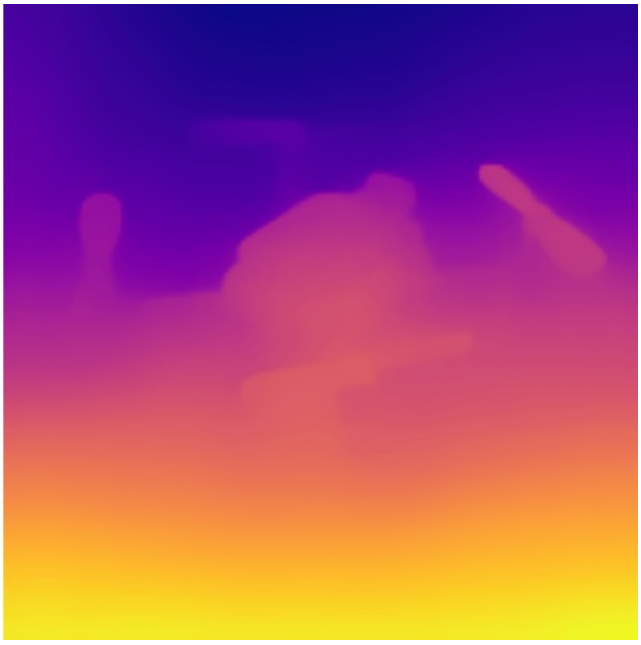
# Monocular Depth Estimation for Light-Weight Real-Time Obstacle Avoidance



Niels Sombekke

Layout: typeset by the author using LaTeX.
Cover illustration: Bitraze trademark & Niels

# Monocular Depth Estimation for Light-Weight Real-Time Obstacle Avoidance

Niels Sombekke
12685739

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

*Supervisor*
Dr. A. Visser

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

July $1^{st}$, 2022

**Abstract**

Autonomous nano-drones are becoming increasingly popular. Their light weight and small size provide major advantages in safety and maneuverability, being able to navigate through dense crowds or narrow spaces. These properties make nano-drones promising for indoor environments where they can be used in areas inaccessible or dangerous to humans. To push the state-of-the-art for the autonomy of nano-drones the International Micro Air Vehicle (IMAV) organizes the Nanocopter AI Challenge. In this competition teams need to fly a nano-drone through an obstacle course. The main challenge is working with the limited hardware of the nano-drone, being equipped with a monocular gray-scale camera and a small neural network processor. This work covers the obstacle avoidance aspect of the competition. Using recent developments in monocular depth estimation (MDE) for embedded devices we develop a real-time light-weight control algorithm. The most optimal steering angle is extracted by applying a horizontal strip of average bins and three criteria to the generated depth map. We compare the influence on performance of using a metric or inverse depth map by testing and evaluating two light-weight MDE models in a simulated environment. Simultaneously, the influence of the number of average bins is tested. The best performing control algorithm, using an inverse depth map and 5 bins, is able to reach the goal at least 80% of the time in three increasingly challenging stages.

# Contents

# Chapter 1

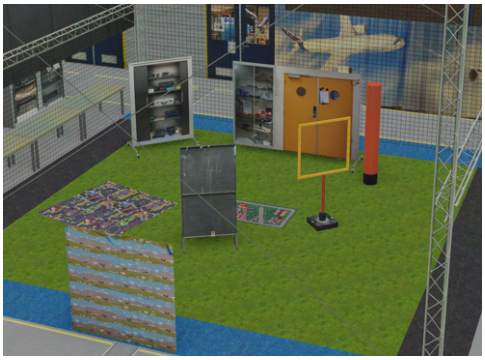
# Introduction

There is a surge of interest in the development and usage of autonomous nano-drones [1]. These drones are thanks to their light weight able to safely operate near humans whilst their small size allows for navigation through narrow spaces. Both these properties make them promising for indoor environments where they, for instance, can be used for mapping, surveillance, finding gas-leaks or assisting in search and rescue missions by exploring areas inaccessible to humans. However they also impose new challenges when implementing artificial intelligence for autonomous flight. Large and average-size drones have the power available to exploit high-end computational devices, but this is not a feasible option for the nano-drones since they can only carry and power a few sensors, memory, and processing power onboard. Autonomy of nano-drones is thus a challenge yet to be solved, one of the key aspects of autonomous navigation is having a reliable algorithm for obstacle avoidance. Therefore, the main aim of this thesis is implementing an obstacle avoidance algorithm while being constrained by the hardware limitations of the nano-drone. Research is done on the difference between using depth and disparity maps and corresponding resolutions for the control algorithm.

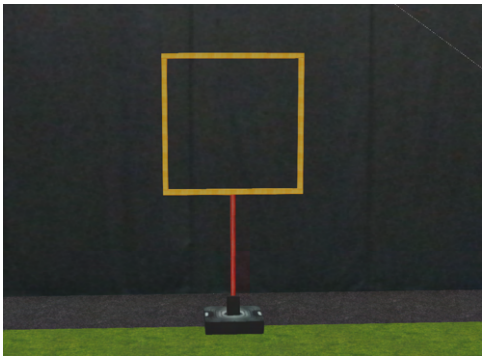
## 1.1 Nanocopter AI Challenge

To push the state-of-the-art for the autonomy of drones the International Micro Air Vehicle (IMAV)[1] organizes a conference and multiple competitions every year. This year it will be held in Delft, The Netherlands, from 12 to 16 September 2022.[2] One of these competitions is the Nanocopter AI Challenge wherein teams are challenged to let a nano-drone fly as fast as possible through an obstacle course, as seen in figure 1.1. The nano-drone is equipped with a single-camera

[1] http://www.imavs.org/
[2] https://2022.imavs.org/



(a) Cyberzoo



(b) Yellow gate

Figure 1.1: Challenge environment

and a small neural network processor. The score is mainly determined by distance flown within the allocated time, but extra points can be gained when the drone successfully passes through one of the yellow gates. The location, shape and type of obstacle are unknown. Obstacles can be moved during the run but never directly in front of the nano-drone.

An reactive obstacle avoidance algorithm thus has to be implemented which is able to generalize well on different kind of environments, next to that a path planning solution should be created to navigate the obstacle course as efficient as possible. The UvA Drone team, consisting of two bachelor students and supported by one staff member of the University of Amsterdam, will participate in this competition and work on these challenges.

## 1.2 Platform

The Nanocopter AI Challenge requires all participating teams to use identical hardware which consists of a Crazyflie 2.1 and an AI Deck 1.1, both produced by Bitcraze. Bitcraze is a company that develops and manufactures nano-drones for research and education, they provide an open-source ecosystem with multiple expansion decks and development tools. The Crazyflie 2.1 is an open-source robotic development platform that, with its size of 92x92x29mm and takeoff weight of 27g, fits in the palm of your hand. It supports multiple expansion decks with automatic detection. One of these expansion decks is the AI Deck 1.1, which is inspired by the PULP-Shield PCB design as presented by ETH Zurich [2]. The deck is a processing board built around the GAP8 RISC-V multi-core microcontroller unit (MCU) for artificial intelligence purposes. In addition, there is an ultra low power (ULP) 320x320 gray-scale monocular camera from Himax and an ESP32 for WiFi

(a) Crazyflie 2.1 (b) AI Deck 1.1 (c) Assembled

Figure 1.2: Platform used

connectivity. It is designed to be mounted over or under the Crazyflie 2.1.

| | |
|---|---|
| Takeoff weight | 27g |
| Size (WxHxD) | 92x92x29mm |
| Flight time (stock battery) | 7 minutes |
| Controller | Bluetooth LE / Crazyradio dongle |
| Radio range | ∼1km |
| IMU | 3 axis accelerometer / gyroscope<br>High precision pressure sensor |

Table 1.1: Crazyflie 2.1 Specifications

| | |
|---|---|
| Weight | 4.4g |
| Size (WxHxD) | 30x52x8mm |
| Microcontroller | GAP8 - ULP 8+1 core RISC-V MCU |
| Camera | Himax HM01B0 - ULP 320x320 gray-scale camera |
| Power supply | 3V-5V @ VCOM up to 300mA - Max 1.5W |
| Connectivity | ESP32 (WiFi) |
| HyperFlash | 512Mbit / 64MB |
| HyperRam | 64Mbit / 8MB |

Table 1.2: AI Deck 1.1 Specifications

The GAP8 ULP microprocessor is designed by GreenWaves and based on the PULP open core. [3] It incorporates nine RISC-V cores along with a neural processing unit (NPU) which is designed to accelerate convolutional neural networks (CNN). It is capable of running up to 250Mhz. Greenwaves claims that the GAP8 offers peak performance of 200 millions of operations per second (MOPS) at 1mW and up to 10 billions of operations per second (GOPS) at a few tens of mW.
Teams should take these hardware limitations, especially the monocular camera,

| | |
|---|---|
| Total processing power | 22.65 GOPS |
| Power efficiency | 4.24 mW/GOP |
| L1 Memory | 80kb |
| RAM Memory | 512kb |
| Fixed Point | 8, 16, 32-bit |
| Floating Point | None |

Table 1.3: GAP8 Specifications

processing power and memory, into account when implementing their solution for the Nanocopter AI Challenge competition.

## 1.3 Related Work

Obstacle avoidance for autonomous drones is a well studied area with several widely recognized approaches available. These approaches are purely reactive or based on a local or even global map, created with simultaneous localization and mapping (SLAM), structure from motion (SfM) or machine learning. Using low-cost ultrasonic [4, 5] or infrared [6] sensors are light and have a low computational burden, however they can be noisy. Approaches using SLAM [7] or SfM [8] divide the problem into two separate processes, mapping and planning. They use sensor information with high-frequency feature extraction and matching to build a local map of the surroundings [9, 10, 11], a path is planned and the local map is updated repetitively. Though these approaches have been proven to be effective, it is not feasible to implement them on the nano-drone due to its hardware limitations. Using only depth information for obstacle avoidance is still an option, as shown in [12, 13]. These depth maps are often computed from stereo images using the triangulation ranging technique [14, 15, 16], but the nano-drone lacks a stereo camera. The rise of deep learning has however shown major improvements in monocular depth estimation (MDE), the task of inferring depth from one single image [17, 18]. Obstacle avoidance based on MDE has proven to be successful [19, 20], however the inference of the depth maps in these approaches is done offboard whilst the goal of the competition to run as much of the model onboard. [2] has already shown that the AI deck 1.1 is able to run deep CNN for obstacle avoidance, albeit for only corridor-esque environments. [21, 22, 23, 24] show the latest work on MDE for embedded devices, state-of-the-art models able to run on devices without a GPU.

## 1.4 This Work

In this thesis advantage is taken of the recent progress made in MDE by using the MDE models made for embedded devices. By using it in combination with a control scheme the aim is to develop an obstacle avoidance algorithm which could be run onboard the Crazyflie nano-drone. While the potential of MDE for obstacle avoidance has been clearly shown, there remains the challenge of implementing it onboard a nano-drone. Therefore, the main research question driving this research is:

> *Can a lightweight realtime obstacle avoidance algorithm be created using Monocular Depth Estimation while constrained by the hardware limitations of Bitcraze's AI deck 1.1?*

To answer this question we will use and compare two MDE models, FastDepth [24] and PyD-Net [25, 26]. Both models are encoder-decoder networks where a RGB or gray-scale image can be in inserted, they differ however in the output they produce. FastDepth produces a depth map based on metric depth while PyD-Net produces an inverse depth map which is proportional to a disparity map. This is a fundamental difference which we will also research by combining them with a similar control algorithm and comparing the performance in obstacle avoidance. This brings about the second research question:

> *What are the differences in performance between using a depth map and an inverse depth map for an obstacle avoidance algorithm?*

The control algorithm will be simple and inspired by [19], using a strip of average depth values it will be able to get an angular velocity that steers it away from obstacles. The number of bins of this strip will determine the resolution and number of angular velocities it can choose. This leads to our third and final research question:

> *What influence does the resolution of a control algorithm based on depth maps have on the performance of an obstacle avoidance algorithm?*

Results will be obtained in the Webots simulator provided by the IMAV because the AI Deck 1.1 was not available during research.

## 1.5 Outline

The remainder of this thesis is structured as follows. In the next chapter the preliminary theoretical knowledge is covered, relevant for understanding the research

that is done. This will be followed by the approach in which we will cover the steps taken in the development of the complete obstacle avoidance pipeline. This pipeline will be used to obtain the results in the simulator, these results will be shown in chapter 5. In the discussion we will evaluate these results and go over possible improvements for future work. Finally, we will conclude by answering the three research questions.

# Chapter 2

# Theory

In this chapter an overview will be given on the theoretical components of the research.

## 2.1 Encoder-Decoder Network

An encoder-decoder model consists of two neural networks and a hidden state between them, as shown in Figure 2.1. The encoder is a network that takes the input and outputs a feature map, vector or tensor. This feature vector contains the essential information of the input for the task and is called the hidden state. The other network, the decoder, has usually got the same structure as the encoder but reversed. The decoder uses the hidden state to create the intended output. This architecture has been proven to work in natural language processing (NLP) tasks like translation [27], by encoding sentences into a feature/context vector the meaning of the sentence can be stored and used by the decoder for correct translation. It can also be used in computer vision (CV) tasks [28], a fully convolutional encoder is able to extract high-level low-resolution features from the input image which the decoder can use to return the desired output.

