in images by looking at small patches of the input to discover features such as edges, shapes, specific patterns, textures, and color contrasts. Two key features of the convolutional layer that differentiate it from a fully connected layer are *sparse interactions* and *parameter sharing*. While a hidden unit in a fully connected layer depends on all neurons in the previous layer, a hidden unit in a convolutional layer is only connected to a small, local region of the input. In a fully connected layer, each hidden unit has its own set of weights and biases for its connection to the input neurons. The hidden units in a convolutional layer share the same set of parameters but will depend on input from different spatial positions in the input because of the sparse interactions. The learnable parameters of a convolutional layer consist of a set of filters, or

kernels, plus a bias term for each filter. Filters are often spatially small in width and height but extend through the full depth of the input volume, and can be seen as feature detectors that scan the for specific features.

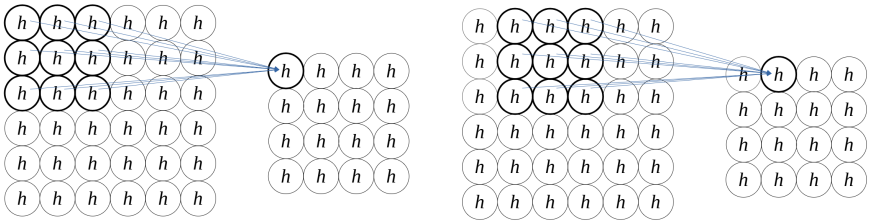


Figure 5: Graphical illustration of a convolutional layer with a 3×3 filter size. As for a fully connected layer each blue arrow represents a weight, but each neuron in the output layer is only connected to a small part of the neurons in the previous layer. The same weights are used for all neurons in the output layer.

The mapping between an input layer and the hidden units in a convolutional layer can be interpreted as a convolution between the input and filter, hence the name. The convolution operation takes the sum of products of the input and the kernel to return a scalar value. During a forward pass through a convolutional layer, each filter is convolved across the width and height of the input volume, computing dot products between entries of the filter and input at different spatial positions. This can be visualized as sliding the filter over the input volume, as illustrated in Figure 5. Each filter creates a two-dimensional activation map, and these are stacked together along the depth dimension to create the output volume. The depth of the output volume hence corresponds to the number of filters used, and each filter can learn to look for different features in the input. In CNNs for image classification, convolutional layers are often combined with a pooling layer explained in Section 2.2.3, and final fully connected layers before classification.

2.2.3 Pooling Layers

A CNN is commonly structured in blocks with one or more convolutional layers, each block followed by a pooling layer. A pooling layer aims to summarize and condense information from a previous convolutional layer by reducing the spatial size of its input. This also reduces the number of computations and trainable parameters needed to be learned during training and allows subsequent convolutional layers to work on larger sections of data.

Similar to convolutional layers, pooling layers only depend on small regions of the input, but do not introduce any extra trainable parameters. Pooling layer operate independently on every depth slice of the input and resizes it spatially, leaving the depth dimension unchanged. For example, a pooling layer with size

2×2 and stride 2 will down sample every depth slice in its input by 2 along both width and height, discarding three-fourths of the input features.

A common choice is to use max pooling, taking the maximum over small sections of the input features. Max pooling layers can be interpreted as keeping information about whether a feature is present in a region or not, but not keeping its precise position. A 2×2 max pooling layer with stride 2 is illustrated in Figure 6, down-sampling a 6×6 input to a 3×3 output. Another alternative is to use average pooling, instead of taking the average over small sections of features.

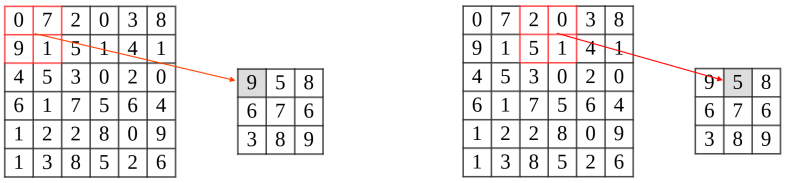


Figure 6: Max pooling layer with pooling filter size 2×2 and stride 2.

2.2.4 Depthwise Separable Convolutions

Standard convolution applies filters across all input channels and combines these values in a single step. Depthwise separable convolution breaks this up into two parts: a filtering stage using depthwise convolution, and a combination stage using pointwise convolution. This reduces the number of parameters and the computational complexity compared to a standard convolution.

Depthwise convolution applies a filter to a single input channel or depth slice at a time to create an activation map that only depends on one depth slice of the input, while the standard convolution applies filters with the same depth as the input volume. In depthwise separable convolution, as many depthwise filters as the number of input channels are applied to create an output volume with the same depth as the input. This step is followed by the application of 1×1 pointwise convolution to compute a linear combination of each spatial point across their depth. Each filter used in the pointwise convolution stage creates a two-dimensional activation map of the same spatial size as its input. These maps are stacked together to create an output volume with the same depth as the number of pointwise convolutional filters used.

2.3 Object Detection

A classic and fundamental problem in computer vision is image classification, where the goal is to identify which one of a predefined set of classes an image falls under. Methods for object detection take this one step further by tackling two problems: classification and localization. These types of methods do not only identify if an image contains certain classes of objects but also determine

their location within the image. General object detection algorithms input an image and produces rectangular bounding boxes surrounding detected objects, with associated class labels, and can handle both multiple classes and multiple occurrences of objects within the same image. These types of object detection methods cannot, however, determine the shape of a detected object. This can instead be done with instance segmentation, where each pixel is classified to belong to a particular object. Instance segmentation is more resource-demanding than object detection and therefore not considered in this thesis.

Traditional approaches to image classification and object detection are based on feature descriptors. A feature is a measurable piece of information about the content of an image, such as an edge, a corner, a specific shape, or a region of interesting points. In the context of image classification and object detection, interesting features are those that can help to identify objects in the image. Feature-based object detection methods generally require manual fine-tuning of numerous parameters, leaving it up to the designer of the algorithm to identify which features are important [13]. It can be a long process of trial-and-error to decide which features best describe different classes of objects.

Methods based on deep neural networks have become increasingly popular in the field of computer vision. The development of deep learning algorithms is powered by the improvements in computational hardware and the increased availability of training data. Convolutional neural networks can effectively extract complex features and therefore serve as a good alternative to the handcrafted traditional feature extractors. Deep learning also has been shown to enable higher accuracy in tasks such as image classification and object detection compared to traditional techniques [13].

Machine learning methods such as neural networks are trained rather than programmed, and the training exploits the massive amount of visual data that is available in many contexts today. In supervised learning, a model is trained on an annotated dataset to discover underlying patterns in the data. In contrast to classical techniques that are more domain-specific, methods based on neural networks can be re-trained or fine-tuned on a custom dataset for new use cases, which provides more flexibility.

The performance of deep learning methods comes with some trade-offs such as computational complexity, training time, and the requirement of large datasets. The task of preparing and annotating a dataset for supervised learning can be tedious and time-consuming but is very important since it affects the model's performance. Features learned by a neural network are specific to the dataset it was trained on, and the accuracy of a model is therefore affected by the size and quality of the dataset used for training. To perform well on new, unseen data, a training dataset must be large enough and properly constructed. This also implies that it is difficult to construct an object detector for classes not already well-documented [14]. If only a limited amount of labeled training data for a specific task is available, the performance of a model can be increased by using the available data to fine-tune a model that is pre-trained for an auxiliary

task. Models that are pre-trained on a large, general dataset can also be used as a base model to reduce training time for a specific task.

2.4 Onboard Visual Navigation on Small Drones

One way to overcome the limitations in available hardware on nano-sized drones is to offload computationally and memory-intensive tasks to a remote base station or a network-connected server using a wireless communication link. However, these approaches come with some drawbacks. The communication between a base station and the drone introduces latency, limits the operative range, and creates a sensitivity to external disturbances and signal jamming. Uploading sensor data to the cloud will not work without a network connection, and can cause safety and privacy issues if the data is exposed to the cloud. By performing all computations onboard, the drone is not required to continuously maintain a stable communication link with a base station which creates better reliability, also removing overhead from transferring sensor data from the drone to the base station, and receiving results to act on, as well as the risk of losing connection, and vulnerability to signal inference. The operating range can also be extended since there is no requirement to stay within a limited distance to the base station.

The limited battery life on nano-sized drones calls for effective hardware and optimized algorithms if they are to be running onboard in real-time without draining the battery. Onboard computations can, however, open up a wide range of applications, for example with high-security requirements or in inaccessible areas where it is not possible to be close to a base station.

Another alternative for motion planning and control in unknown environments is to use visual information captured by an onboard camera, without creating a map of the environment. Being equipped with a camera enables the drone to perceive the environment around it, and a camera coupled with an image processing algorithm can create a reliable system for autonomous navigation [5]. Cameras today can be small, lightweight, and inexpensive, which is advantageous in the setting of nano-sized, low-power, and low-cost drones, and still be good enough to be used as a base for autonomous navigation. However, the interpretation of complex, high-dimensional data such as images can be challenging. In recent years, machine learning approaches such as deep learning models based on CNNs have proven to be very successful when operating on this type of data.

Traditional systems for autonomous navigation and control are often structured into two modules: perception and control. Raw sensor data is processed by the perception module to derive a high-level state that provides relevant and meaningful information for the task to be solved, and the control module uses this information to determine control signals to be provided to the hardware. For example, a system can use GNSS data to determine its position which is useful information for the control module, and algorithms such as SLAM use

sensor data to build a map of the environment that can be used for navigation and control. In the context of this thesis, another example of a high-level state is the image coordinates for a bounding box surrounding a detected object, which can be used to compute control commands to achieve object following.