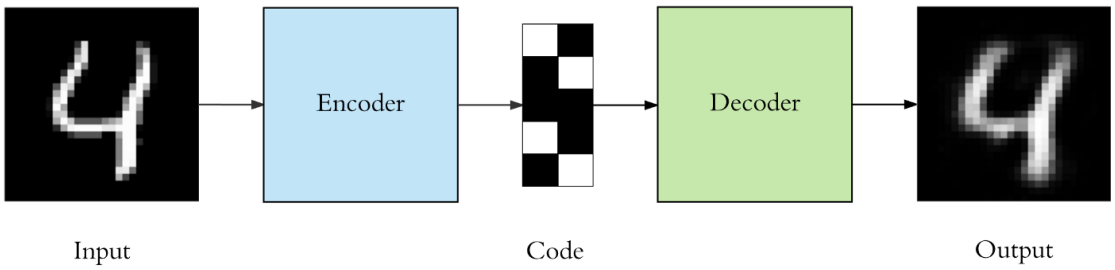



Figure 2.1: Encoder-Decoder Network

## 2.2 (Inverse) Depth Maps

A depth map is an image that contains information about the distance relative to the camera, it can be used to understand geometric relations within a scene. An inverse depth map, which is roughly equal to a disparity map, is able to represent features that are very far off in the distance. Objects like the sun or the clouds, which have a depth value of infinite, will become zero which leads to fewer problems [29]. This does however lead to getting relative depths between objects and not metric depths from the camera, this can be seen in Figure 2.2 where both poles will be perceived as equally close while clearly the second one is further away.
There are several techniques for obtaining depth maps including monocular depth estimation and stereo vision. With stereo images a disparity map can be created by using pixel matching and the shift of these pixels in the two images. When the baseline ($B$) (distance between cameras) and the focal length ($f$) of the stereo camera setup is known a depth can be inferred from the disparity map by using the triangulation formula: $depth = (B * f)/disparity$. [30]

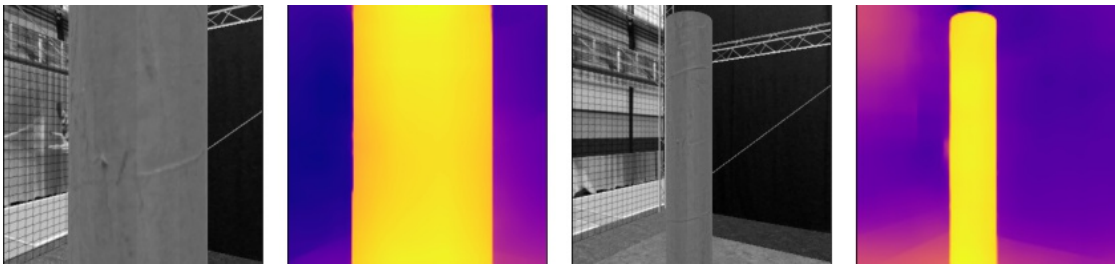


Figure 2.2: Inverse depth map gives relative depths between objects

## 2.3 Monocular Depth Estimation

Monocular depth estimation (MDE) is the task of inferring depth from one single image. Traditional approaches are primarily based on computer vision (CV), using features such as perspective, occlusions, texture information and objects sizes and localization. Even though these approaches are generally not computationally heavy, they do not produce accurate outcomes. Recent approaches use deep convolutional neural networks which has shown to give superior results [31], as displayed in figure 2.3.

Figure 2.3: MDE using the state-of-the-art MiDaS model [31]

# Chapter 3

# Approach

This chapter will discuss the approach taken for this research. It consists of showing the complete pipeline, highlighting the two MDE models and the implemented control algorithm, and explaining how the two models will be evaluated.

## 3.1 Pipeline

The pipeline is shown in figure 3.1 and consists of two modules, the MDE model and the control algorithm. First the Crazyflie's 320x320 gray-scale camera feed is inserted into one of the two MDE models, where it is transformed into a metric depth map or an inverse depth map. This depth map is inserted into the control algorithm which uses the depth values and the angle towards the goal to determine a yaw and a forward speed for the nano-drone.
The MDE models used, FastDepth[1] and PyD-Net[2], were chosen because of their performance on embedded devices and well-documented code for training and testing, this simplified the process of implementing the models into the pipeline.

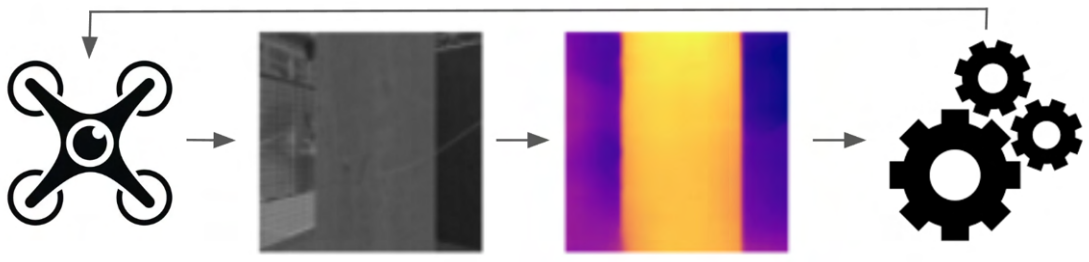


Figure 3.1: Pipeline consisting of the MDE model and the control algorithm

[1]https://github.com/dwofk/fast-depth
[2]https://github.com/mattpoggi/pydnet

## 3.2 FastDepth

FastDepth is a MDE model proposed by researchers at MIT in 2019 [24]. It features an efficient and lightweight encoder-decoder network which further reduced in computational complexity and latency by applying network pruning to the whole network.

### 3.2.1 Architecture

The model consists of an encoder and a decoder, as shown in figure 3.2. The encoder is based on MobileNet, an efficient network that employs depth-wise decomposition to significantly lower its complexity [32]. Depth-wise decomposition expands regular convolutions into depthwise seperable convolutions while maintaining high accuracy [33]. The decoder uses merge and upsample operations on the output of the encoder to form the depth prediction. By using five upsample layers, consisting of 5x5 convolutions expanded by depth-wise decomposition and nearest-neighbor interpolation, and a single pointwise layer at the end a slim and fast decoder is created. By using skip connections the decoder is able to use higher-level feature maps from the encoder to reconstruct a more detailed output. These feature maps are added via addition to the three middle layers of the decoder. Finally the network is pruned after training with NetAdapt [34], which iteratively removes redundant channels until a target accuracy or complexity is reached.

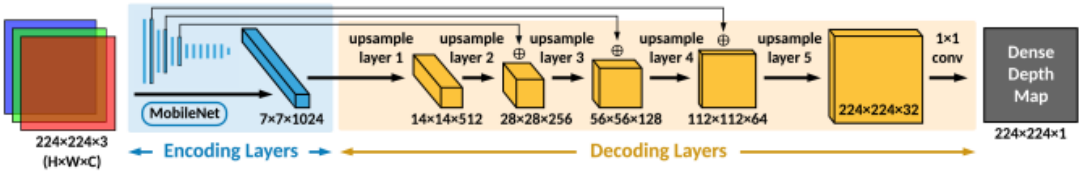



Figure 3.2: FastDepth model

### 3.2.2 Training

The model is trained and evaluated on the NYU Depth v2 dataset [35], a dataset comprised of 407024 single frames from a variety of indoor scenes made with both the RGB and depth cameras of a Microsoft kinect, and implemented in PyTorch. The encoder based on MobileNet is pretrained on ImageNet [36]. The rest of the network is trained for 20 epochs with a batch size of 8, learning rate of 0.01 and a stochastic gradient descent (SGD) optimizer with a momentum of 0.9 and a weight decay of 0.0001. It uses the L1 loss as its loss function.

### 3.2.3 Performance

The final model produces a metric depth map. It has a root-mean-square-error (RMSE) of 0.604m and a $\delta 1$ accuracy, the percentage of predicted pixels where the relative error is within 25%, of 0.771, which was at time of publishing (2019) close to state-of-the-art.

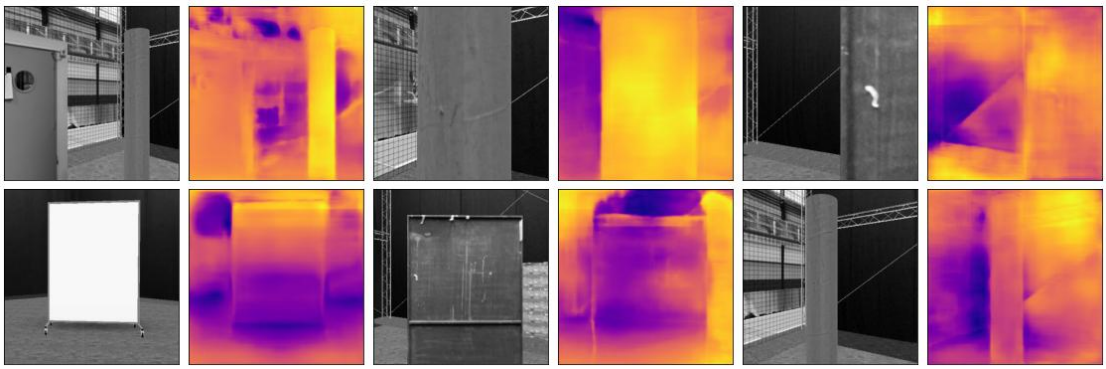


Figure 3.3: FastDepth on camera input nano-drone in simulator

## 3.3 PyD-Net

### 3.3.1 Architecture

The model has a pyramidal architecture which is based on a classical computer vision principle; the image pyramid. This specific pyramid consists of 6 levels (figure 3.4), where each layer corresponds to an image resolution from half to $\frac{1}{64}$ of the original image size. These resolutions are obtained by a small encoder architecture with 12 convolutional layers in which two 3x3 convolutional layers are applied on each level, reducing the resolution down to the lowest resolution. Each of these down-sampling modules produce an increasing number of extracted features which will be used by the module's depth decoder made of 4 3x3 convolutional layers. The output of this decoder can be used to extract a depth map for that specific level but will primarily be used to be passed on to the next level of the pyramid through a 2x2 deconvolution layer with stride 2. This increases the spatial resolution by a factor

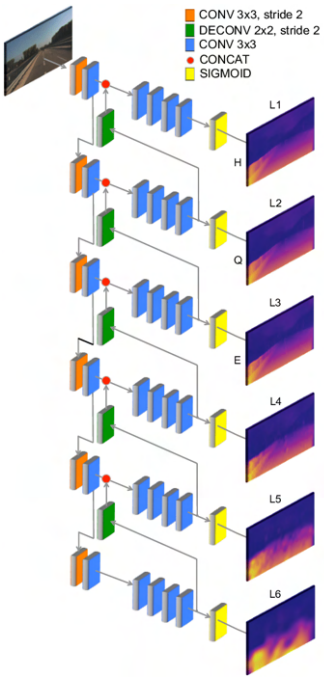



Figure 3.4: PyD-Net model

of 2. This deconvolved feature map is concatenated to the encoded features of that specific level. This procedure is repeated up to the level of highest resolution, which enables the model to predict depth at full resolution. This strategy allows to include the global context of lower image resolution and the details of the higher image resolution while reducing the model's complexity significantly.

### 3.3.2 Training

The model is trained in an unsupervised manner as proposed by [37] on a part of the raw KITTI dataset [38], a dataset with stereo frames of hours of traffic scenarios, and is implemented in Tensorflow. The input data is augmented by randomly flipping the images and applying the following transformations: Random gamma correction, additive brightness and color shifts. It is trained for 50 epochs with a batch size of 8 and a learning rate of 0.0001 for the first 60%. For the remaining 40% this learning rate is halved every 20%. For optimization the Adam optimizer, an extension of stochastic gradient descent, is used with the default settings $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ as proposed in the original paper [39]. A multi-scale loss function is used that computes the loss at each level of the pyramid.

### 3.3.3 Performance

The final model is evaluated on a test split of the KITTI dataset. It produces an inverse depth map at the six decreasing resolutions. It has at half resolution a root-mean-square-error (RMSE) of 5.929 and a $\delta 1$ accuracy, the percentage of predicted pixels where the relative error is within 25%, of 0.8. For a quarter resolution this is 6.185 and 0.789 and for an eight resolution 7.161 and 0.751. These lower resolutions can be inferred by stopping the network early, which results in a significant speed-up of the network of up to 18 times for an eight resolution.

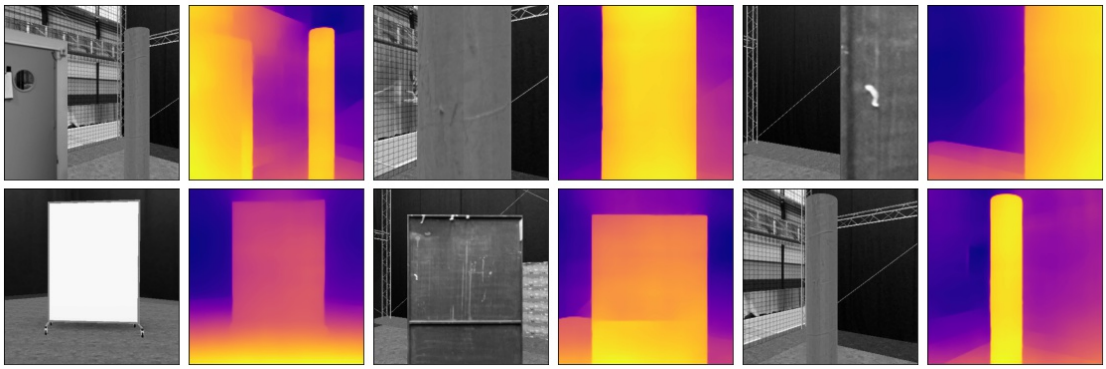


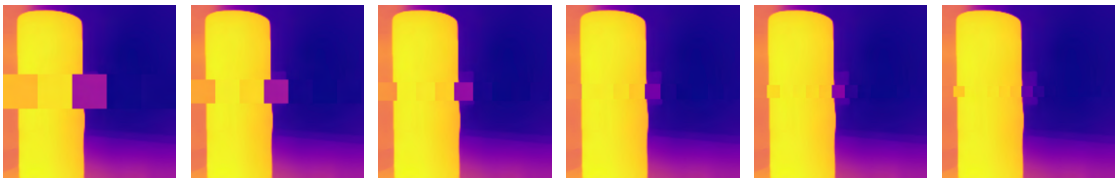
Figure 3.5: PyD-Net on camera input nano-drone in simulator

## 3.4 Control Algorithm

The control algorithm is similar for both the depth map generated by the Fast-Depth model and the inverse depth map generated by the PyD-Net model. It uses the depth map to choose the steering direction the nano-drone should take to avoid an obstacle. It consists of two modules and is currently only limited to one fixed-height due to limitations within the simulator environment. The control algorithm can however, because of its simplistic nature, be easily expanded.

### 3.4.1 Average strip

First, the depth map is simplified by using a horizontal strip consisting of $N$ equal bins through the middle of the image as proposed by [19]. The depth values of these bins are determined by taking the average of all the depth values within the bin. The amount of bins $N$ can be seen as the resolution of the control algorithm and can be adjusted, it is however limited to odd numbers as a center bin for heading straight is required. A vertical strip can be added to enable steering in the up and down directions. Using this average strip reduces the influence of noise and discretizes the number of commands. To finalize the first part of the algorithm the strip of the corresponding frame is added to the front of a deque. The deque has a max size of 50, which means that adding a new strip when the deque is full will remove one at the back. The max size of the deque is also an adjustable parameter and should be adjusted with the throughput time of the MDE model. The final strip which will be given to the next part of the algorithm is an average of this deque and thus an average of the previous 50 strips. This is done to implement a simple memory into the algorithm which is useful when the nano-drone has the obstacle no longer in sight even though it has not passed the obstacle yet.



(a) N=5 (b) N=7 (c) N=9 (d) N=11 (e) N=13 (f) N=15

Figure 3.6: Resolution differences for average strip

### 3.4.2 Direction picker

The direction picker uses the average strip, the angle towards the goal, a control dictionary and a depth threshold to choose the most promising direction. The

angle towards the goal is initially calculated by using the *atan*2 function on the position of the goal and the drone, however this is only feasible in the simulation as these positions cannot easily be acquired by a real nano-drone. Nevertheless, this solution will suffice as this work mainly focuses on obstacle avoidance and less on the path planning aspect. This angle is then used to choose the bin that steers the nano-drone in the direction of the goal. First the bins are each given an equal discrete angle range based on the Field of View (FoV) of the camera, the nano-drone's default FoV is 0.84 radians. By letting the two outer bins get all values outside the FoV of the camera the nano-drone is able to turn around when the goal is positioned behind it.


Figure 3.7: Example ($N$=5): Choosing goal bin when angle towards goal = -0.105

The depth threshold is used to determine which bins are free of obstacles and thus which steering commands will keep the nano-drone away from obstacles. Here the implementation slightly differs for the two MDE models. The FastDepth model uses metric depth and thus bins above a certain metric threshold are seen as safe bins, while the PyD-Net model uses inverse depth values so here bins below a certain threshold are considered as safe. The threshold used for the FastDepth model is 1.4m and for the PyD-Net model the value 80, this parameter is also adjustable and it influences how close the nano-drone gets to an obstacle before avoiding it.


Figure 3.8: Example: Determining safe bins with a depth threshold = 80

Finally, the bin will be picked that is safe and closest to the goal bin. As this is the bin which brings you towards the goal without bumping into an obstacle.
There are however two scenarios that require special attention. The first is when all bins are above or below the set depth threshold, this is the case when the nano-drone is right in front of an obstacle. No bin can be picked so it forces the nano-drone to turn around by choosing the outer bin which is closest to the goal bin. The other scenario is when there are two safe bins with equal preference, this

Figure 3.9: Example: Choosing a safe bin closest to the goal bin

can occur when for example the middle bin is the goal bin and only the two outer bins are safe. They are equally far away from the goal bin so the bin with the most depth will be chosen, if this is the highest or lowest value depends on which model is used.

(a) No safe bins

(b) Equally preferred bins

Figure 3.10: Exception cases

The index of the chosen bin is used to obtain the yaw steering and forward speed from a predetermined control dictionary. This control dict is created by using the resolution parameter ($N$), a parameter for the yaw turning speed (yawRate) and a parameter for the forward speed (forwardSpeed). The default values for these speeds are respectively 0.5 and 0.2. By adjusting the yawRate parameter the nano-drone is able to do sharper turns when necessary. The bin in the middle only moves the drone forward while the outer bins only turn the drone. The bins in between move the drone forward and turn the drone, the yawRate increases by a factor of 2 as the bins get closer to the outer bins.

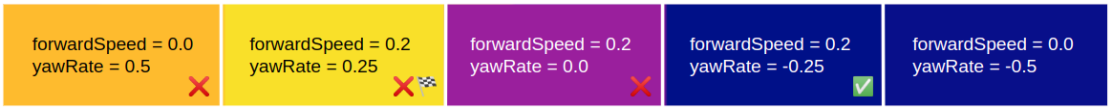



Figure 3.11: Example: Using the chosen bin to get steering commands

This concludes the control algorithm, a simple and easily adjustable implementation which works for both MDE models. This makes for a fair comparison of the two complete pipeline implementations.

## 3.5 Evaluation

The evaluation is done in the Webots simulator of the competition environment which is based on Delft's real Cyberzoo (figure 1.1)[3]. Both the models with their corresponding control algorithm will be tested, with resolutions varying between $N$=5 and $N$=15, in three different obstacle stages with increasing difficulty. Each model + resolution combination will be tested for 10 runs, this makes for a total of 360 runs for all possible combinations. The setup of each run is predetermined to evaluate both algorithms on the same stages. The nano-drone will always start at (0, -4). The goal is fixed at Y = 4 while its X-value is different for each run. The obstacles are obtained from the cyberzoo environment and placed between between the nano-drone and the goal, each obstacle has a randomized distance (in a range) from the nano-drone and a randomized angle.
The performance of the algorithm will be evaluated as follows:

1. Success rate: Total number of succesful runs divided by the total number of runs.

2. Average time (s): Sum of the duration of every successful run divided by the total number of successful runs.

3. Average distance (m): Sum of the distances travelled of every successful run divided by the total number of successful runs.

If the nano-drone crashes into an obstacle the simulator will reset and the run will count as a failure. The run will be completed when the nano-drone gets within 0.2 meters of the goal.

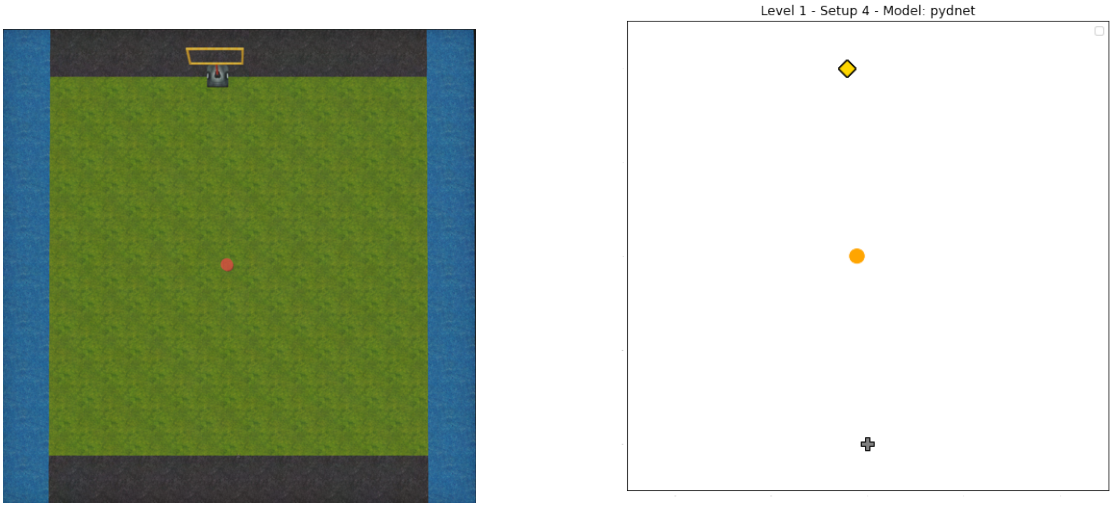
(a) Orange pole (b) Metal panel (c) White panel

Figure 3.12: Obstacles used for evaluation

[3]https://tudelftroboticsinstitute.nl/labs/cyber-zoo

### 3.5.1 Stage 1

The first stage consists of one single obstacle, a round orange pole with an estimated diameter of 25cm. This orange pole will be positioned between 2 and 4 meters from the starting position of the nano-drone. Due to its texture and color, it is easy to distinguish from the background. This in combination with its shape and size make it the easiest obstacle to avoid.




(a) Simulator (b) Plotted

Figure 3.13: Stage 1

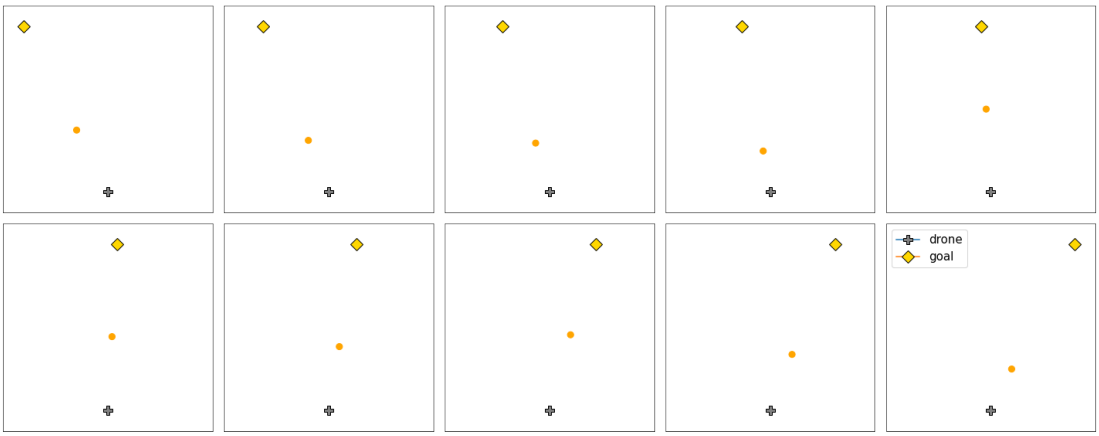
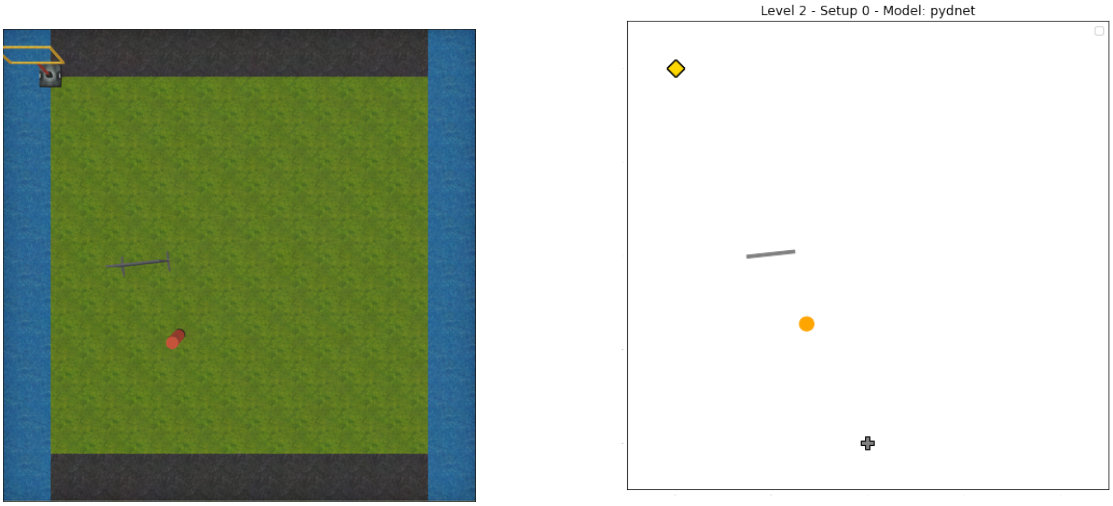



Figure 3.14: 10 predetermined setups stage 1

### 3.5.2 Stage 2

The second stage consists of the orange pole and a metal panel with an estimated with of 80cm. The metal panel will be behind the pole and between 4.5 and 6 meters from the starting position. Its angle can be in range of -1 and 1 radians. The metal panel is harder to distinguish because the texture has a similar color as the background, especially when using black and white images. Furthermore, it can influence the path taken around the orange pole due to its position behind it.




(a) Simulator (b) Plotted

Figure 3.15: Stage 2

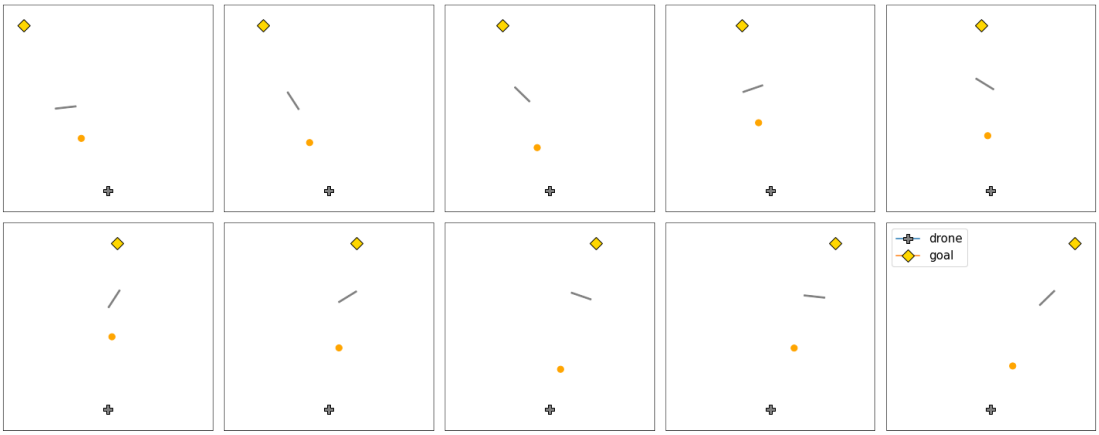
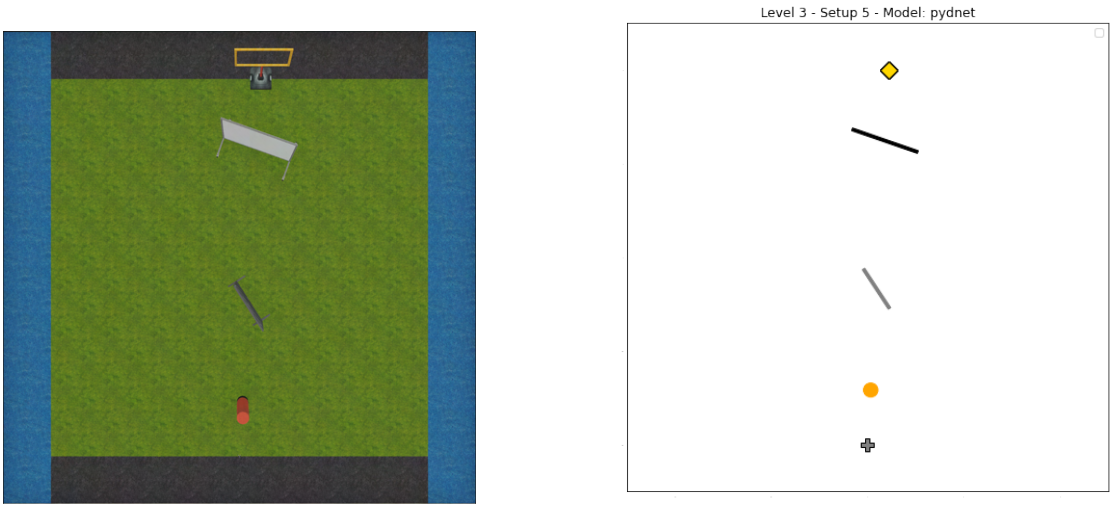



Figure 3.16: 10 predetermined setups stage 2

### 3.5.3 Stage 3

The third stage consists of the orange pole, metal panel and a white panel with an estimated width of 1.2m. The white panel is added and positioned behind the metal panel. The white panel is between 5 and 6.5 meters from the starting position. Its angle can be in range of 2.1 and 4.2 radians. The white panel is due to its size a difficult obstacle to avoid, requiring the nano-drone to make sharp turns or turn around when it gets too close.