Depending on the platform, the control signals generated by the control module could either be direct low-level inputs for controlling the motors, or target points such as the desired velocity that is then given to optimized low-level controllers. The Crazyflie 2.1 platform uses the latter, where a low-level controller already is implemented in the open-source software for the platform, responsible for keeping the drone stable during flight and achieving given set points by controlling the motors.

An alternative is to use end-to-end models for autonomous control. Such methods directly map raw sensor data to control actions, either as direct low-level control outputs or as set points for a low-level controller. Deep neural networks have proven to be successful for this type of end-to-end learning. Supervised training of an end-to-end model requires a dataset with ground truth annotations for control, and this can come from various sources. One example is to use a trained operator to control the robot in the desired way, or to use a hand-designed or learned controller. It could also be possible to use data collected by humans or other vehicles, as long as it applies to the same problem. Data collected by cars and bikes have shown to be able to transfer to the navigation of drones [5]. This approach exploits the data available today and reduces the demand for a custom dataset.

Both approaches, traditional and end-to-end algorithms, have advantages and disadvantages depending on the application and the available data. A mediated approach where the controller is hand-designed gives the engineer explicit control over the resulting behavior and is more transparent and easier to debug. If the drone behaves unexpectedly, the designer can inspect the higher-level state to determine whether the problem is in the perception or the control module. Using object detection and following as an example, one can start by checking if objects of interest are detected with good accuracy. If not, the detection algorithm must be improved, and otherwise one can continue by evaluating the controller. An end-to-end approach on the other hand is conceptually simple and has the potential to be very computationally efficient and requires less prior knowledge about the system. Learning the coupling between perception and control in an end-to-end manner can provide a simple, lightweight system with high generalization abilities [5, 7]. An end-to-end model can be seen as more of a black box, and it can be harder to understand how and why the model works, or more importantly, why it might not work in some situations.

3 Related Work

3.1 Object Detection

3.1.1 Two-stage Object Detection Methods

The popularity and use of convolutional neural networks for image classification and object detection have increased tremendously over the last ten years, starting in 2012 when the use of CNNs contributed to a substantial boost in image classification accuracy on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [15]. Since then, many new computer vision models based on CNNs have been presented.

Object detection can be approached as an image classification problem by sampling boxes from an image, and applying an image classifier to each box to classify them as either objects or backgrounds. The question is then how to sample boxes effectively. One alternative is to use a sliding window approach where boxes of different sizes and aspect ratios slide over the image, applying a classifier to each sample to perform detection. However, applying an image classifier is an expensive operation so a more effective alternative is to use a region proposal algorithm to extract proposals for areas that may contain an object. One early method for bounding box object detection using region proposals combined with convolutional neural networks is R-CNN, Regions with CNN features [16]. The model uses selective search for region proposals to extract around two thousand regions of the image that might contain objects. A CNN based on AlexNet [15] is then used to extract feature vectors from the region proposals, and each region is finally classified with a category-specific support vector machine (SVM).

While R-CNN was successful at its time, the method has some limitations. It is still a slow algorithm since an image classifier is applied to each of the over two thousand region proposals. Another problem is that it is not a model that can be trained end-to-end, but consists of three independent parts; selective search for region proposal, CNN for feature extraction, and SVM for classification. A lot of work has been put into finding end-to-end alternatives to this approach, and two of its most famous successors are Fast R-CNN and Faster R-CNN.

Fast R-CNN builds on the same method as R-CNN, but with improvements for a higher training and testing speed as well as an increased detection accuracy [17]. Instead of using individual feature maps for each region proposal, Fast R-CNN applies a CNN on the entire image to create a feature map and then uses a region of interest pooling layer to extract feature vectors from this feature map. Each feature vector is then fed to a sequence of fully connected layers for classification and regression.

The bottleneck for Fast R-CNN and other effective region-proposal-based object detection methods is the computation of the region proposals, not the detection network itself. This bottleneck was addressed with Faster R-CNN by introduc-

ing a region proposal network (RPN) to replace the time-consuming method for region proposal [18]. The RPN shares convolutional features with the detection network, reducing the time for generating region proposals and making the model end-to-end trainable.

3.1.2 Single-stage Object Detection Methods

Approaches like R-CNN and its successors are called two-stage detectors since they break down the problem of object detection into two stages; identifying region proposals, and classifying objects within these regions. Even though the methods have developed to be more effective, they still do not reach real-time detection results. New architectures have been presented that address the bottlenecks of two-stage detectors to enable real-time detection and two of the most popular ones are You Only Look Once (YOLO) [19] and Single Shot Multibox Detector (SSD) [20]. Unlike the R-CNN family, these models do not use region proposals. They are called one-stage or single-stage detectors, referring to the fact that the task of object localization and classification is done in one forward pass of a single network, simultaneously predicting bounding boxes and associated class confidence scores. Two-stage detectors can achieve very accurate results but are typically slower than single-stage networks since they require multiple iterations to take place in the same image. Single-stage detectors provide a simpler pipeline for end-to-end training and can achieve a higher speed since only one iteration over the image is needed for detection.

YOLO provided further improvements in terms of speed [19]. Instead of applying the model to an image at multiple locations and scales such as with R-CNN and its successors, a single network is applied to the whole image for predicting bounding boxes and class probabilities. The name comes from the fact that the model only takes one look at the image to predict what objects are where, instead of first using a region proposal method to generate potential bounding boxes and then running a classifier on these proposed boxes.

SSD is another single-stage object detection method that similarly to YOLO, only uses one forward pass of the network to simultaneously predict bounding boxes and associated classes [20]. The SSD model uses a feed-forward convolutional network that produces a fixed number of bounding boxes and confidence scores for the presence of object class instances in those boxes. The SSD model finishes with a step called non-maximum suppression, an algorithm that inputs a list of proposal boxes and their corresponding confidence scores and outputs a list of filtered proposals where highly overlapping boxes are grouped together into a single box.

The core idea of SSD is to predict box offsets and category scores for a fixed set of default boxes of different aspect ratios using 3×3 convolutional filters applied to feature maps at different scales. The model is mainly built on convolutional layers, and can roughly be divided into three stages: a feature extraction stage, the actual detection heads, and a post-processing stage.

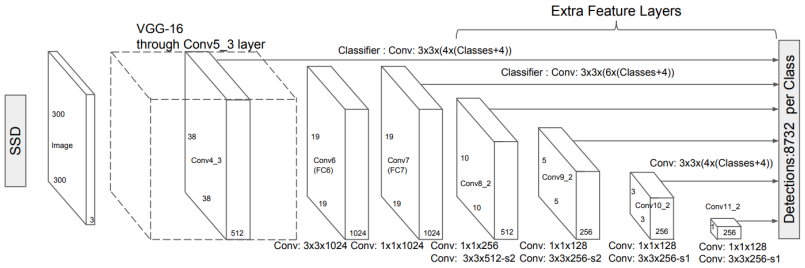


Figure 7: SSD architecture with VGG16 as feature extraction network [20].

The feature extraction stage is based on standard architecture for image classification but truncated before any classification layers. The SSD model presented in the original paper uses an architecture called VGG16 [21] as a basis for the feature extraction stage as shown in Figure 7, but other convolutional architectures for image classification could be used for feature extraction as well. VGG16 was selected because of its ability to extract useful features from images and strong performance in high-quality image classification tasks. VGG16 consists of five convolutional blocks with increasing depth, each block followed by a max pooling layer to decrease the spatial size. VGG16 ends with a max pooling layer followed by two fully connected layers before the output layer for classification, but this last part of the network is removed when used for feature extraction in SSD. The output from the last convolutional layer in VGG16 is used as a feature map on which to base the detection. More efficient networks for image classification have evolved since SSD was presented, such as MobileNets explained in Section 3.1.3.

At the end of the base network for feature extraction, additional convolutional feature layers with progressively decreasing sizes are added. The intermediate output from these layers is kept to allow the detection to be done at various scales since each feature map contains information about larger and larger regions of the image. The detection heads can also use the output from existing feature layers from the base network.

Every feature map cell is associated with a set of default bounding boxes, or anchors, of different dimensions and aspect ratios. The detection heads, which also consist of convolutional layers, have the role of producing offsets from these default boxes and class confidence scores for each box to generate box prediction. The detection heads can output repeated detection for a single object, so the final post-processing stage uses non-maximum suppression to group together highly overlapping boxes into a single detection.

3.1.3 MobileNet

MobileNet [8] is an efficient and lightweight network architecture for image classification that is built on depth-wise separable convolutions and designed

for mobile and embedded vision applications. It achieves high accuracy, but not as high as some full-fledged networks. The architecture strives to minimize the latency of smaller-scale networks, so that computed vision applications can run well on embedded and mobile devices. The 28-layer network uses depthwise separable convolutions explained in Section 2.2.4, except for the first layer which uses a full, standard convolution. All convolutional layers are followed by a batch norm and ReLu non-linearity, including both the depthwise and pointwise convolutional layers.

The original paper compares the performance of MobileNet with depthwise separable convolutions to a corresponding network with standard convolutions. The performance for image classification on ImageNet, a well-known benchmark dataset, only drops one percent while using significantly fewer parameters and multiplications. MobileNets uses 3 x 3 depthwise separable convolutions, which results in eight to nine times fewer parameters and multiplication operations compared to standard convolutions, depending on the depth of the output layers.

3.2 Object Following

In [22], a UAV platform that autonomously detects, tracks, and follows another drone is presented. The system is based on a deep learning model for object detection combined with a simple control algorithm. All computations are performed onboard on a Jetson TX2 processor and the system does not rely on any external signals for positioning. A target drone is detected in images captured by a camera mounted on the hunting drone. The object detection model used is a lighter variant of YOLO, modified to only detect a 'drone' class and trained on a custom dataset containing almost 60 000 images. The bounding box coordinates of a detected target are sent to the navigation control system. If a target is detected, a direction and a step size are calculated based on the location of the bounding box relative to the center of the image. Both a proportional and a constant step size policy are tested. For the constant step size, an input image is divided into a 3x3 grid where each cell is associated with a given direction and step size. The values corresponding to the cell in which the center of the target's bounding box is located are used as input to the flight controller. The proportional policy calculates the step size and direction proportionally based on the distance between the center of the detected target's bounding box and the camera image center. When tested on a custom hardware platform, the hunting drone was able to successfully follow a target drone, unless the target moves erratically. While the detection algorithm achieves a frame rate high enough for real-time tracking without compromising the accuracy too much, the frame rate and the detection accuracy are mainly what limit the performance. Both a gray-scale model and a 3-channel RGB (red, green, blue) model were tested, and a comparison showed that the gray-scale model gave more than 60% speedup with only a small loss in accuracy compared to using color images.

The problem of controlling a mobile ground robot with a fixed camera such that

a target of interest is detected, tracked, and followed while moving through a complex, unstructured environment is tackled in [23]. A deep learning algorithm is used to detect targets (in this case humans) in images captured by the onboard camera. Bounding box coordinates of a detected target provide information about the size and position of the target in the image, which can be used to compute control commands for the robot. To satisfy the real-time constraints of the applications, an SSD-MobileNet architecture is used as a detection model. The control objective is to drive the detected target's bounding box towards the desired set point by suitable choices of control input commands. One possible choice of the desired set point is a bounding box of a specific size located in the center of the camera image. However, the robot is assumed to move on the ground with a fixed camera and so the vertical position of the bounding box is therefore not considered to be controllable. Also, since the robot cannot control the orientation of the target, the height and width of the bounding box are considered to provide some redundant information. Instead, the height and width are combined to a single measure of the length of the bounding box diagonal. The control objective becomes to match the length of the detected target's bounding box diagonal and the horizontal position of the bounding box with an ideal set of values. A proportional-integral controller is used to compute control inputs for the robot's forward motion and yaw rate to achieve this. The proposed control algorithm was tested both using simulations and through real experiments, and both showed successful results for following the target and maintaining it within the camera's field of view.

Another example of a drone that uses an object detection algorithm to produce input for computing control commands for autonomous navigation is presented in [1], specifically for the application of inspection of wind turbines. An autonomous system could replace manual inspection of wind turbines with drones, which requires an expert pilot to monitor the camera as well as the movements of the drone. The task is solved by first locating a wind turbine, navigating toward it, and finally following a predefined path for inspection. The presented system is based on a sensor consisting of two deep convolutional neural networks trained to detect wind turbines, each coupled to an image sensor in a stereo vision camera. The system uses YOLO with a ResNet50 feature extraction network for object detection, and the center of detected bounding boxes is used to navigate toward the wind turbines. The distance between the drone and the wind turbine is calculated using stereo vision. Even though some noise was present due to the drone not being completely stable, and noise from the object detection algorithm, the sensor data showed to be accurate enough to be used for servoing after some filtering.

3.3 Applications on the GAP8 processor

A demonstration of how the GAP8 embedded processor, described in Section 4.2, can be used to enable autonomous visual navigation onboard a Crazyflie 2.1 using a convolutional neural network model and images captured by a low-

power front-facing image sensor is presented in [6]. Here, the task of following the movements of a human while staying at a constant distance from the subject is addressed. The model is called PULP-Frontnet and leverages the PULP-based GAP8 embedded processor, and is successful in following the pose of a moving human. A shallow neural network with four convolutional layers and one fully connected layer inputs gray-scale images from the front-looking camera and outputs a prediction of the target's pose in three-dimensional space as a vector (x, y, z, θ). The predicted pose is used to calculate a low-level setpoint for the drone to make it stay in front of the subject at the desired distance. The setpoint is executed by a low-level controller based on the open-source firmware for the Crazyflie 2.1 provided by Bitcraze. The performance is evaluated on three different model sizes with different input image resolutions, two with 160 x 96 pixels and one with 80 x 48 pixels. The smallest model achieved a frame rate of 135 FPS using 8-bit quantization when being deployed on the GAP8 processor.

An example of object detection using the GAP8 is described in [24], where a low-power device for automatic license plate recognition is presented. The system combines the GAP8 embedded processor with a low-power image sensor. The proposed system operates in two steps: detection and recognition. An input image is first processed by the detection stage to identify license plates, which are processed to recognize the registration numbers. This multi-model inference approach uses an SSD object detector based on MobileNet for detecting license plates, and the result is cropped and resized before being fed to a license plate recognition model (LPRNet) for optical character recognition. To reduce the number of parameters and computational complexity of the model, the SSD detector is replaced with a lighter counterpart called SSDLite. The LPRNet architecture is also refined to reduce the memory footprint, and operations not supported by the platform are removed. Images are captured by a low-power Himax QVGA grayscale camera (320 x 240 pixels), the same type of camera used on the AI-deck in this thesis. The final model recognizes registration numbers down to a cropped size of 30 x 5 pixels and achieves a throughput of 1.09 frames per second (FPS). The SSDLite-MobileNet model for object detection alone achieved a throughput of 1.73 FPS.

3.4 End-to-End Neural Networks for Autonomous Navigation

An alternative to the traditional mapping-localization-planning approach for autonomous navigation is to use end-to-end neural network models, trained to directly output commands that can be used for flight control. Instead of using processed sensor data to compute control commands, the control policy is learned directly from raw sensor data by deep neural networks.

One example of a closed-loop end-to-end neural network-based visual navigation system is called DroNet [5]. DroNet enables autonomous control of a drone equipped with a camera when flying through the streets of a city. The system is based on a lightweight and relatively shallow neural network with a residual

CNN architecture that processes visual information directly to produce output control commands for a flying drone. The model utilizes supervised learning and is trained on data collected by cameras on cars and bicycles. These vehicles are already integrated into the traffic system of a city, and the model learned to follow basic traffic rules from the training data. The policy learned by DroNet is also shown to be able to generalize well to new, unseen environments, different from the training, and allows successful flights in indoor corridors and parking lots. Input images from the camera are fed to the network model which predicts two outputs; a collision probability to let the drone recognize dangerous situations, and a steering angle to keep the drone navigating safely. The drone is controlled by modulating the forward velocity by a low-pass filtered version of the collision probability and modulating the yaw by a low-pass filtered version of the predicted steering angle. The network complexity and processing time are reduced by letting the two tasks share all residual layers but be inferred by two separate fully connected layers independently, and the two tasks were trained on separate datasets. The model is tested on a Parrot Bebop 2.0 drone with a weight of over 2.5 kg, and the velocity commands are produced by running the network on an Intel Core i7 2.6 GHz CPU receiving images at a frame rate of 30 FPS from the drone through WiFi.

DroNet has been adapted to work on the Crazyflie platform, running all computations onboard the nano-sized drone. For this task, a printed circuit board called PULP-Shield has been developed, featuring the PULP-based GAP8 SoC for running neural network inference and a low-power Himax camera [11]. The memory footprint and computational load of the model are reduced to fit within the resources available on the GAP8 processor and to meet the real-time constraints of the application. The original DroNet uses a 32-bit floating-point representation of the neural network, but the GAP8 processor does not support floating-point operations so the model is reduced to a 16-bit fixed-point representation, also resulting in a smaller memory footprint of the model.

The network is also re-trained from scratch using quantization-aware training, and the training dataset used for the collision probability is extended with a dataset collected by a Himax camera, the same as the one used in the drone platform. Together, these changes did not affect the performance of the model in any negative way. The PULP-shield is a forerunner to the AI-deck used in this thesis, further described in Section 4.2.

In [25] three approaches, two versions of a mediated model and one end-to-end learning model, were tested and compared for solving the task of hovering in front of a freely moving human, following its movements. The first mediated approach learns a mapping from input images to a pose estimation of the human subject and uses this together with an estimation of the drone's current measured velocity to calculate control signals for the drone. This approach inspired the PULP-Frontnet model described in Section 3.3. The end-to-end model directly learns a function to predict control commands based on input images and the current velocity of the drone. The first approach requires ground truth

information about the pose of the human subject and the end-to-end model requires ground truth values for the control commands. Ground truth values for control signals could either be obtained by recording a skilled human pilot or from a designed controller that is given the ground truth pose information. The third approach learns a model to estimate the target's pose similar to the first approach, but instead of using a hand-designed controller to produce control signals, a mapping is learned from the estimated pose and the drone's measured velocity to control commands for the drone. This approach requires ground truth values for both the target's pose and control signals but does not require the design of a controller as in the first approach.

In some cases, end-to-end learning is slower to converge and requires more training samples compared to mediated approaches [26]. This comparison did, however, not show any significant differences in quantitative performance, learning difficulty, perceived quality of robot behavior, or robustness to challenging inputs when comparing the different approaches, but with slightly smoother trajectories with the end-to-end approach.

4 Hardware Platform

4.1 Crazyflie 2.1

The drone platform used in this thesis is the quadcopter Crazyflie 2.1 manufactured by Bitcraze AB, shown in Figure 8. The platform has an open-source structure in terms of both hardware and software and is mostly used in research projects. The drone has a take-off weight of 27 g including battery and a size of 92 x 92 x 29 mm. This small form factor enables it to be used indoors with minimal safety measures. The drone is equipped with a pressure sensor and an IMU including an accelerometer and a gyroscope and supports communication over radio. The capabilities of the drone can be extended by adding extra hardware called *decks* that can be placed either over or under the main platform. The four DC motors allow a maximum weight of 42 g which enables up to 15 g of additional hardware and achieves a baseline flying time of 7 minutes.