(a) Simulator (b) Plotted

Figure 3.17: Stage 3

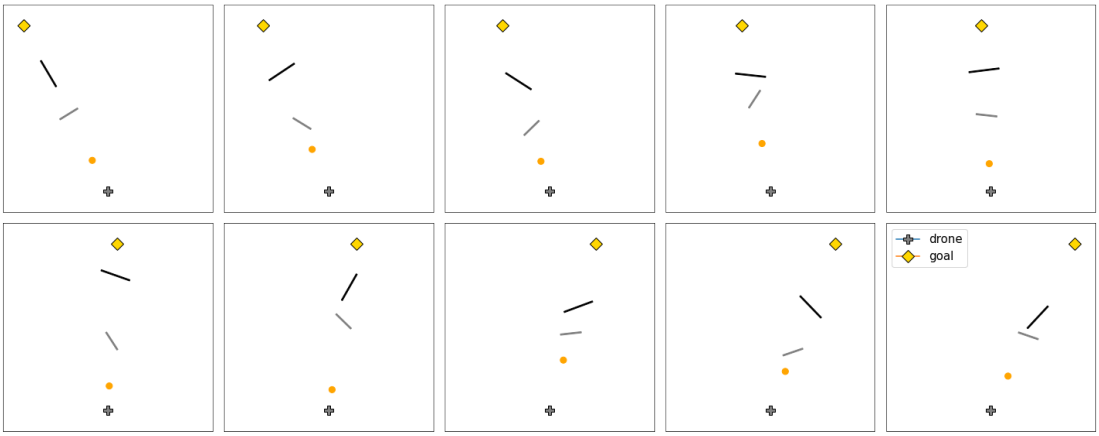



Figure 3.18: 10 predetermined setups stage 3

# Chapter 4

# Results

The results are presented in the tables 4.1 4.2 4.3 below, where the runs per model per setup are also plotted. These plots show the flight-path of the nano-drone for each resolution, they can also be found individually in a large format in Appendix A. Referencing to a plot as an example will be done in the following format: *(stage-setup-model)*.
The results show that PyD-Net has an overall better performace than FastDepth, the plots confirm this by showing PyD-Net reaching the goal more often and taking smoother paths around the obstacle in all 3 stages. The results and plots of the more challenging stages 2 and 3 enforce this by showing a significant difference in average distance and therefore also average time between the models. In the FastDepth model the nano-drone seems to take long detours when it is pointed towards one of the corners of the room, as can be seen in plots *(2-1-FastDepth)*, *(2-5-FastDepth)*, *(3-9-FastDepth)*. Which is expected behavior as a corner has the most depth when being enclosed by a wall left and right. This happens, though less often, also for the PyD-Net model, as seen in *(2-3-PyD-Net)*, *(3-5-PyD-Net)*. Finally, the results also show that for both models the lower resolution control algorithm performs better than the higher ones. the only exception can be found in the second stage for the FastDepth model. This increase in performance is once again confirmed by the plots as the red line ($N$=5) can be seen maneuvering through the obstacle course to get to the goal in most of the setups.

## 4.1 Stage 1

| Performance | Resolution (N) | 5 | 7 | 9 | 11 | 13 | 15 |
|---|---|---|---|---|---|---|---|
| FastDepth Model | Success rate | **0.9** | 0.5 | 0.6 | 0.5 | 0.4 | 0.2 |
| | Average time (s) | 45.96 | 44.01 | 43.78 | **43.40** | 43.72 | 43.52 |
| | Average distance (m) | 8.34 | 8.08 | 8.17 | 8.05 | 8.09 | **8.00** |
| PyD-Net Model | Success rate | **0.9** | 0.8 | 0.4 | 0.5 | 0.3 | 0.3 |
| | Average time (s) | 44.46 | **43.43** | 44.70 | 44.36 | 44.46 | 44.50 |
| | Average distance (m) | 8.28 | **8.20** | 8.43 | 8.37 | 8.40 | 8.40 |

Table 4.1: Stage 1 performance

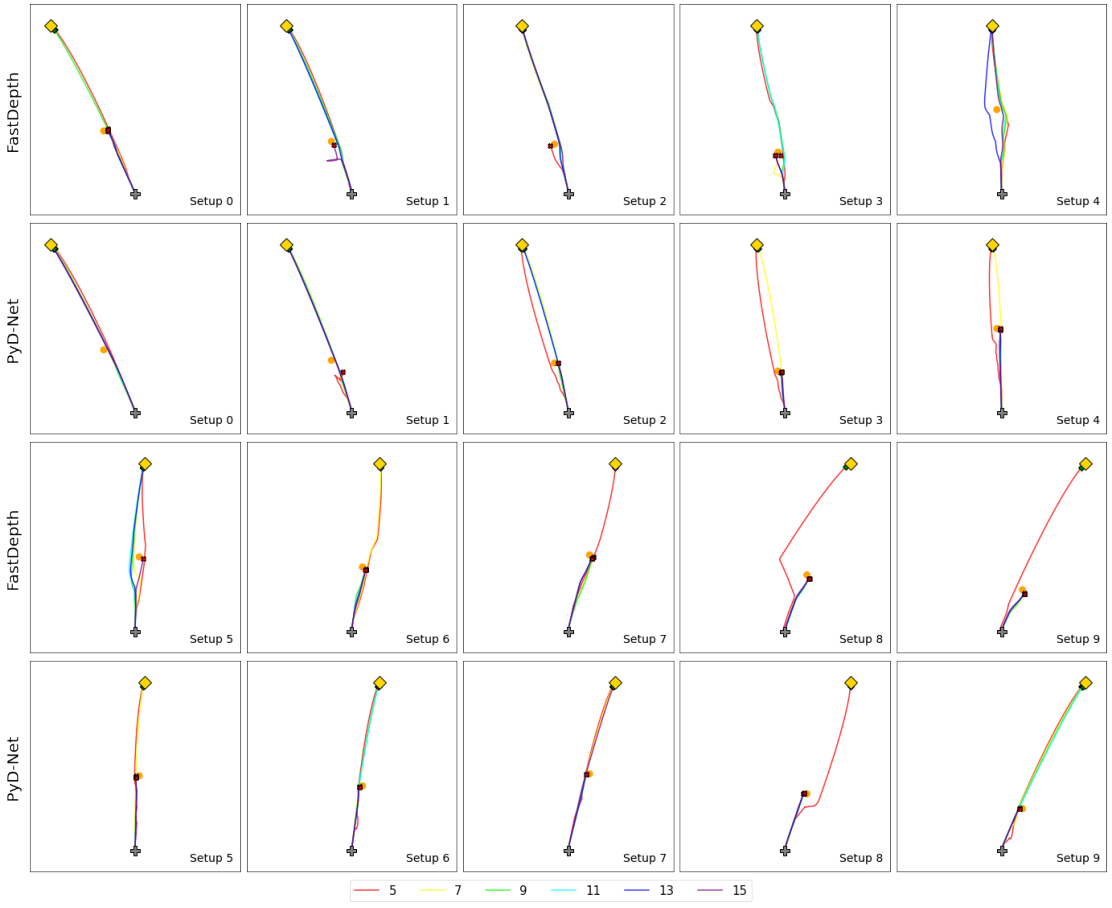



Figure 4.1: Plotted results Stage 1

## 4.2 Stage 2

| Performance | Resolution (N) | 5 | 7 | 9 | 11 | 13 | 15 |
|---|---|---|---|---|---|---|---|
| FastDepth Model | Success rate | 0.4 | **0.6** | 0.3 | 0.2 | 0.2 | 0.2 |
| | Average time (s) | 69.50 | 68.57 | **62.03** | 65.02 | 65.71 | 83.30 |
| | Average distance (m) | 11.79 | 12.25 | **10.60** | 11.40 | 11.49 | 13.69 |
| PyD-Net Model | Success rate | **0.8** | 0.4 | 0.3 | 0.1 | 0.2 | 0.2 |
| | Average time (s) | 55.03 | **43.26** | 44.59 | 44.22 | 45.20 | 45.28 |
| | Average distance (m) | 9.83 | **8.15** | 8.34 | 8.38 | 8.55 | 8.55 |

Table 4.2: Stage 2 performance

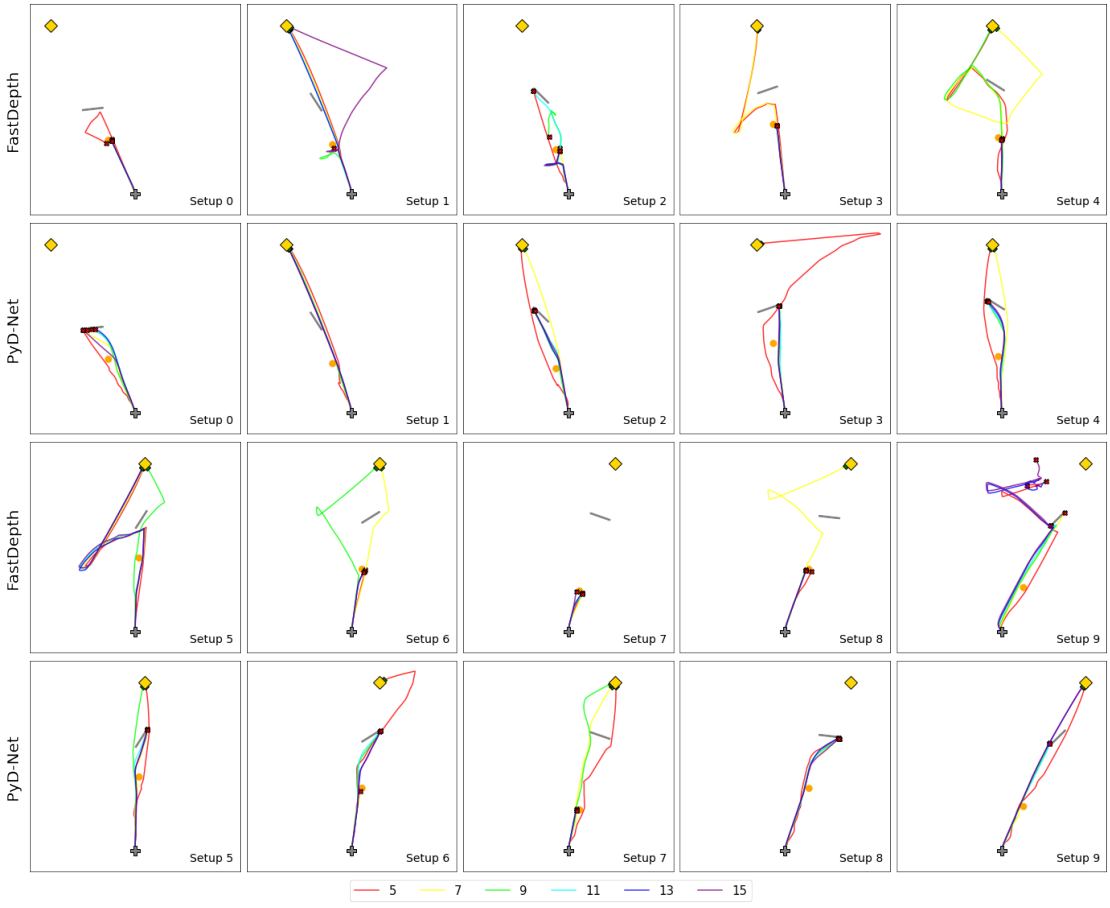



Figure 4.2: Plotted results Stage 2

## 4.3 Stage 3

| Performance | Resolution (N) | 5 | 7 | 9 | 11 | 13 | 15 |
|---|---|---|---|---|---|---|---|
| FastDepth Model | Success rate | **0.8** | 0.4 | 0.3 | 0.1 | 0.0 | 0.0 |
| | Average time (s) | 80.31 | **57.38** | 85.70 | 74.82 | NaN | NaN |
| | Average distance (m) | 12.73 | **9.99** | 13.81 | 12.30 | NaN | NaN |
| PyD-Net Model | Success rate | **0.9** | 0.8 | 0.7 | 0.4 | 0.5 | 0.5 |
| | Average time (s) | 46.67 | 46.16 | 49.65 | **44.78** | 46.03 | 44.85 |
| | Average distance (m) | 8.54 | 8.48 | 9.19 | 8.38 | 8.56 | **8.36** |