The Crazyflie 2.1 features a heterogeneous multiprocessor architecture with two microcontrollers:

- an STM32 main application microcontroller unit (MCU) with a 168MHz Cortex-M4 core, 192kb SRAM and 1Mb flash,
- an nRF5 radio and power management MCU with a 32MHz Cortex-M0 core, 16kb SRAM and 128kb flash [27].

The STM32 is responsible for reading sensor data, runs the control and estimation algorithms, and is interfaced with the motor drivers and the deck expansion interface. The nRF5 is responsible for the radio communication, enabling power to the rest of the system, and detecting and checking installed expansion decks. In the configuration used for this project, the hardware platform is extended with two decks from Bitcraze called the *Flow deck* and the *AI-deck*.

Figure 8: Crazyflie 2.1 with no decks (source: Bitcraze)

The Flow deck enables motion detection in both horizontal and vertical directions which helps the drone to keep a stable flight and reduces long-term drift. It weights 1.6g and features an optical flow visual sensor that measures movements in relation to the ground, and a time-of-flight ranging module that provides a high-precision distance measurement to the ground [28].

4.2 AI-deck

The AI-deck is a printed circuit board developed by Bitcraze and an improvement of the PULP-Shield used for deploying DroNet on a Crazyflie 2.0 [6,7,11]. The AI-deck features the GAP8 multi-core System-on-Chip (SoC), designed for computing machine learning workloads [29], coupled with a low-power, QVGA monochrome camera, and an ESP32 WiFI microcontroller unit. The GAP8 embedded processor belongs to a new class of MCUs based on the PULP architecture paradigm and enables energy-efficient computations within the limited power envelope of a nano-sized drone [6]. It combines an eight-core parallel compute cluster and a single-core controller on the same System-on-Chip. The AI-deck has two Cortex-M 10-pin JTAG interfaces, one for the AI-deck and one for the ESP32, for programming each MCU.

The ESP32 enables image streaming and handling control over the drone via WiFi. However since the primary goal of this thesis is to develop a completely autonomous system where the detection and navigation run onboard the drone without any external computation or communication, the WiFi module is only used for debugging purposes during testing. The usage of WiFi also requires a higher power consumption, which further reduces the limited battery life.

The GAP8 SoC is a multi-core processor and an embodiment of the PULP architectural paradigm where the idea is to couple a traditional MCU for managing input-output-oriented tasks with a programmable general-purpose accelerator dedicated to executing data-parallel computations, inside the same System-on-Chip. The GAP8 is composed of one main RISC-V core called the fabric controller (FC), coupled with a programmable general-purpose accelerator called the cluster that is built up of eight RISC-V cores identical to the one in the fabric controller, all nine cores supporting the same extended RISC-V instruction set architecture.

Figure 9 shows an overview of the GAP8 architecture. All nine cores have access to a 512kB L2 memory which holds data and program code for both the fabric controller and the cluster. The FC is responsible for control and communication and has access to a 16kB L1 memory for data and a 1kB instruction cache. The eight cores in the parallel cluster are responsible for the computationally intensive task and share access to both a 64kB L1 memory and a 4kB instruction cache. The 64 kB shared L1 memory is fast with high bandwidth, but kept small in size due to its cost. The larger L2 memory has higher access latency and lower bandwidth compared to the L1. The processor also has access to an external optional L3 memory. The latency, access time, and bandwidth are even more

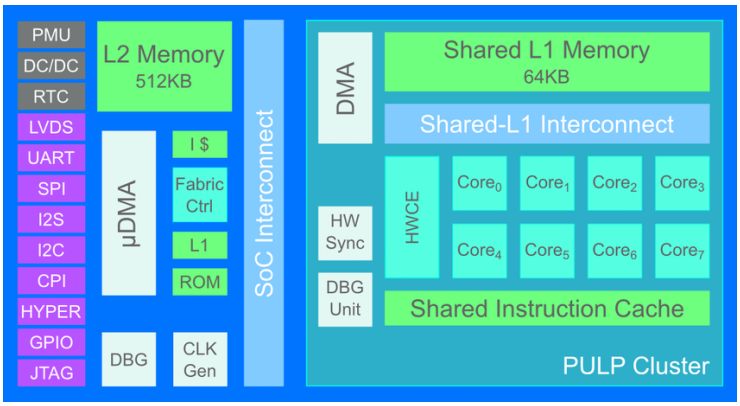


Figure 9: Architecture of the GAP8 embedded processor used on the AI-deck [7].

limited on the L3 memory, and access to L3 consumes more energy compared to the lower memory levels. The most effective use of the architecture is to try to keep data as close as possible to the cores using it, in the L1 memory.

There are three main challenges that need to be addressed when deploying a deep learning model on the GAP8 processor. Firstly, there is however no support for data caching between the different levels of memory on the GAP8 chip, so memory movements must be handled explicitly by the application code. The motivation for this is to keep the chip as small as possible and to avoid the power penalty associated with data caches [29]. Explicitly handling the memory movements in the hierarchical structure can reduce the energy spent on memory accesses given predictable data movements but it is then necessary to know where to store data to achieve a good performance.

The second challenge is that the processor does not have a floating-point unit. Common deep learning frameworks for training neural networks such as TensorFlow, PyTorch, and Caffe, use floating-point numbers to represent networks. While floating-point numbers can cover a wider range and provide higher accuracy, fixed-point arithmetic is more efficient and less energy-consuming compared to floating-point operations. To save resources, the GAP8 processor only supports 8-bit and 16-bit fixed-point operations. This means that a floating-point model needs to be quantized to either 8-bit or 16-bit fixed-point representation before being deployed on the GAP8. Quantization reduces the precision used to represent the parameters of a model, but makes it more lightweight and reduces the memory footprint. To preserve the quality of the model, it is necessary to make sure that the accuracy is not significantly degraded after quantization. Besides saving memory space, the quantization allows models to fully exploit the parallelism provided in the chip by leveraging the support for vectorization and Single Instruction Multiple Data (SIMD) instructions in the cluster, which can result in a significant speedup. But to do this, the processor needs to be provided with parallelized and vectorized code.

Finally, to fully exploit the eight-core cluster, parallel and vectorized code are needed. The manufacturer of the GAP8 processor, GreenWaves Technologies, provides a toolset called GAP*flow* to facilitate the process of running deep learning inference tasks on the parallel cluster. GAP*flow* consists of two main blocks, NNTool and AutoTiler, and is fed with a network graph of a trained model. The toolset takes care of the explicit memory management and quantization and produces parallel and vectorized code optimized to run on the eight-core cluster on the GAP8.

If the input model is not already quantized, it is possible to provide the NNTool with sample input images for computing a quantization calibration for fixed-point representation and converting the model. To minimize data movements during execution on the GAP8, NNTool can fuse together some of the network layers that operate locally on data, for example, a convolutional layer, a pooling layer, and a ReLu activation. The explicit memory management is automated by the AutoTiler. Fed with an optimized C-coded graph model generated by the NNTool, the AutoTiler computes an optimal partitioning of data and generates code for moving these partitions between different levels of memory available on the chip while running the network model efficiently on the eight-core cluster. The resulting code is then compiled and deployed on the GAP8.

4.3 Crazyflie Control system

The Crazyflie hardware platform comes with an open-source firmware with support for communication, stabilization, and low-level control of the quadcopter. The firmware is implemented in C and can be modified if needed. The firmware enables high-level control of both the position (x, y, z) and the attitude (roll, pitch, yaw) of the drone. These parameters can be controlled in either absolute values or in terms of velocity. Hence it is possible to give the Crazyflie instructions to fly towards a certain point or with a specific speed in a given direction. These instructions are given by defining setpoints, where either an absolute value or speed for both attitude and position values can be defined.

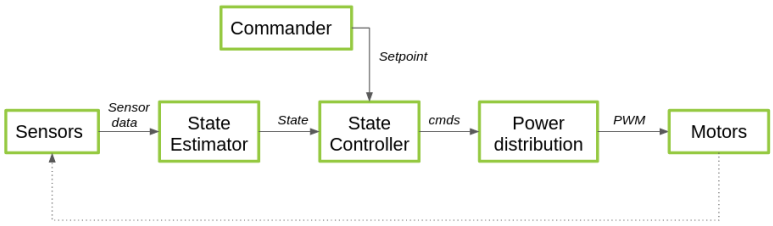


Figure 10: Overview of the stabilizer module (source: Bitcraze)

Figure 10 illustrates the path from sensor acquisition to motor control for the Crazyflie, all together collectively referred to as the stabilizer module. The

Crazyflie is dependent on data from onboard sensors to estimate its current state, which is used by the state controller to compute appropriate control commands to be executed by the motors.

The sensors that are used for state estimation in the configuration used in this thesis are the onboard accelerometer, gyroscope, and pressure sensor as well as the time-of-flight-sensor and optical flow sensor on the Flow deck. The accelerometer measures the drone's current acceleration in body-fixed coordinates, the gyroscope measures the angle rate (roll, pitch, yaw), the pressure sensor measures the air pressure, the time-of-flight sensor measures the distance to the ground, and the optical flow sensor measures movement relative to the ground. The sensor data is sent to the state estimator whose purpose is to calculate an estimate of the current state of the Crazyflie.