Table 4.3: Stage 3 performance

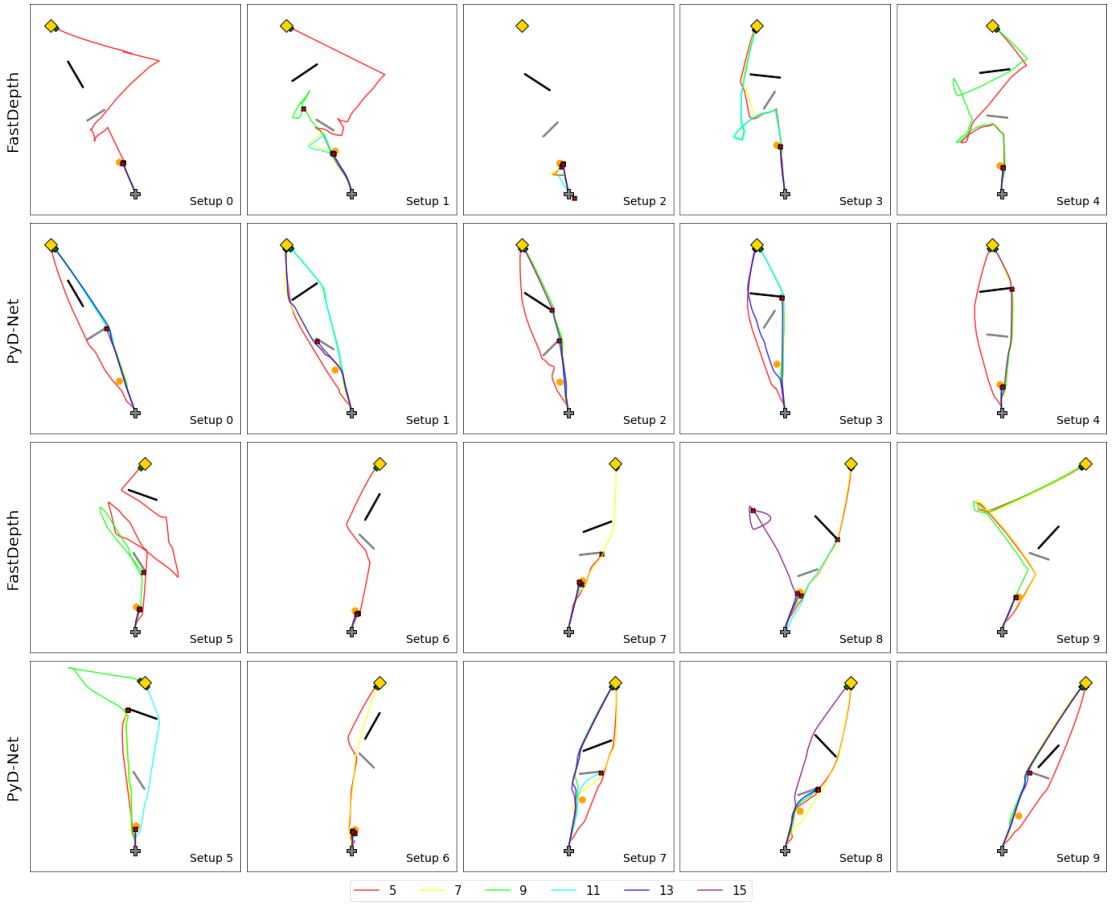



Figure 4.3: Plotted results Stage 3

# Chapter 5

# Discussion

The aim of this thesis was to develop a light-weight real-time obstacle avoidance algorithm based on MDE for the AI Deck 1.1, which is mounted on the Crazyflie 2.1 and used in the Nanocopter AI Challenge. From the results, it becomes evident that it is possible to create this light-weight and real-time algorithm using MDE models specifically made for embedded systems. When using a resolution of 5 bins the algorithm based on the PyD-Net model is able to reach the goal at least 80% of the time in all tested environments, while the FastDepth model has similar performance except in stage 2. The results are however obtained in a simulator, which will never fully match performance in the real world. Especially when considering that the models used can be very sensitive to changes in lighting, texture or background, as can be seen in Figure 3.3 of the FastDepth model. This can also be seen in plot *(2-9-FastDepth)* where the control algorithm with the FastDepth model crashes while having an open path towards the goal. This is likely caused by the position of the goal which was near one of the corners, as mentioned in the previous chapter corners seem to attract the nano-drone which leads to large distances flown. However, it can also be seen in the corresponding plots that when the nano-drone reaches the corner at a certain distance it turns around and heads toward another direction. The combination of these two properties plus heading towards the goal is likely what caused the unexpected behavior of the control algorithm which led the nano-drone to crash. While both models have roughly the same $\delta_1$ accuracy the PyD-Net model produces better and less noisier results in the simulator environment, as can be seen when comparing Figure 3.3 with Figure 3.5. This could have potentially also played a role in the lesser performance of the FastDepth control algorithm.

The results show that the lower resolutions of the control algorithm perform significantly better than the higher resolutions. The higher resolutions tend to get very close to the obstacle because the smaller bins return a lower steering speed

thus forcing the nano-drone to make a tighter turn around it. This however makes the turn often too tight and thus making the nano-drone hit the obstacle, as can be seen in *(3-6-FastDepth)*. Taking a tight turn around an obstacle is not per definition something you should avoid, as it can lead to faster times and lower distances.
Finally, implementing the models on the AI deck 1.1 is not feasible as the power and memory constraints of the deck cannot be matched by the MDE models. FastDepth has an estimated active power consumption of 1.9W, requires 11MB of storage and 100MB of RAM memory [24]. PyD-Net has similar requirements as it has an estimated active power consumption of 2.5W, requires 8MB storage and 150MB of RAM memory [25]. Table 1.2 shows that the AI deck cannot offer these requirements thus limiting this work to the simulated environment.

## 5.1 Future work

As mentioned above, there are some improvements that can be made in both the MDE models and the control algorithm. First of all, the MDE models can be changed for newer and more robust and light-weight models like PyD-Net2 [23] or [40] which can serve as an improvement for the FastDepth model. This can be easily done because the model and control algorithm are two independent components.
The control algorithm can be extended by implementing a vertical average strip for controlling the height of the drone. Furthermore improvements can be made in avoiding the nano-drone being attracted by the corners of the room, a triangular function could for example be used to detect a corner move away from it if necessary.
One of the main problems of the control algorithm in this work is higher resolutions having the tendency to hug the obstacle it has to avoid, this can be solved by implementing configuration-space expansion as described in [41]. This enlarges the obstacle in the depth map to the minimum size that guarantees the nano-drone to safely avoid it.

# Chapter 6

# Conclusion

In this work, a light-weight real-time obstacle avoidance algorithm was created based on two MDE models for the Nanocopter AI Challenge. Furthermore, the performance of these models was compared and multiple resolutions ($N$) were tested for the control algorithm. This control algorithm is based on taking an average horizontal strip with N bins. Using these bins, a goal angle and a depth threshold a direction is picked and the obstacle is avoided. To compare the two complete pipelines they were tested in three stages with increasing difficulty in the simulator environment provided by the IMAV.
The main research question of this thesis was:

> *Can a lightweight realtime obstacle avoidance algorithm be created using Monocular Depth Estimation while constrained by the hardware limitations of Bitcraze's AI deck 1.1?*

As described in the previous chapter, we succeeded in developing an light-weight algorithm which is able to avoid obstacles using MDE in real-time. Our model of choice is the PyD-Net model as it is more stable and has an overall better performance. However, even though the hardware limitations were considered while selecting the models, implementation on the AI deck is difficult as both models still require a significant amount of memory and power. Nevertheless, the IMAV allows off-board processing of images at the price of a lower score multiplier, so in principle the method proposed can still be used in the Nanocopter AI Challenge. Especially when considering that the control algorithm is light-weight and can be run onboard.
The following sub-question was used to compare the models in terms of performance of their depth map combined with the control algorithm:

> *What are the differences in performance between using a depth map and an inverse depth map for an obstacle avoidance algorithm?*

Comparing the depth map with the inverse depth map shows a higher overall performance in the latter one. Because inverse depth maps use relative depth between objects, the nano-drone turns early because it perceives the obstacle to be closer than it actually is. This results in a safer path around the obstacle. The FastDepth model however is noisy compared to the more stable PyD-Net model and this leads to unexpected behavior of the nano-drone, like it taking a longer path towards the goal. To further compare the performance of these two depth maps one could use two state-of-the-art MDE models and compare their performance in the simulator environment.
The following sub-question was used to compare the influence the resolution of the control algorithm, in terms of number of bins, has on its performance:

> *What influence does the resolution of a control algorithm based on depth maps have on the performance of an obstacle avoidance algorithm?*

Finally, the resolution has a high influence on the performance of the control algorithm based on MDE. As mentioned before, lower resolutions outperform the higher ones because their turning angle is greater when avoiding an obstacle. This results in a higher success-rate as more obstacles are cleared.
The task of implementing a control algorithm based on MDE on the AI Deck 1.1 is difficult as state-of-the-art embedded MDE models are not yet light-weight enough to be implemented on its limited hardware. This work has however shown that running a MDE model that produces inverse depth maps off-board combined with a simple low-resolution control algorithm results in high performance in three increasingly difficult obstacle stages.

# Bibliography


[1] Hanna Miiller, Daniele Palossi, Stefan Mach, Francesco Conti, and Luca Benini. Fünfiiber-drone: A modular open-platform 18-grams autonomous nano-drone. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1610–1615. IEEE, 2021.

[2] Daniele Palossi, Francesco Conti, and Luca Benini. An open source and open hardware deep learning-powered visual navigation engine for autonomous nano-uavs. In *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 604–611. IEEE, 2019.

[3] Francesco Conti, Davide Rossi, Antonio Pullini, Igor Loi, and Luca Benini. Energy-efficient vision on the pulp platform for ultra-low power parallel computing. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6. IEEE, 2014.

[4] Nils Gageik, Thilo Müller, and Sergio Montenegro. Obstacle detection and collision avoidance using ultrasonic distance sensors for an autonomous quadrocopter. *University of Wurzburg, Aerospace information Technologhy (germany) Wurzburg*, pages 3–23, 2012.

[5] Meng Guanglei and Pan Haibing. The application of ultrasonic sensor in the obstacle avoidance of quad-rotor uav. In *2016 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*, pages 976–981. IEEE, 2016.

[6] Nils Gageik, Paul Benz, and Sergio Montenegro. Obstacle detection and collision avoidance for a uav with complementary low-cost sensors. *IEEE Access*, 3:599–609, 2015.

[7] Omid Esrafilian and Hamid D Taghirad. Autonomous flight and obstacle avoidance of a quadrotor by monocular slam. In *2016 4th International Conference on Robotics and Mechatronics (ICROM)*, pages 240–245. IEEE, 2016.

[8] Changchang Wu. Towards linear-time incremental structure from motion. In *2013 International Conference on 3D Vision-3DV 2013*, pages 127–134. IEEE, 2013.

[9] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[10] Michael Montemerlo and Sebastian Thrun. Simultaneous localization and mapping with unknown data association using fastslam. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 2, pages 1985–1991. IEEE, 2003.

[11] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849. Springer, 2014.

[12] Yuichiro Niwa, Shûichi Yukita, and Hiroshi Hanaizumi. Depthmap-based obstacle avoidance on rough terrain. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 2, pages 1612–1617. IEEE, 2004.

[13] Akshay Aggarwal, Astha Kukreja, and Pankaj Chopra. Vision based collision avoidance by plotting a virtual obstacle on depth map. In *The 2010 IEEE International Conference on Information and Automation*, pages 532–536. IEEE, 2010.

[14] Kimberly McGuire, Guido De Croon, Christophe De Wagter, Karl Tuyls, and Hilbert Kappen. Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone. *IEEE Robotics and Automation Letters*, 2(2):1070–1076, 2017.

[15] Larry Matthies, Roland Brockers, Yoshiaki Kuwata, and Stephan Weiss. Stereo vision-based obstacle avoidance for micro air vehicles using disparity space. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 3242–3249. IEEE, 2014.

[16] Boitumelo Ruf, Sebastian Monka, Matthias Kollmann, and Michael Grinberg. Real-time on-board obstacle avoidance for uavs based on embedded stereo vision. *arXiv preprint arXiv:1807.06271*, 2018.

[17] Yue Ming, Xuyang Meng, Chunxiao Fan, and Hui Yu. Deep learning for monocular depth estimation: A review. *Neurocomputing*, 438:14–33, 2021.

[18] Xingshuai Dong, Matthew A Garratt, Sreenatha G Anavatti, and Hussein A Abbass. Towards real-time monocular depth estimation for robotics: A survey. *arXiv preprint arXiv:2111.08600*, 2021.

[19] Punarjay Chakravarty, Klaas Kelchtermans, Tom Roussel, Stijn Wellens, Tinne Tuytelaars, and Luc Van Eycken. Cnn-based single image obstacle avoidance on a quadrotor. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 6369–6374. IEEE, 2017.

[20] Xin Yang, Jingyu Chen, Yuanjie Dang, Hongcheng Luo, Yuesheng Tang, Chunyuan Liao, Peng Chen, and Kwang-Ting Cheng. Fast depth prediction and obstacle avoidance on a monocular drone using probabilistic convolutional neural network. *IEEE Transactions on Intelligent Transportation Systems*, 22(1):156–167, 2019.

[21] Siping Liu, Laurence T Yang, Xiaohan Tu, Renfa Li, and Cheng Xu. Lightweight monocular depth estimation on edge devices. *IEEE Internet of Things Journal*, 2022.

[22] Mehmet Kerim Yucel, Valia Dimaridou, Anastasios Drosou, and Albert Saa-Garriga. Real-time monocular depth estimation with sparse supervision on mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2428–2437, 2021.

[23] Matteo Poggi, Fabio Tosi, Filippo Aleotti, and Stefano Mattoccia. Real-time self-supervised monocular depth estimation without gpu. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[24] Diana Wofk, Fangchang Ma, Tien-Ju Yang, Sertac Karaman, and Vivienne Sze. Fastdepth: Fast monocular depth estimation on embedded systems. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6101–6108. IEEE, 2019.

[25] Matteo Poggi, Filippo Aleotti, Fabio Tosi, and Stefano Mattoccia. Towards real-time unsupervised monocular depth estimation on cpu. In *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 5848–5854. IEEE, 2018.

[26] Filippo Aleotti, Giulio Zaccaroni, Luca Bartolomei, Matteo Poggi, Fabio Tosi, and Stefano Mattoccia. Real-time single image depth perception in the wild with handheld devices. *Sensors*, 21(1):15, 2020.

[27] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase

representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[28] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[29] JM Martínez Montiel, Javier Civera, and Andrew J Davison. Unified inverse depth parametrization for monocular slam. Robotics: Science and Systems, 2006.

[30] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[31] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[32] 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. *arXiv preprint arXiv:1704.04861*, 2017.

[33] Yihui He, Jianing Qian, and Jianren Wang. Depth-wise decomposition for accelerating separable convolutions in efficient convolutional neural networks. *arXiv preprint arXiv:1910.09455*, 2019.

[34] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018.

[35] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[36] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[37] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 270–279, 2017.

[38] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012.