There are two different state estimators implemented in the Crazyflie firmware, called the complementary filter and the extended Kalman filter. The complementary filter is used when no relevant sensor data other than from the IMU is available, and the filter estimates the drone's attitude and altitude. The more complex extended Kalman filter accepts sensor input from both internal and external sensors. In this hardware configuration used for this thesis, the only additional data that is available is from the Flow deck, but the Kalman filter can accept data from other decks not used here as well. With the additional data from the Flow deck, the Kalman filter can provide information for full pose estimation, including attitude, position, and velocity.

The goal of the state controller is to move the drone into a new position based on a setpoint. The default controller in the Crazyflie firmware uses a cascade proportional-integral-derivative (PID) controller with four PID controllers. A setpoint is sent to either a position or velocity controller that determines the desired pitch and roll angles. These values are then fed to an attitude controller resulting in desired angle rates. These are in turn sent to an angle rate controller for determining appropriate motor commands.

5 Method

This thesis proposes a solution for object detection and following using two separate modules; one for object detection and one for controlling the drone. The control module runs directly on the STM32 processor on the Crazyflie while the object detection module is implemented on the GAP8 on the AI-deck. Both modules are implemented in C.

The Crazyflie firmware provided by Bitcraze only supports communication from the AI-deck to the STM32 one byte at a time. The firmware is modified with an implementation of a communication protocol between the GAP8 on the AI-deck and the STM32 for a safe communication of bounding box coordinates from the detection stage to the control system. The modification supports the communication of four values packed together to safely communicate the position and size of a bounding box to the STM32 from the GAP8.

5.1 SSD-MobileNet for Object Detection

The task of object detection and following requires a detection method that is capable of real-time execution on the limited resources available on the Crazyflie 2.1 extended with the AI-deck, while still achieving accuracy on a satisfactory level. To exploit the effective execution of deep neural networks provided by the GAP8, an SSD-MobileNet model trained to detect human bodies is selected. The model architecture is supported by the AutoTiler and is shown to work on the hardware platform [24].

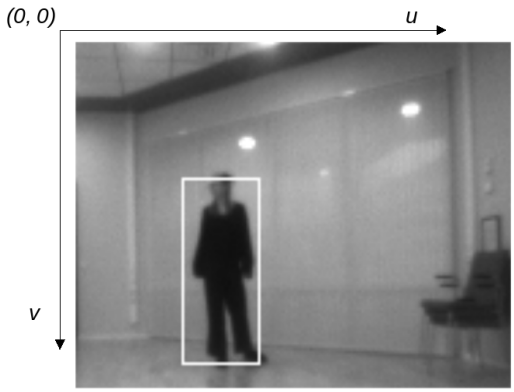


Figure 11: Image coordinate system *(u, v)* with origo in the image's upper left corner.

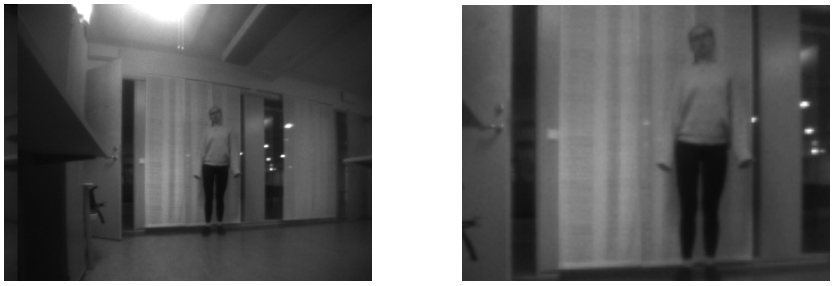
The selected object detection model accepts images with a resolution of 160 x 120 pixels as input and results in one or more bounding boxes for every detected body in the image, each with an associated score as a measure of how

sure the model is that the detected object is a human body. The bounding box coordinates are given by the detection model in the form:

$$b = [u_{bb}, v_{bb}, w_{bb}, h_{bb}]$$

where u_{bb} and v_{bb} represent the top left position of the bounding box in image coordinates and w_{bb} and h_{bb} the pixel width and height of the bounding box respectively. The image coordinate system is illustrated in Figure 11.

The Himax QVGA camera on the AI-deck captures images with a resolution of 320 x 240 pixels. Images must hence be cropped to 160 x 120 pixels before being fed to the object detection network. The cropped region is selected to be the middle part of the image, shifted 20 pixels upwards, which manages to capture humans at a few meters distance from the drone when flying at a height of half a meter. An alternative would be to use a lower resolution when capturing the images, but this is not supported by the software. Another option would be to use a detection model that supports a higher resolution or to train a new model accepting higher resolution images but that is beyond the scope of this thesis. The cropping is illustrated in Figure 12, showing an image captured in QVGA resolution and its cropped version.



(a) Full resolution image, 320 x 240 pixels (b) Cropped image, 160 x 120 pixels

Figure 12: A full resolution image compared with a cropped version.

The pre-trained SSD-MobileNet model is fed to the NNTool together with a set of sample images for calibration of the quantization. NNTool converts the model into a 16-bit fixed-point representation and fuses together layers where it is possible. The resulting C-coded graph model is converted by the AutoTiler, which generates code for running inference on the detection model, optimized to be deployed on the GAP8 processor. The GAP8 processor on the AI-deck is programmed with an Olimex ARM-USB-TINY-H debugger with an Olimex ARM-JTAG-20-10-pin adapter to connect the debugger with the deck.

5.2 Control

The objective of the tracking algorithm is to control the drone in such a way that the difference between a detected target's bounding box and an ideal bounding box is minimized. By knowing the exact location of a target in captured images, the goal is to always keep the target at the desired position in the camera's field of view. The only information available for control is the relative position of the target in the form of bounding box coordinates. This information needs to be translated into a speed and direction to fly in to meet the control objective.

The coordinate system used by Bitcraze and in the *Crazyflie firmware* is illustrated in Figure 13. Both the global coordinate system (X, Y, Z) and the body-fixed coordinate system (x, y, z) follow the east, north, up (ENU) convention. A positive value for the speed in x corresponds to a forward motion of the drone and a positive value for the speed in y results in a motion to the left from the drone's perspective. The altitude angles for pitch, roll, and yaw are also included in the image.

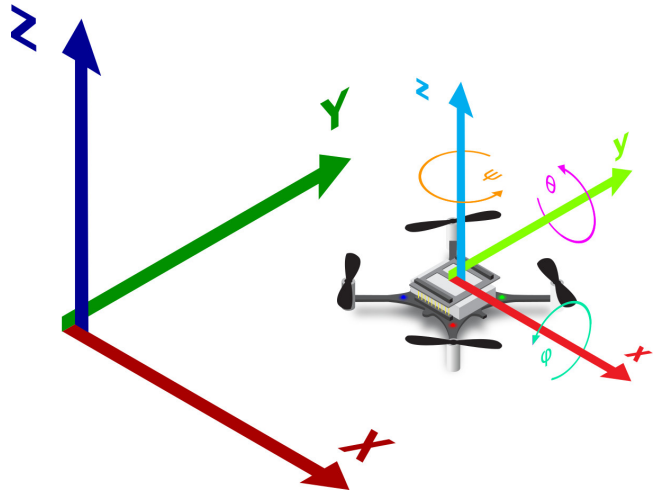


Figure 13: Global coordinate system (X, Y, Z) and local coordinate system in body fixed coordinates (x, y, z) for the Crazyflie.

The stabilization module with an extended Kalman filter, implemented in the Crazyflie firmware as explained in Section 4.3, allows a stable flight while feeding the system with high-level commands in the form of setpoints. The system is provided with an application code for translating bounding box coordinates in an image into appropriate setpoints for the state controller. The application code hence replaces the *Commander*-box in the illustration of the stabilization module in Figure 10. This creates a sensor feedback control loop that only relies on data from onboard sensors for an autonomous flight. Three separate proportional controllers are implemented to calculate setpoint values: one for the forward velocity, one for the lateral velocity, and one for the yaw rate.

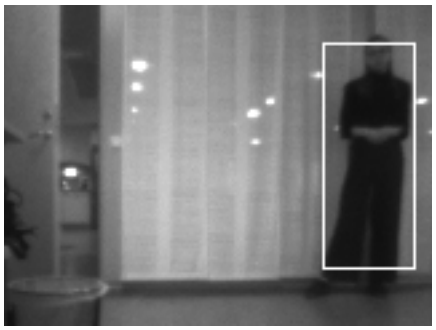


Figure 14: Image captured by the camera on the AI-deck showing a detected object, in this case a human body, appearing to the right in the field of view of the drone.

This thesis only considers motions in the horizontal plane and a controller for adjusting the altitude is therefore not implemented. For an application with an object that changes altitude, for example when following another drone, a controller for the altitude would be needed. The velocities in the setpoints are represented in meters per second, and the attitude rates in radians per second.

5.2.1 Lateral motion

The control objective is to minimize the difference between values from a bounding box given by the detection algorithm and the values from an ideal bounding box, by controlling how the drone flies. If a target appears to be on the side in an image captured by the fixed front-facing camera, as the example in Figure 14, the drone should move in such a way that the target will be centered in its field of view. A reasonable choice as a control objective to achieve this, is to minimize the horizontal difference between the center of an image and the center of the detected target's bounding box. There are two possible ways to control the drone to minimize the horizontal error, either by flying to the side or rotating (yaw). If necessary, the drone can then also move forwards or backward to reach a desired distance from the target.