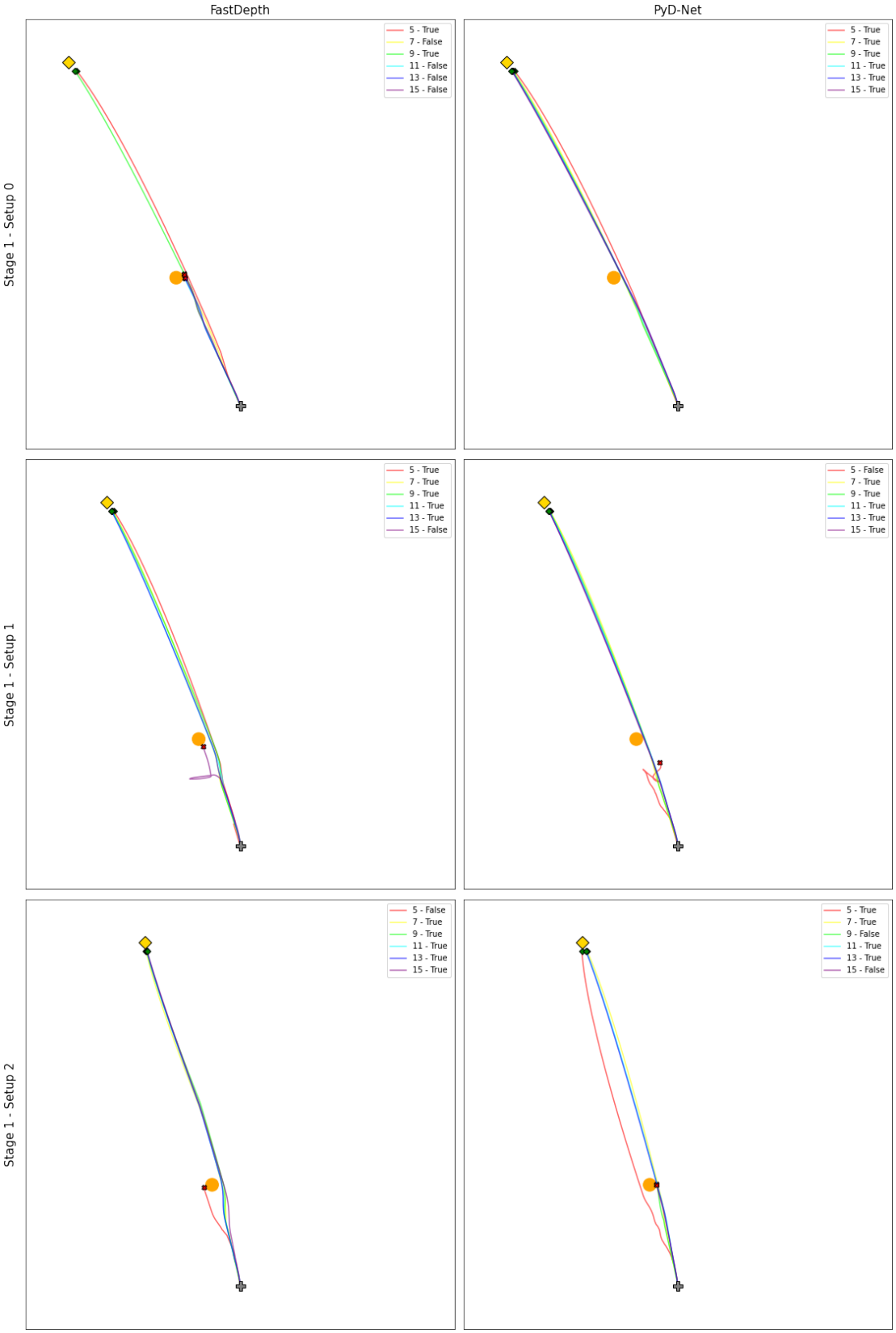
[39] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

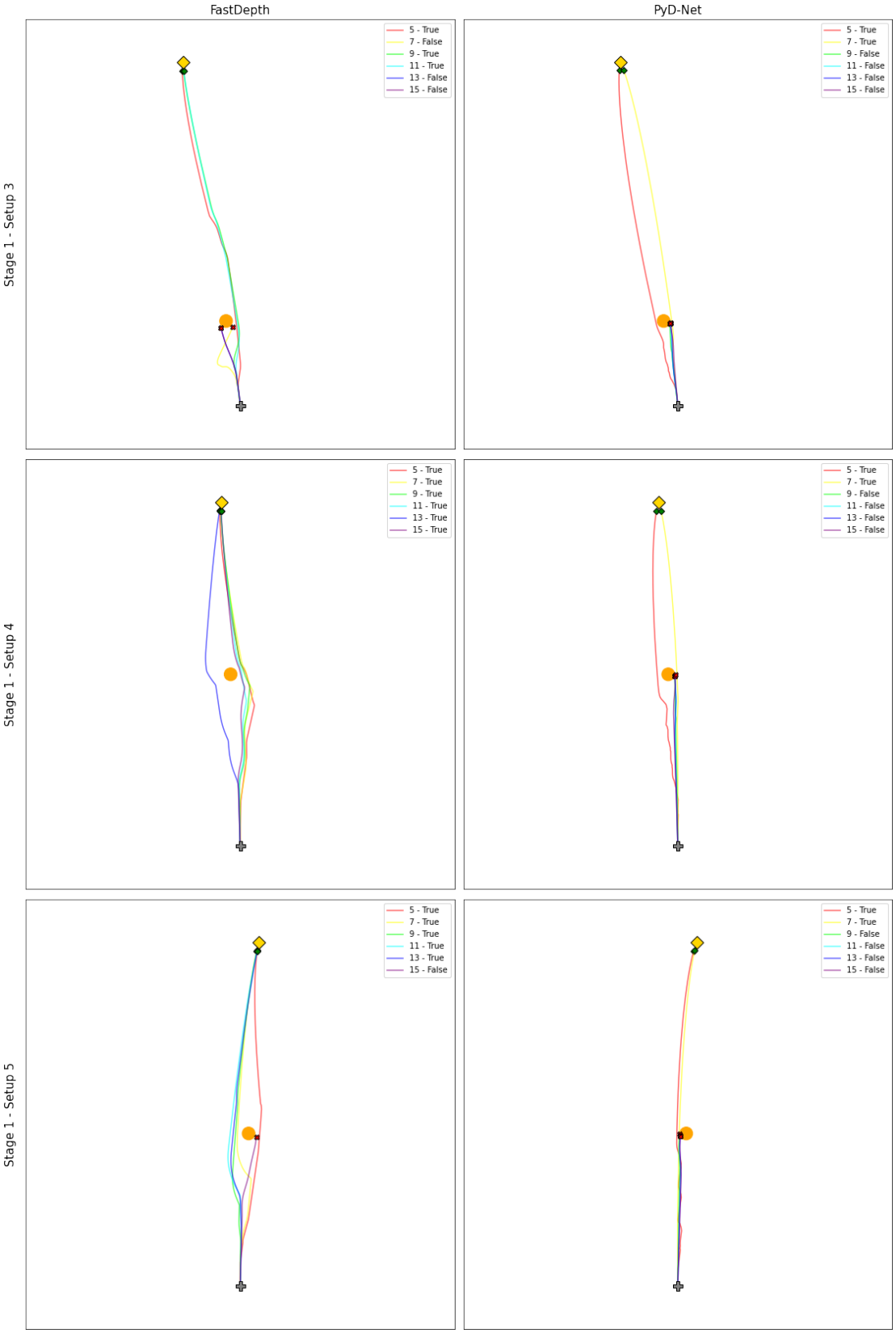
[40] Tim Heydrich, Yimin Yang, Xiangyu Ma, Yu Liu, and Shan Du. A novel lightweight network for fast monocular depth estimation. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2260–2264. IEEE, 2022.

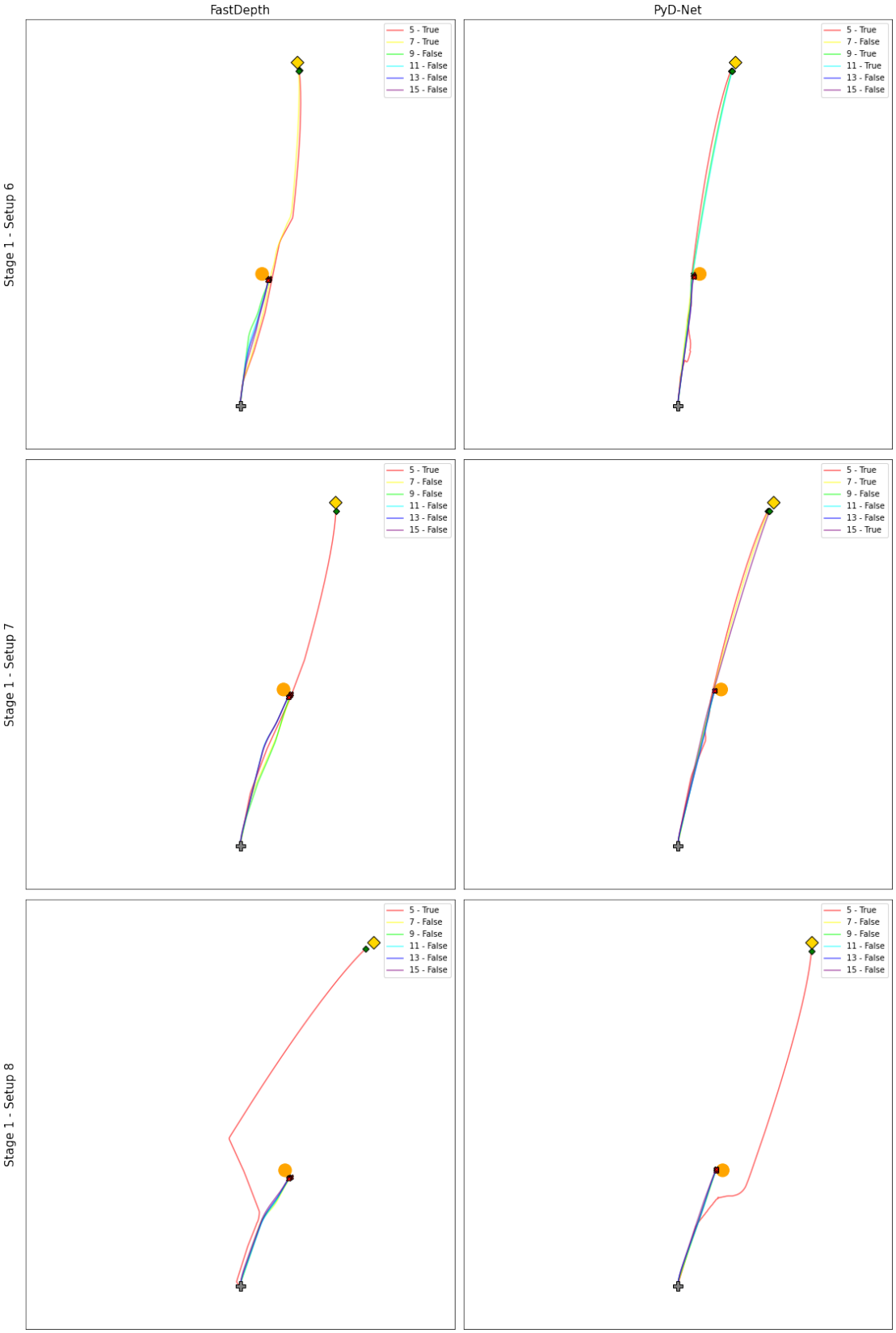
[41] Roland Brockers, Anthony Fragoso, Brandon Rothrock, Connor Lee, and Larry Matthies. Vision-based obstacle avoidance for micro air vehicles using an egocylindrical depth map. In *International Symposium on Experimental Robotics*, pages 505–514. Springer, 2016.

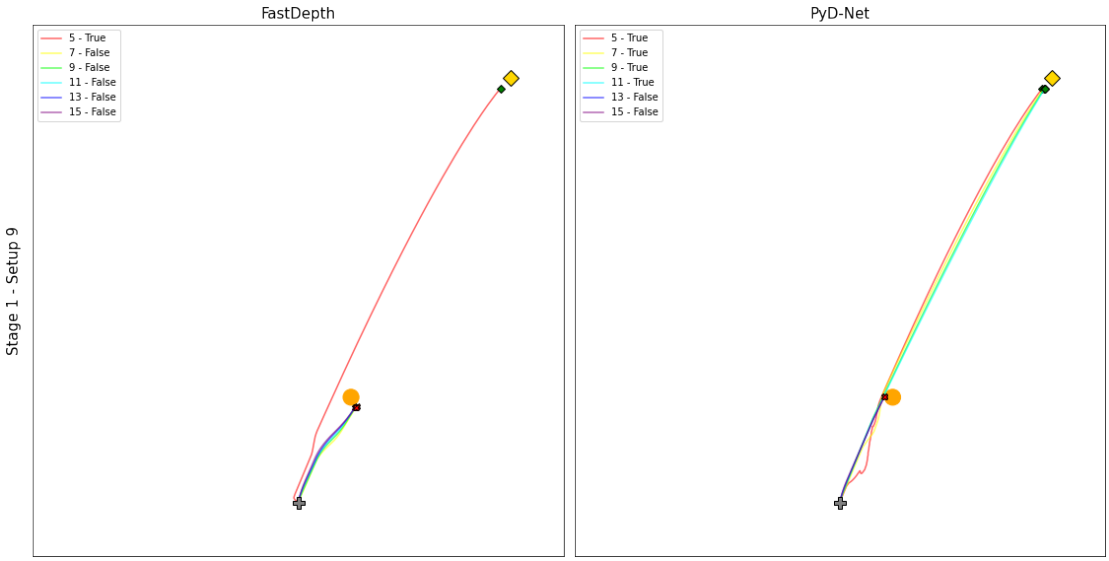
# Appendix A

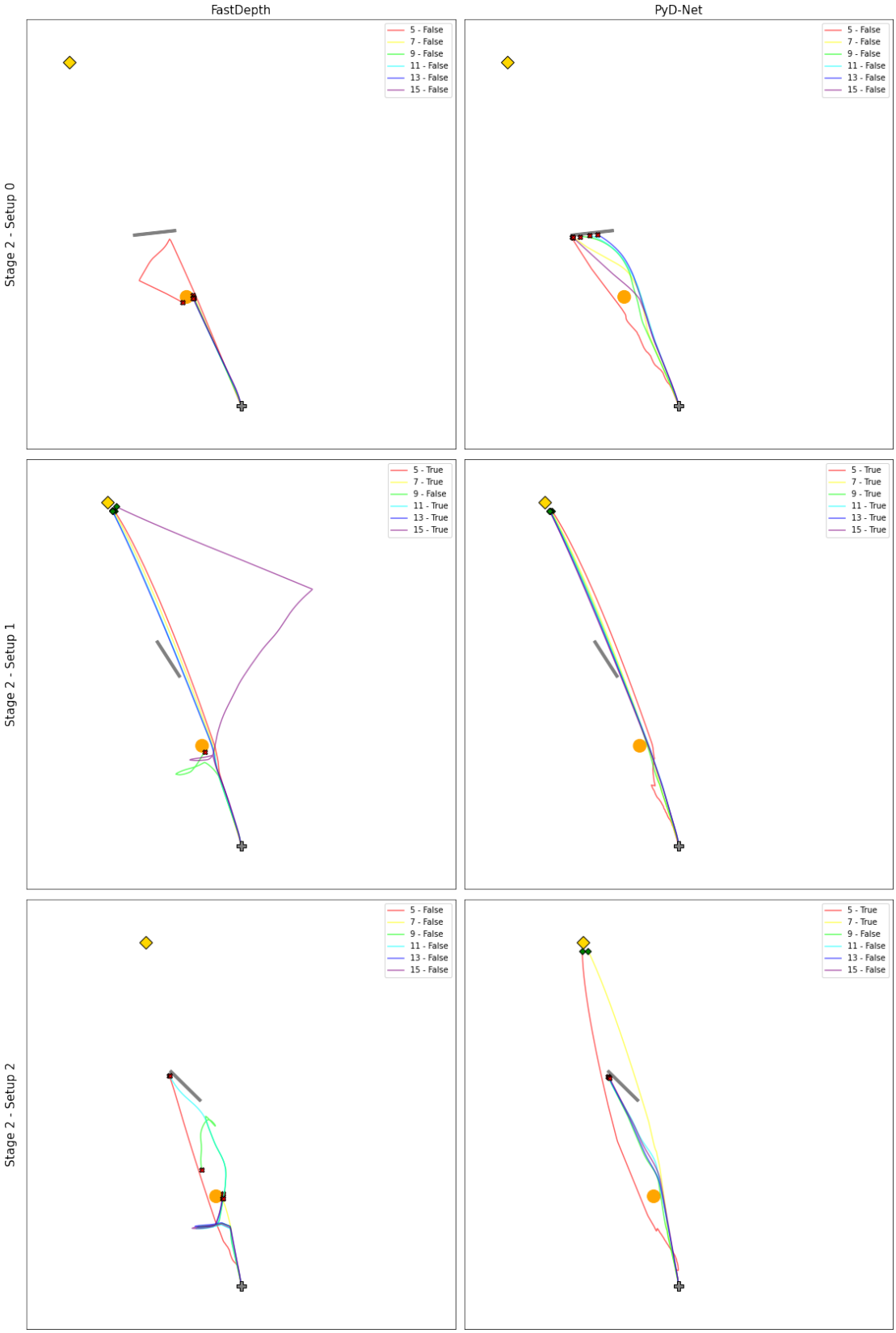
# Plotted Runs

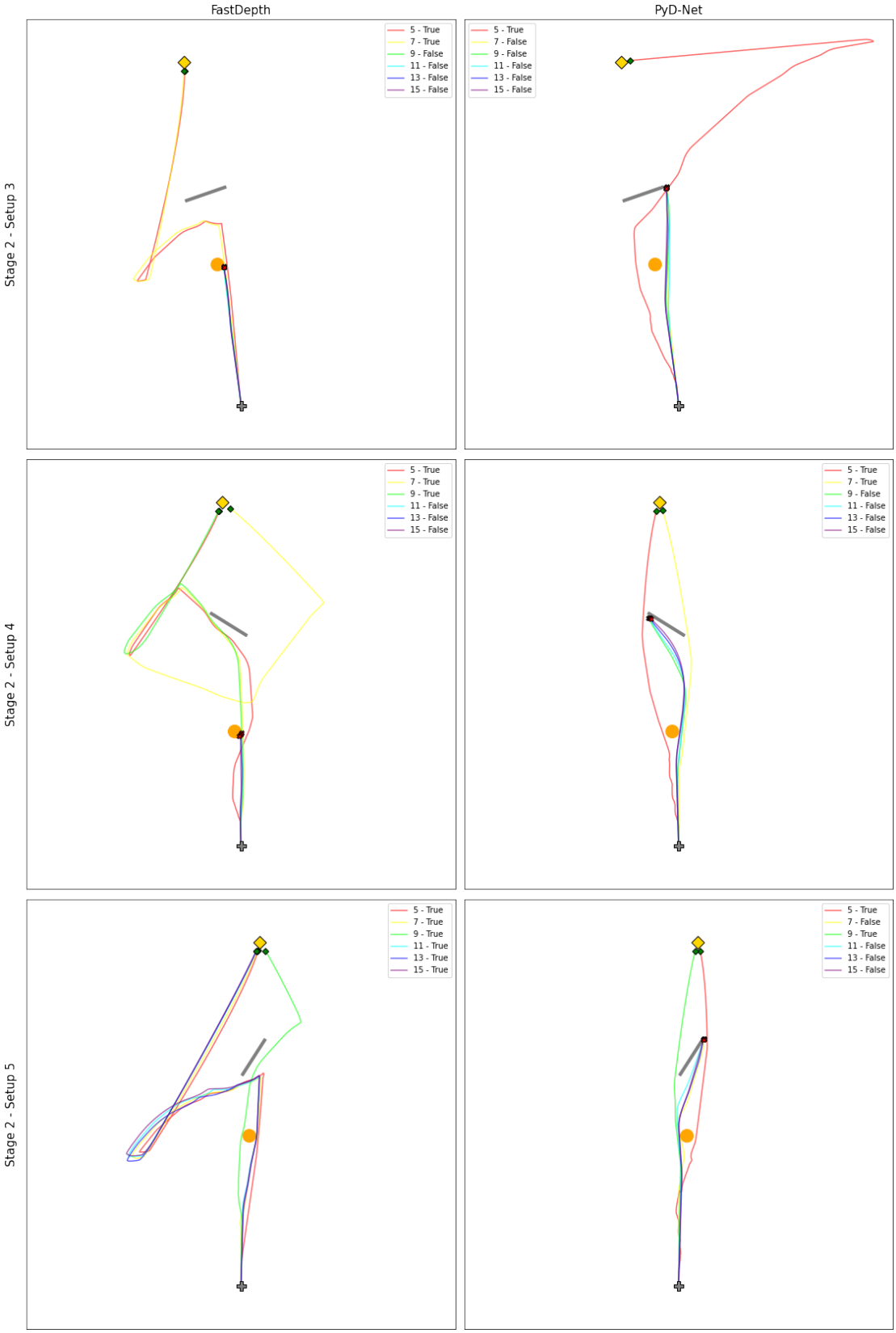

FastDepth
PyD-Net
Stage 1 - Setup 0
Stage 1 - Setup 1
Stage 1 - Setup 2
5 - True
7 - False
9 - True
11 - False
13 - False
15 - False
5 - True
7 - True
9 - True
11 - True
13 - True
15 - True
5 - True
7 - True
9 - True
11 - True
13 - True
15 - False
5 - False
7 - True
9 - True
11 - True
13 - True
15 - True
5 - False
7 - True
9 - True
11 - True
13 - True
15 - True
5 - True
7 - True
9 - False
11 - True
13 - True
15 - False

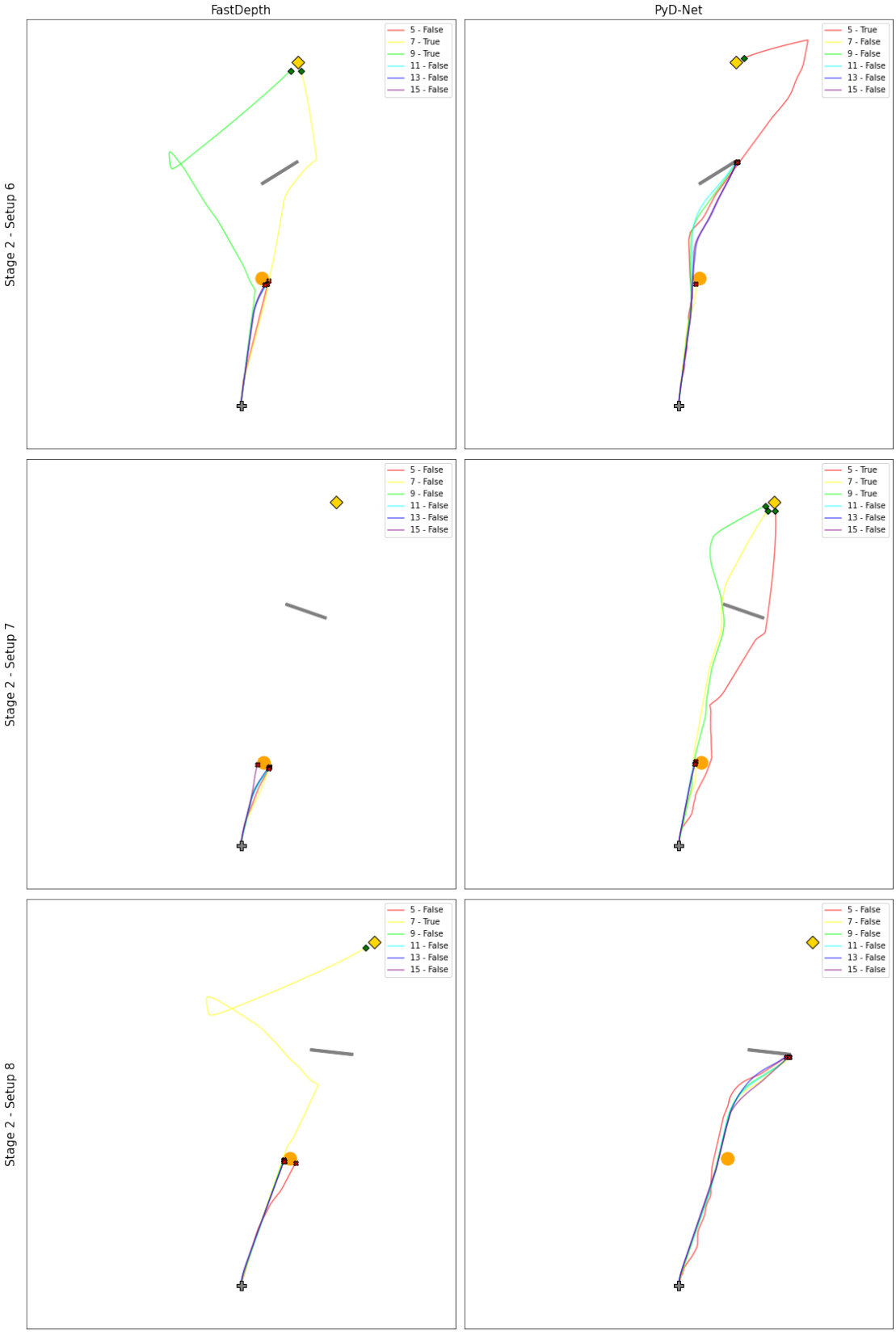

FastDepth
PyD-Net
Stage 1 - Setup 3
Stage 1 - Setup 4
Stage 1 - Setup 5
5 - True
7 - False
9 - True
11 - True
13 - False
15 - False
5 - True
7 - True
9 - False
11 - False
13 - False
15 - False
5 - True
7 - True
9 - True
11 - True
13 - True
15 - True
5 - True
7 - True
9 - False
11 - False
13 - False
15 - False
5 - True
7 - True
9 - True
11 - True
13 - True
15 - False
5 - True
7 - True
9 - False
11 - False
13 - False
15 - False

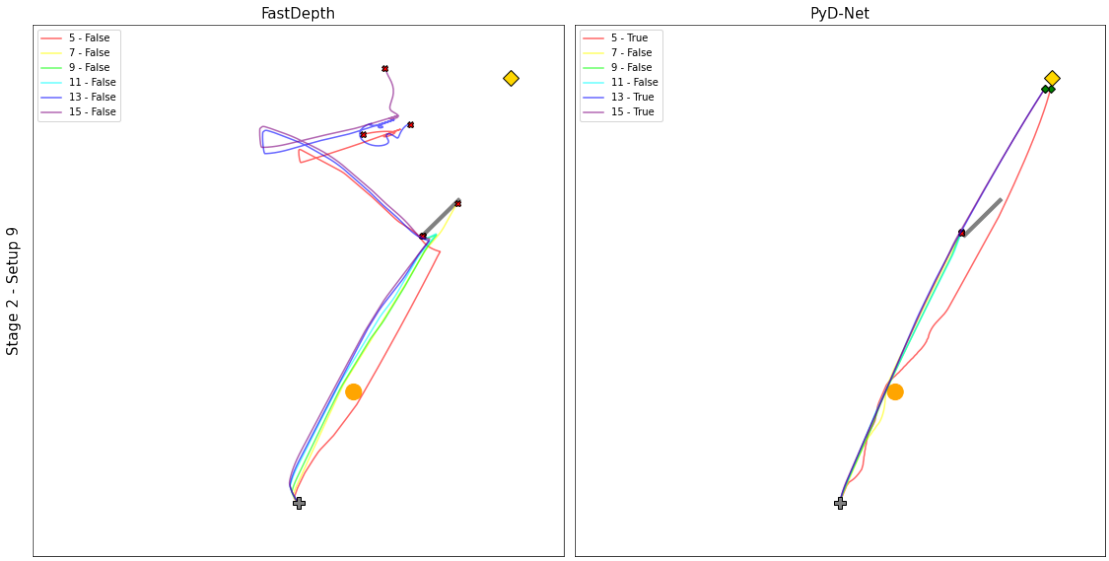

FastDepth
PyD-Net
Stage 1 - Setup 6
Stage 1 - Setup 7
Stage 1 - Setup 8
5 - True
7 - True
9 - False
11 - False
13 - False
15 - False
5 - True
7 - False
9 - True
11 - True
13 - False
15 - False
5 - True
7 - False
9 - False
11 - False
13 - False
15 - False
5 - True
7 - True
9 - False
11 - False
13 - False
15 - True
5 - True
7 - False
9 - False
11 - False
13 - False
15 - False
5 - True
7 - False
9 - False
11 - False
13 - False
15 - False

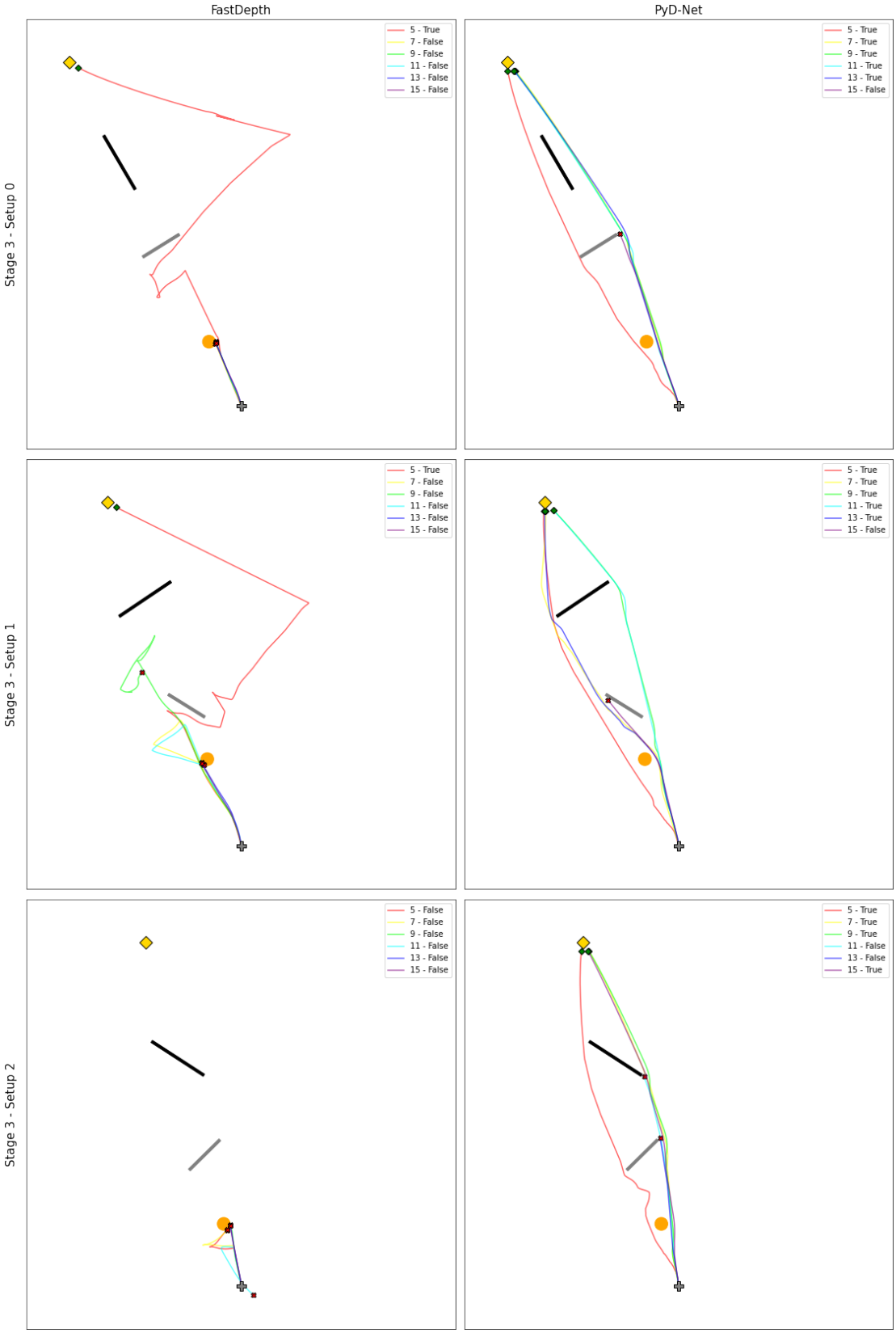

FastDepth
PyD-Net
Stage 1 - Setup 9
5 - True
7 - False
9 - False
11 - False
13 - False
15 - False
5 - True
7 - True
9 - True
11 - True
13 - False
15 - False