A simple approach is to either only adjust the lateral position or only adjust the yaw, depending on what is most suitable for the application. In this thesis, a target is detected in images captured by a front-facing camera. One assumption that can be made from this is that if a target can be identified in an image and the drone can 'see' the target, there is a free area with no obstacles in the line of sight between them. In that case, a safe choice would be to only adjust the yaw instead of flying sideways, since the latter might risk the drone crashing into an unseen obstacle. However, there might be other applications where it is desirable to follow a moving object in a perspective from the side. If using lateral motion to control the drone, it could be smart to use some additional type of sensors for detecting possible obstacles that are not within the field of

view of the camera.

Based on the control objective, to minimize the horizontal error between the center of the bounding box giving the target's position in the image and the center of the image, two separate controllers are implemented, one for speed in the y-direction and one for the yaw rate. Let (u_0, v_0) denote the center of the image in the camera coordinate system. The center of the bounding box is then calculated as

$$u_c = u_{bb} + w_{bb}/2,$$

and the error to be minimized as

$$e_u(t) = u_0 - u_c(t).$$

The same error value is used as input for both the lateral motion controller and the yaw rate controller,

$$u_y(t) = K_{p_x} \cdot e_u(t)$$
$$u_{yaw}(t) = K_{p_{yaw}} \cdot e_u(t)$$

The gains are selected by trial-and-error testing with increasing values until a point where the drone moves too far or too much. The uncertainties in the detection made it hard to properly test the controller, and therefore only a simple, proportional controller is used.

5.2.2 Forward motion

To control the forward and backward motion it is desirable to know the distance between a detected target and the drone. One possibility is to transform the 2D information about the target's position given by the bounding box into a 3D space. If the drone was equipped with two cameras, stereo vision could be used to compute the distance [1]. Stereo vision could also be simulated by comparing bounding boxes in images from two different points in time, combined with knowledge about the distance traveled by the drone between these two points. The Crazyflie 2.1 is equipped with sensors that provide data that can be used for estimating the traveled distance. The estimate is however not very accurate. Combined with uncertainty and fluctuations in bounding box sizes, this approach is not suitable for this application. In [30] the problem of calculating the distance to a detected target is solved with a reinforcement learning algorithm that learns an appropriate velocity for a given ratio between a detected and ideal bounding box.

Assuming there is a desired size and position for the target's bounding box, the goal is to strive the detected bounding box towards the ideal one. The ratio between a desired and detected bounding box can be computed and should be one when the drone is at a desired distance from the target. If the detected bounding box appears smaller than the desired one, the drone should fly forward

to correct this, and similarly fly backward if the detected bounding box is larger than desired. The ratio between the detected and desired bounding box together with the exact height of the target and trigonometric calculations can be used to compute the distance needed to be traveled by the drone to reach the desired distance from the target. However, the exact height of the target might not be known, and there can also be variations in the sizes of detected bounding boxes which makes this approach less reliable.

Another alternative is to directly rely on the bounding box size and position in the image as input for controlling movements of the drone [22] [23], instead of trying to estimate the distance to a target. The difference between a detected bounding box and a given set point can be used directly as input to compute a setpoint for the drone's forward motion. Here, the height of the bounding box is directly used by the proportional controller to calculate a setpoint value for the forward velocity.

Let h_0 denote the desired height of the bounding box. The error to minimize is then given as

$$e_h(t) = h_0 - h_{bb}(t),$$

and the proportional controller with gain K_{p_h} is given by

$$u_x(t) = K_{p_h} \cdot e_h(t).$$

The value of h_0 needs to be adjusted depending on the image resolution and the type of object to be detected. In this case, the target is human bodies, and images with a resolution of 120 x 160 pixels are used. The desired height of the bounding box is set to be 100 pixels, $h_0 = 100$.

5.3 Implementation

The hardware configuration requires two separate programs: one for the GAP8 embedded processor on the AI-deck, and one for the STM32 MCU on the Crazyflie 2.1. The communication protocol between the AI-deck and the STM32 provided by the *Crazyflie firmware* [Footnote version] only supports one byte at a time to be sent from the GAP8 to the STM32. The firmware[1] is therefore updated with support for sending packages of data containing bounding box coordinates from the GAP8 to the STM32. The implementation is inspired by an application for streaming images[2] but modified to fit the purpose of this application.

In the code implementation for object detection, all bounding boxes that the detection model categorizes as human bodies are traversed, and boxes with sizes smaller than a threshold are discarded to avoid this type of false positive. The

[1] Version 2021.06, accessed in October 2021. The firmware has since then been updated with support for better communication between the AI-deck and the STM32.

[2] `https://github.com/jeguzzi/ai_sail/`

last box with reasonable values is selected as input to the control algorithm. For this application, the threshold is set to a bounding box size of 30×40 pixels. If the detection model identifies two human bodies in the image, both bigger than the threshold, only one of them will be selected as input to the control algorithm.

6 Results

6.1 Experimental Setup

To validate the implementation, the motion of the Crazyflie 2.1 is recorded while flying in front of a human target moving in a pre-defined way. The hardware is extended with an additional deck called multi-ranger, equipped with five ToF sensors of the same type as on the Flow deck. The multi-ranger deck gives the Crazyflie the ability to measure the distance to solid objects appearing in front of, behind, to the left, to the right, and above the drone. The distance is measured with a millimeter precision of up to 4 meters, depending on surface and light conditions [31]. Measurements are conducted with the AI-deck mounted on top of the main drone platform, and the Flow deck and multi-ranger deck under the platform with the Flow deck furthest down, as in Figure 15. Measurements are conducted by placing the Crazyflie 2.1 in a corner with 1 meter distance to the wall behind and to the left of the drone before startup. An indication of its movements is given by measuring the distance to the wall behind and to the left, recording how these distances change when the drone flies.

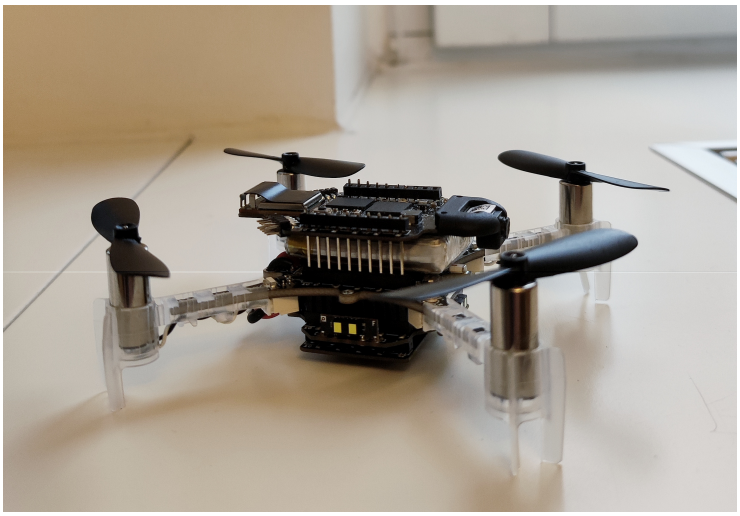


Figure 15: Crazyflie 2.1 equipped with a battery, the AI-deck mounted on top, and Flow deck plus multi-ranger deck mounted under the main platform.

After startup, a human steps in front of the drone. The drone is then allowed to stabilize its position before, at a given point in time, the human takes a step one meter to the side or one meter backward. The drone then flies until its movements stabilize again. Distance measurements from the multi-ranger deck as well as indications of whether detection was made or not are transferred over radio to a computer during the flight. This set-up is used to test the controllers for lateral and forward motion, but because of difficulties in accurately measuring the yaw of the drone, no test result for the yaw controller is presented.

6.2 Results from Step Response

The lateral controller is tested with $K_{P_x} = 0.05$. Figure 16 illustrates the lateral motion of the drone as the distance to a wall on its left measured by the multi-ranger deck for four different test runs, all recorded under the same conditions. The red areas in the graphs represent periods when the deployed SSD-MobileNet model successfully can detect the human in front of the drone. Figures 16(a) and 16(b) show two examples when the detection algorithm continuously manages to detect the target while Figures 16(c) and 16(d) give examples when the detection algorithm does not succeed in continuous detection. All examples show that the measured distance to the wall increases by approximately one meter after the human takes a step one meter to the side. At the time of the step response, visualized as a green line, the person standing in front of the drone takes a step one meter to its left. Figure 16(a) shows that the drone overshoots but can stabilize after around 20 seconds. The example in Figure 16(b) suggests that the drone oscillates in front of the target both before and after the person has moved. Figures 16(c) and 16(d) shows that the distance to the wall increases after the step when the detection algorithm succeeds but also indicates that the drone hovers in the same position in the air when the detection fails.

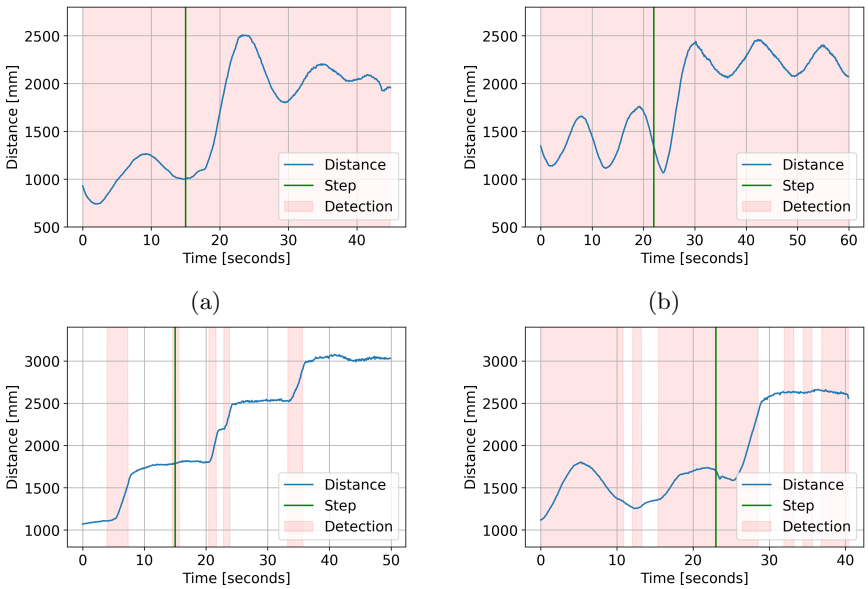


Figure 16: Examples of step responses for lateral motion when a target move one meter to the side.

The controller for forward velocity is tested with $K_{p_h} = 0.01$ and an ideal bounding box height in pixels $h_0 = 100$. Figure 17 shows the measured distance to the wall behind the drone, where the green line indicates the point in time when a person in front of the drone takes a step one meter backward away from the drone.

The detection algorithm does not manage to continuously detect the target in any of the test runs. However, when a detection is made the drone adjusts its position in the desired way, moving forward. All examples show that the drone loses detection after some forward motion towards the target. One possible reason for this could be that the drone comes too close to the target to successfully detect it.

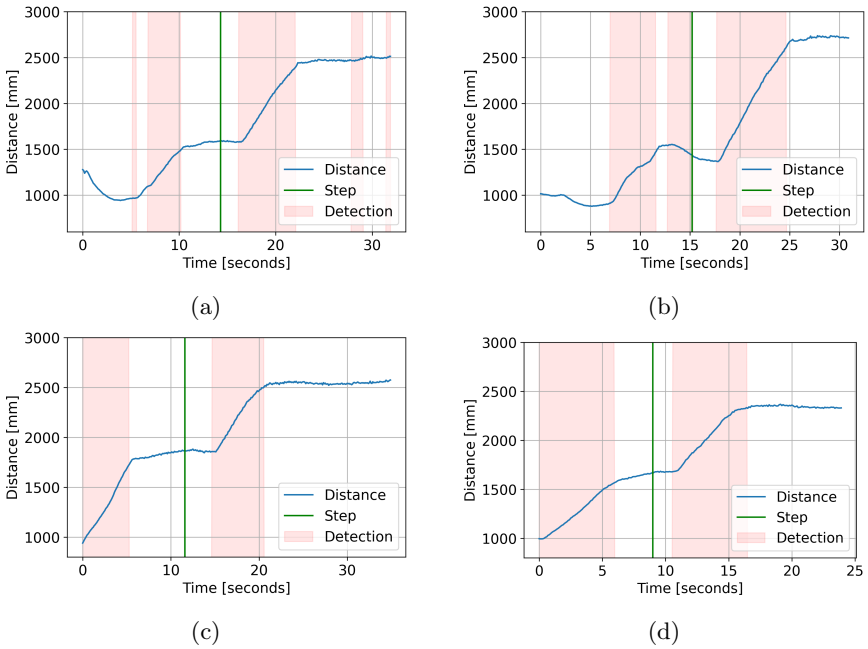


Figure 17: Examples of step responses for the drone's forward motion when a target move one meter away from the drone.

6.3 Performance Measurements

The SSD-MobileNet object detection algorithm achieved a frame rate of 0.85 FPS. The total flying time with a fully charged battery reached 5 minutes and 30 seconds, carrying the AI-deck and the Flow deck. From the examples in Figure 16 and Figure 17 it can be seen that it takes around 10 seconds from when the target moves until the drone has adjusted its position. For safety reasons, the maximum velocity is set to $0.5m/s$ which implies that it will take at least two seconds for the drone to adjust its position within one meter. From the slope in the graphs in Figure 16 and Figure 17 it can be seen that the speed of the drone is often lower than the maximum $0.5m/s$, and the speed can be increased by increasing the values of the proportional controllers K_{p_x} and K_{p_h}.

7 Discussion

7.1 Limitations

The performance of the detection model is perhaps the most severe limitation of this implementation. The detection is not always accurate, and as seen in the graph in Figure 16 and Figure 17 it often isn't able to detect a target standing in front of the drone. The low FPS also affects the control and can be one explanation for the oscillating behavior in Figure 17(a), since it takes over a second to analyze an image. The poor performance of the detection model also makes it hard to properly test and evaluate different strategies for controlling the drone.

There are also some limitations to the selected method for measuring the movements of the drone. Even though the drone was placed at the same spot on the floor for each run, drift, and movements occurring during startup made the initial position in the air slightly different for each test run. The movements during startup also included some rotation, or yaw, resulting in the drone facing a different direction for each run. This in turn affects the measured distance; the wall behind the drone will appear further away if the ToF sensor is facing backward or the side is not perpendicular to the wall. Taking this into consideration, the presented distance data in Figures 16 and 17 should not be taken as exact, but rather as an indication of the drone's movements.

Using an external positioning system for both the drone and the target could improve the measurements of the drone's movements, and the motion relative to the target. This would also overcome the differences in starting position and allow to map trajectories of the drone and the target to visualize the movements.

Another limitation of the detection model is that the size of the bounding box can fluctuate, even if the target is at the same distance from the camera. This can cause a problem when using the height of the bounding box to control the forward motion. Sometimes the bounding box surrounds the whole person, but other times a part of the legs or head can be cut off, making the control algorithm think that the distance between drone and target has changed. In Figure 18, two persons standing at the same distance to the drone are detected, but the height of the bounding boxes differs between them. The same thing can also happen to a single person in two different frames. If the detection model would run faster, it could be possible to use a filtered version or an average of the height from several detections to minimize the effect of these fluctuations.

Another problem with the detection model is that it sometimes wrongly classifies parts of the background to be a human body and an example of this is shown in Figure 19. Given the low image resolution and the size of a human in relation to the field of view, bounding boxes with a small size with respect to the resolution can be discarded as false positives, and these are therefore not considered correct classifications in the code implementation. The program running on the AI-deck iterates over all detections in an image and disregards all boxes with a size

Figure 18: 160×120 image captured by the camera on the AI-deck, showing two people where both are detected and identified as human bodies. The legs of the person to the left are cropped out by the detection algorithm whereas the bounding box surrounding the person to the right captures the full body.

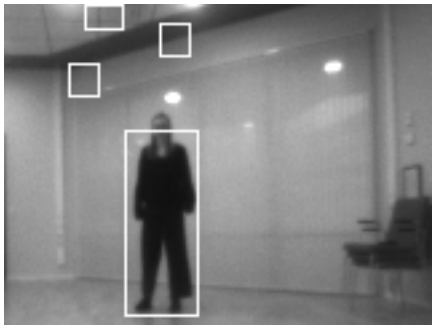


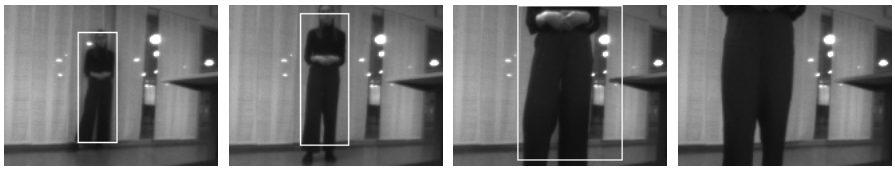
Figure 19: Example when the detection model successfully detects a human present in the image, but also missclassifies parts of the background to be human bodies.

smaller than a threshold which for this application is set to be 30×40 pixels. The last detection with values larger than the threshold is kept and sent to the control module on the STM32.

The graphs in Figure 17 show that the drone loses detection after flying towards the target, both after the first initialization and after the human takes a step. One reason for the drone to lose detection when flying towards a target is that it might come too close to be able to detect it. The images in Figure 20 show an example of this, with a person stepping closer and closer to the camera. When coming too close, a large part of the body is not visible to the camera and the detection model cannot identify the human body anymore. This is a problem when detecting objects that are relatively large compared to the field of view such as a human body, that in this case can take up the full height of an image when being close to the drone. Worth noticing is that the detection model still manages to detect the human body even if the head and feet are cut off.

A challenge with the setup of the camera on the AI-deck is that it is fixed, only looking straightforward. To gain knowledge about the environment in every direction the drone has to rotate or fly around, but it is difficult to receive visual information directly above or below it. The drone needs to move around to avoid having a large blind spot, but with the risk of losing sight of a detected object. Targets to the sides or behind could easily be missed unless a good search algorithm is implemented to start after a certain time without any detection.

Another consequence of using a fixed camera is that the camera angle relative to the ground will change with the motion of the drone. A quadcopter will tilt slightly in the direction it travels in, tilting more with a higher velocity. If an object detection algorithm is used as in this thesis, the detection model needs to be able to handle cases with slightly rotated objects. This can be solved by including rotated images in the training dataset. When conducting experiments as explained in Section 6.1, the speed was deliberately kept low for safety reasons, resulting in a negligible effect from the tilt.



(a) (b) (c) (d)

Figure 20: Human target getting closer and closer to a camera. The detection is lost when the target comes too close to the camera in Figure 20(d).

7.2 Future Work

To increase the performance of the detection model, several strategies can be applied. By using 8-bit quantization instead of 16-bit quantization, the time for analyzing each image could be decreased, thanks to the support for vectorization in the GAP8 processor. The SSD part of the object detection model could also be replaced with a lighter version called SSDLite [32] with fever computations to decrease the inference time. However, when implementing these types of optimizations with respect to time it is important to ensure that the accuracy of the model is not compromised.