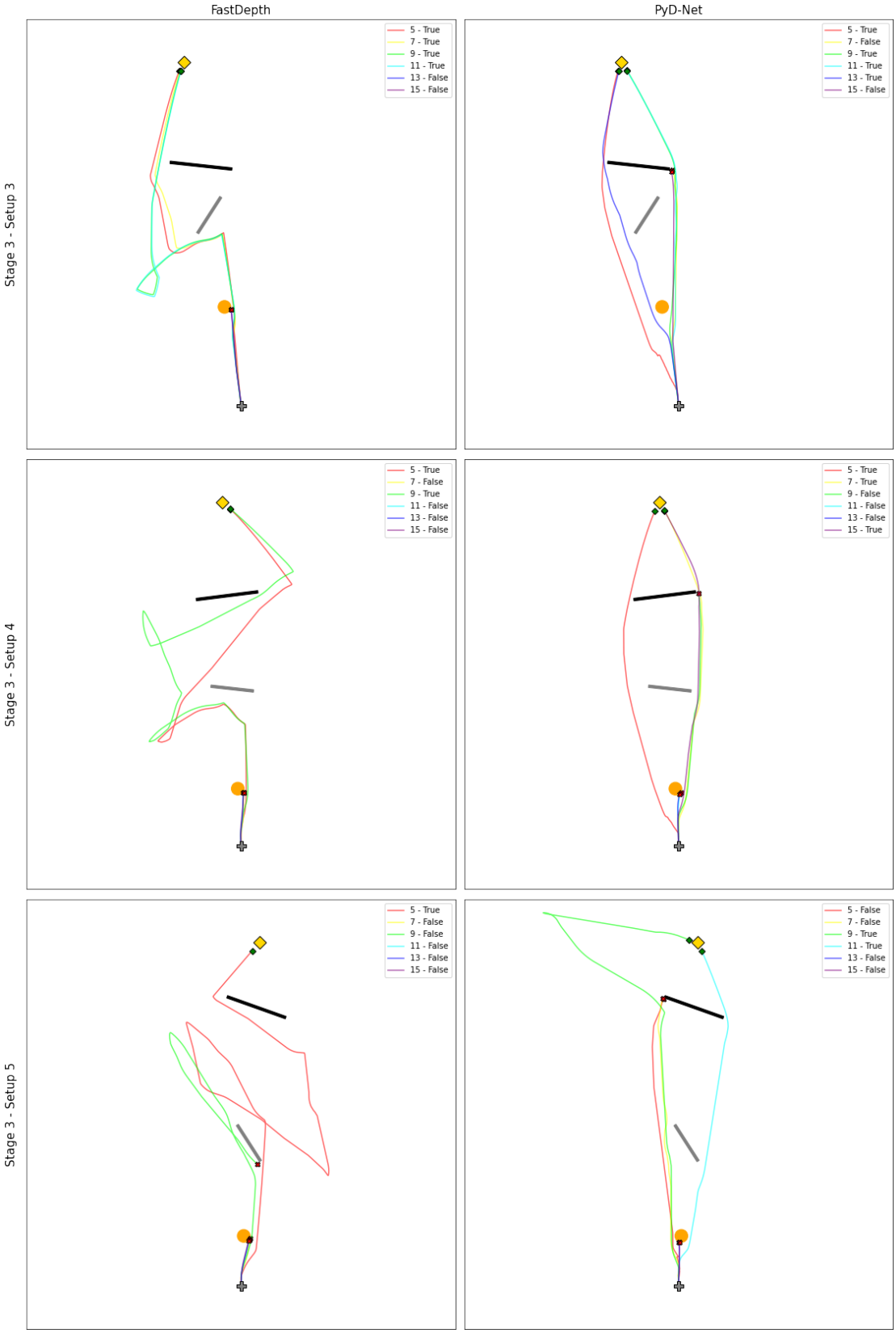

FastDepth
PyD-Net
Stage 2 - Setup 0
Stage 2 - Setup 1
Stage 2 - Setup 2
5 - False
7 - False
9 - False
11 - False
13 - False
15 - False
5 - True
7 - True
9 - False
11 - True
13 - True
15 - True
5 - False
7 - True
9 - False
11 - False
13 - False
15 - False
5 - True
7 - True
9 - True
11 - True
13 - True
15 - True
5 - False
7 - False
9 - False
11 - False
13 - False
15 - False
5 - True
7 - True
9 - False
11 - False
13 - False
15 - False

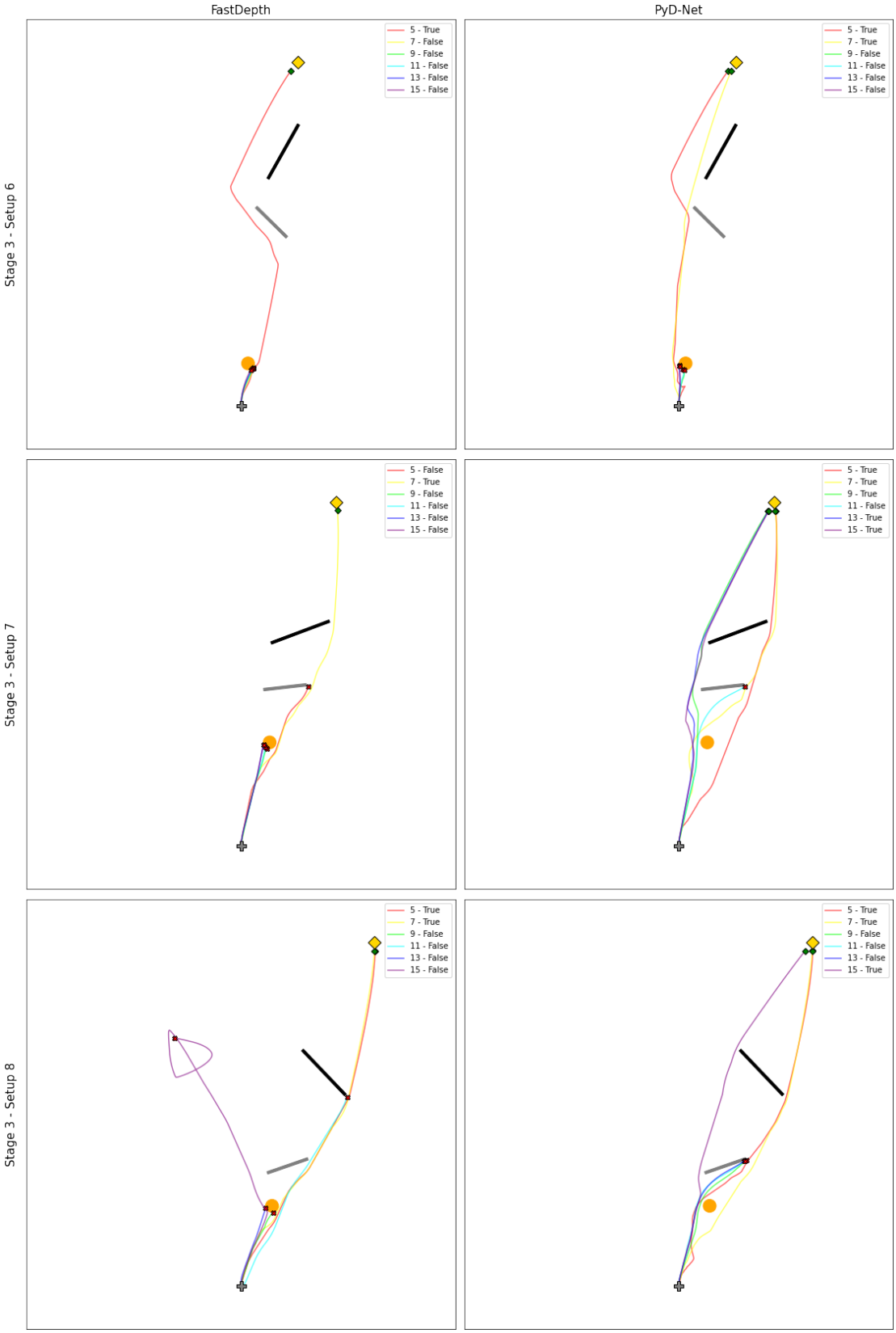

FastDepth
PyD-Net
Stage 2 - Setup 3
Stage 2 - Setup 4
Stage 2 - Setup 5
5 - True
7 - True
9 - False
11 - False
13 - False
15 - False
5 - True
7 - False
9 - False
11 - False
13 - False
15 - False
5 - True
7 - True
9 - True
11 - False
13 - False
15 - False
5 - True
7 - True
9 - False
11 - False
13 - False
15 - False
5 - True
7 - True
9 - True
11 - True
13 - True
15 - True
5 - True
7 - False
9 - True
11 - False
13 - False
15 - False

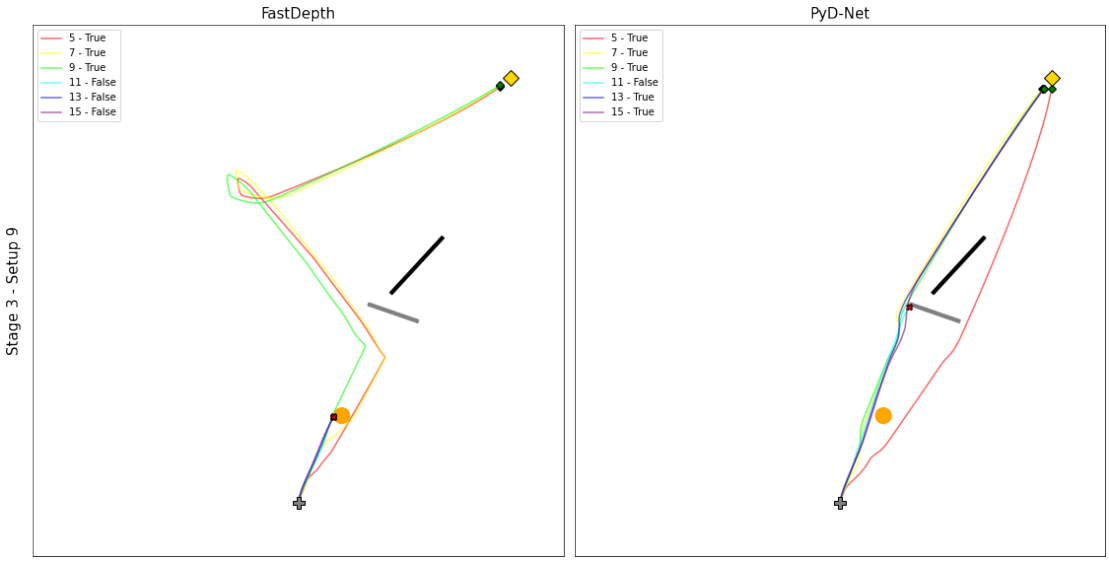

FastDepth
PyD-Net
Stage 2 - Setup 6
Stage 2 - Setup 7
Stage 2 - Setup 8
5 - False
7 - True
9 - True
11 - False
13 - False
15 - False
5 - True
7 - True
9 - False
11 - False
13 - False
15 - False
5 - False
7 - False
9 - False
11 - False
13 - False
15 - False
5 - True
7 - True
9 - True
11 - False
13 - False
15 - False
5 - False
7 - True
9 - False
11 - False
13 - False
15 - False
5 - False
7 - False
9 - False
11 - False
13 - False
15 - False


FastDepth
PyD-Net
Stage 2 - Setup 9
5 - False
7 - False
9 - False
11 - False
13 - False
15 - False
5 - True
7 - False
9 - False
11 - False
13 - True
15 - True


FastDepth
PyD-Net
Stage 3 - Setup 0
Stage 3 - Setup 1
Stage 3 - Setup 2
5 - True
7 - False
9 - False
11 - False
13 - False
15 - False
5 - True
7 - True
9 - True
11 - True
13 - True
15 - False
5 - True
7 - False
9 - False
11 - False
13 - False
15 - False
5 - True
7 - True
9 - True
11 - True
13 - True
15 - False
5 - False
7 - False
9 - False
11 - False
13 - False
15 - False
5 - True
7 - True
9 - True
11 - False
13 - False
15 - True


FastDepth
PyD-Net
Stage 3 - Setup 3
Stage 3 - Setup 4
Stage 3 - Setup 5
5 - True
7 - True
9 - True
11 - True
13 - False
15 - False
5 - True
7 - True
9 - True
11 - True
13 - True
15 - False
5 - True
7 - False
9 - True
11 - False
13 - False
15 - False
5 - True
7 - True
9 - False
11 - False
13 - False
15 - True
5 - True
7 - False
9 - False
11 - False
13 - False
15 - False
5 - False
7 - False
9 - True
11 - True
13 - False
15 - False


FastDepth
PyD-Net
Stage 3 - Setup 6
Stage 3 - Setup 7
Stage 3 - Setup 8
5 - True
7 - False
9 - False
11 - False
13 - False
15 - False
5 - False
7 - True
9 - False
11 - False
13 - False
15 - False
5 - True
7 - True
9 - True
11 - False
13 - True
15 - True
5 - True
7 - True
9 - False
11 - False
13 - False
15 - False
5 - True
7 - True
9 - False
11 - False
13 - False
15 - True


FastDepth
PyD-Net
Stage 3 - Setup 9
5 - True
7 - True
9 - True
11 - False
13 - False
15 - False
5 - True
7 - True
9 - True
11 - False
13 - True
15 - True