Sometimes, the detection model fails to detect a target in a captured image, and sometimes it miss-classifies parts of the background as human bodies. One way to improve the detection accuracy could be to fine-tune the model on a dataset relevant to the application, for example, captured by the same type of camera as on the AI-deck, or retrain the model including the new dataset. The latter has shown to increase the accuracy when implementing DroNet [5] on a Crazyflie with an AI-deck [7].

The implementation presented in this thesis only considers movements in the horizontal plane but could be extended to include vertical movements as well by including a controller for the altitude. A possible objective for that controller would then be to minimize the vertical difference between a certain point in the image and the center of the bounding box by adjusting the thrust. In this case, the tilt of the drone due to its velocity might need to be taken into consideration when implementing a control algorithm.

A general improvement to the approach used in this thesis would be to incorporate algorithms for collision avoidance. For example, the multi-ranger deck, here only used for collecting flight data, could be used to detect nearby solid objects, and this information could be included in the control algorithm to avoid crashing into these obstacles. One approach is to use the sensor data for SLAM to create a map of the environment, but as previously mentioned, this is quite expensive in terms of resources and therefore a challenge in the setting of nano-sized drones. A more lightweight approach would be to use the data to simply stop or change direction if an obstacle is detected within the flying range.

If more than one person is within the field of view of the drone, as shown in Figure 18, the implemented detection module could jump from selecting one person as a target in one frame to the other person in another frame. To allow the drone to keep following the same person, the detection could be combined with a tracking algorithm for following an object in an image stream such as KCF [33] or MDNet [34]. A tracker can also be helpful to keep moving and follow an object even if the detection model fails in a few single images. A tracker generally does not update the dimensions of the bounding box, and if the size is used to control the velocity, the tracker needs to be combined with frequent and continuous detections for high reliability.

Swarms are an active research topic in the field of autonomous UAVs, where several drones collaborate on a task. In the use case of object detection, nodes in a swarm can share information about the position of the object of interest and work together for a better and more accurate prediction of its position. The detection algorithm outputs a score of how sure it is that it has detected an object, and by communicating the positions and scores from detection the nodes could get a clearer view of if and where objects are detected. If some drones would lose the target within their field of view they could get information about its position from other drones and use this to plan their trajectory. When working with small, low-cost, and energy-efficient drones, there needs to be a trade-off between accuracy and speed. Some of the traded accuracies could be compensated by combining the result of several detections from different nodes. A swarm could also possibly solve the problem of several objects of interest appearing within the field of view of the drones, where the swarm could split up, some following each object.

Another approach that can be explored in the future is to learn a controller instead of hand-designing one. One of the biggest obstacles to this approach is the collection of ground truth values for control commands. These could

for example be given by recording an expert pilot or collected by a designed controller that is given ground truth values of the object's position. A controller learned from a human pilot might find patterns in how to best control the drone that is hard to mimic with a hand-designed controller. Taking this one step further, an interesting approach is to use an end-to-end approach such as DroNet but for the task of object following such as in [25] with all computations running onboard the Crazyflie nano-drone platform with the GAP8 processor.

8 Conclusion

This thesis presents an implementation of an object following algorithm using bounding box object detection and simple proportional controllers, running all computations onboard the palm-sized Crazyflie 2.1 quadcopter. The detection model, SSD-MobileNet, provides almost real-time detection in images captured by a low-power grayscale image sensor on a GAP8 embedded processor that is specifically designed for machine learning workloads in low-power settings. Completely relying on onboard computations gives the platform an ability to operate in conditions not restricted by latency or power cost from wireless communication

Test results of the implementation show that the system is able to detect a target and follow simple movements, but the reliability and speed of the detection algorithm need to be improved in order to achieve a satisfactory result. The system does not, however, ensure reliable tracking without a change of target. This is mostly due to the performance of the detection model, which in turn affected the ability to develop good controllers. The implementation reaches 0.85 FPS which results in it taking over a second for a new detection to be made if a previous one fails, making it likely to lose track of the target. The slow and sometimes inaccurate detection also affects the ability to evaluate controllers for the motion of the drone.

Using a computationally lighter object detection model, trained on a custom dataset, and optimized for the available hardware could increase the speed and accuracy of the detection model, which in turn would enable the design of refined controllers. This thesis scratches on the surface of the possibilities of autonomous vision-based control of nano-sized UAVs, an area that can grow in line with the development of more efficient hardware and optimized software.

References

[1] Petar Durdevic and Daniel Ortiz-Arroyo. A Deep Neural Network Sensor for Visual Servoing in 3D Spaces. *Sensors*, 20(5), 2020.

[2] Javier Burgués, Victor Hernández, Achim J. Lilienthal, and Santiago Marco. Smelling Nano Aerial Vehicle for Gas Source Localization and Mapping. *Sensors*, 19(3), 2019.

[3] Patrick P. Neumann, Paul Hirschberger, Zhandos Baurzhan, Carlo Tiebe, Michael Hofmann, Dino Hüllmann, and Matthias Bartholmai. Indoor Air Quality Monitoring using flying Nanobots: Design and Experimental Study. In *2019 IEEE International Symposium on Olfaction and Electronic Nose (ISOEN)*, pages 1–3, 2019.

[4] Lichao Xu, Vineet R. Kamat, and Carol C. Menassa. Automatic extraction of 1D barcodes from video scans for drone-assisted inventory management in warehousing applications. *International Journal of Logistics Research and Applications*, 21(3):243–258, 2018.

[5] Antonio Loquercio, Ana I. Maqueda, Carlos R. del Blanco, and Davide Scaramuzza. DroNet: Learning to Fly by Driving. *IEEE Robotics and Automation Letters*, 3(2):1088–1095, 2018.

[6] Daniele Palossi, Nicky Zimmerman, Alessio Burrello, Francesco Conti, Hanna Müller, Luca Maria Gambardella, Luca Benini, Alessandro Giusti, and Jérôme Guzzi. Fully Onboard AI-powered Human-Drone Pose Estimation on Ultra-low Power Autonomous Flying Nano-UAVs. *IEEE Internet of Things Journal*, pages 1–1, 2021.

[7] Daniele Palossi, Antonio Loquercio, Francesco Conti, Eric Flamand, Davide Scaramuzza, and Luca Benini. A 64-mW DNN-Based Visual Navigation Engine for Autonomous Nano-Drones. *IEEE Internet of Things Journal*, 6(5):8357–8371, 2019.

[8] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR*, 2017.

[9] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR*, 2016.

[10] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. *CoRR*, 2017.

[11] Daniele Palossi, Francesco Conti, and Luca Benini. An Open Source and Open Hardware Deep Learning-Powered Visual Navigation Engine for Au-

tonomous Nano-UAVs. In *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 604–611, 2019.

[12] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, and Thomas B. Schön. *Machine Learning - A First Course for Engineers and Scientists*. Cambridge University Press, 2022.

[13] Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep Learning vs. Traditional Computer Vision. In *Advances in Computer Vision*, pages 128–144, 2020.

[14] Ivan V. Saetchnikov, Elina A. Tcherniavskaia, and Victor V. Skakun. Object Detection for Unmanned Aerial Vehicle Camera via Convolutional Neural Networks. *IEEE Journal on Miniaturization for Air and Space Systems*, 2(2):98–103, 2021.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, 2012.

[16] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[17] Ross Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[18] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[19] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[20] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[21] Shuying Liu and Weihong Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015.

[22] Philippe Martin Wyder, Yan-Song Chen, Adrian J. Lasrado, Rafael J. Pelles, Robert Kwiatkowski, Edith O. A. Comas, Richard Kennedy, Arjun

Mangla, Zixi Huang, Xiaotian Hu, Zhiyao Xiong, Tomer Aharoni, Tzu-Chan Chuang, and Hod Lipson. Autonomous drone hunter operating by deep learning and all-onboard computations in GPS-denied environments. *PLOS ONE*, 14(11):1–18, 11 2019.

[23] Jake Gemerek, Silvia Ferrari, Brian H. Wang, and Mark E. Campbell. Video-guided Camera Control for Target Tracking and Following. *IFAC-PapersOnLine*, 51(34):176–183, 2019.

[24] Lorenzo Lamberti, Manuele Rusci, Marco Fariselli, Francesco Paci, and Luca Benini. Low-Power License Plate Detection and Recognition on a RISC-V Multi-Core MCU-Based Vision System. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[25] Dario Mantegazza, Jérôme Guzzi, Luca Maria Gambardella, and Alessandro Giusti. Vision-based Control of a Quadrotor in User Proximity: Mediated vs End-to-End Learning Approaches. *2019 IEEE International Conference on Robotics and Automation (ICRA)*, May 2019.

[26] Shai Shalev-Shwartz and Amnon Shashua. On the Sample Complexity of End-to-end Training vs. Semantic Abstraction Training. *CoRR*, abs/1604.06915, 2016.

[27] Bitcraze AB. *Datasheet Crazyflie 2.1*, 2021.

[28] Bitcraze AB. *Datasheet Flow deck v2*, 2020.

[29] Eric Flamand, Davide Rossi, Francesco Conti, Igor Loi, Antonio Pullini, Florent Rotenberg, and Luca Benini. GAP8: A RISC-V SoC for AI at the Edge of the IoT. In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 1–4, 2018.

[30] Roman Barták and Adam Vykovský. Any Object Tracking and Following by a Flying Drone. In *2015 Fourteenth Mexican International Conference on Artificial Intelligence (MICAI)*, pages 35–41, 2015.

[31] Bitcraze AB. *Datasheet Multi ranger deck*, 2020.

[32] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018.

[33] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(03):583–596, 2015.

[34] H. Nam and B. Han. Learning Multi-domain Convolutional Neural Networks for Visual Tracking. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4293–4302, 2016.