

Vision-Based Navigation and Deep-Learning Explanation for Autonomy

Sandeep Konam

CMU-RI-TR-17-27

May 2017

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:

Manuela Veloso, Co-chair
Stephanie Rosenthal, Co-chair
Sebastian Scherer
Danny Zhu

*Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Robotics.*

Keywords: Unmanned aerial vehicles, Autonomy, Vision, Navigation, Deep-learning, Interpretability

Abstract

In this thesis, we investigate vision-based techniques to support robot mobile autonomy in human environments, including also understanding the important image features with respect to a classification task. Given this wide goal of transparent vision-based autonomy, the work proceeds along three main fronts. Our first algorithm enables a UAV to visually localize and navigate with respect to CoBot, a ground mobile robot, in order to perform visual search tasks. Our approach leverages the robust localization and navigation capabilities of CoBot while allowing the UAV to search for the object of interest in locations that CoBot cannot access. Second, to enable safe UAV navigation using its monocular camera, we contribute a deep learning based perception system to avoid obstacles in real-time. We demonstrate that using our system, UAVs can navigate safely in various challenging environments. Finally, we address our goal towards justification of vision-based decisions. We investigate an explanation technique to understand the predictions of a deep learning based image classifier. We contribute the Automatic Patch Pattern Labeling for Explanation (APPLE) algorithm for analyzing a deep network to find neurons that are ‘important’ to the network classification outcome, and for automatically labeling the patches of the input image that activate these important neurons. We investigate several measures of importance for neurons and demonstrate that our technique can be used to gain insight into how a network decomposes an image to make its classification. The performance of each of these contributions is demonstrated through experimental results.

Acknowledgments

Firstly, I express my sincere gratitude to my advisors Prof. Manuela Veloso and Dr. Stephanie Rosenthal for the continuous encouragement, guidance, and contribution of ideas for the development of this research. I thank my other thesis committee members Sebastian Scherer and Danny Zhu for being helpful and providing feedback on my work.

I'm thankful to Shichao Yang and Ian Quah for collaborating on critical aspects of this work. I thank Rick Goldstein for constructing the UAV landing base required for a part of this work. I thank Raulcezar Alves and Srikanth Malla for helping me record algorithm demonstration videos.

I thank Sai Prabhakar for all the insightful discussions. I am grateful to the members of the CORAL group for their input on my research as well as the presentation of my work. I would like to thank the rest of the RI community for many great memories.

Finally, I thank my family members and friends for supporting my academic endeavors, I could not have done it without them.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Contributions of the thesis	1
1.3	Outline of the thesis	2
2	UAV and CoBot coordination for indoor object search tasks	3
2.1	Related Work	3
2.2	Multi-Robot Coordination for Object Search Tasks	4
2.2.1	CoBot Capabilities	4
2.2.2	ARDrone Capabilities	4
2.2.3	Multi-Robot Coordination Task	4
2.3	Vision-Based Moving Target Navigation	5
2.3.1	Image Coordinate System	5
2.3.2	Imagining Beyond the Image Frame	7
2.3.3	Moving Target Navigation Algorithms	7
2.3.4	Forward Search and hover above marker	7
2.3.5	Forward Search, Returning Home, Landing	7
2.3.6	Coordination between CoBot and UAV to perform visual search	8
2.4	Experiments	9
2.4.1	Navigation to marker in vision frame	9
2.4.2	Forward Search and hover above marker	9
2.4.3	Forward Search, Returning Home, Landing	9
2.4.4	Coordination between CoBot and UAV to perform visual search task	11
2.5	Future work	12
3	Obstacle avoidance for UAVs using deep learning	13
3.1	Related Work	13
3.2	Visual Perception	14
3.2.1	Dataset	14
3.2.2	Intermediate Perception - depth and surface normal	15
3.2.3	Trajectory prediction	16
3.3	Experiments	17
3.3.1	Training and testing	17
3.3.2	NYUv2 dataset Evaluation	17

3.3.3	Other Public Indoor dataset Evaluation	19
3.3.4	Quadrotor simulation flight	19
3.3.5	Real quadrotor flight	22
3.4	Future work	22
4	Automatic Patch Pattern Labeling for Explanation (APPLE)	25
4.1	Related Work	25
4.2	Approach	26
4.2.1	High importance Neurons	26
4.2.2	High Importance Patches	27
4.2.3	Patch classifier	28
4.2.4	Putting it all together	28
4.3	Experiments	28
4.3.1	CNN Classifier	29
4.3.2	Patch Classifier	29
4.3.3	Demonstration of APPLE	29
4.3.4	Evaluation of Important Patches	30
4.3.5	How does APPLE apply to UAVs?	34
4.4	Future work	35
5	Conclusion	37
	Bibliography	39

List of Figures

- 2.1 CoBot carrying ARDrone. 5
- 2.2 (a) Pink marker as seen through bottom camera. (b) The black rectangle represents the image captured by bottom camera of drone. One of the colored markers (pink marker) is in the field of view where as the other (red marker) can be reached through forward search.(c) Sample Coordinates corresponding to the four primary directions and center of the drone marked on a 640×360 image. . . 6
- 2.3 (a) Through error minimization, UAV has reached center of the marker and hovers above it.(b) UAV landing on CoBot after performing forward search and detecting the marker.(c) UAV navigating forward searching for the marker. Above three parts include sequential captures from the bottom camera of UAV. 8
- 2.4 Plots indicating standard deviation and mean of the observed data from experiments for different tasks 10
- 2.5 Approximate sketch of the path traversed by drone in two scenarios: (a) shows unaffected navigation during forward and backward motion, (b) shows the effect of drift on the course of navigation. 11
- 2.6 Coordination between CoBot and UAV to perform visual search task. (a) Drone taking off from the CoBot, (b)-(c) Drone performing forward search for target marker, (d) Drone hovering above marker after reaching it, (e) Drone performing backward search for CoBot, (f) Drone landing above CoBot. 12
- 3.1 Method Overview. Instead of directly predicting path from RGB image, we propose intermediate perception: first predict depth and surface normal, which are closely related to 3D obstacles, then predict the path from the depth and normal. Both steps utilize CNNs. 14
- 3.2 Generating ground truth path label. Using the provided depth image, a 3D point cloud map is built. Then the best path in red is selected based on the Euclidean obstacle cost and path smoothness cost. 15
- 3.3 Proposed model architecture to predict path from depth and surface normal. It has two branches at the beginning to receive two input information. The prediction result is a label within five classes. For predicting depth and normal images, we use the model of [7]. 17

3.4	Example of path prediction on <i>NYUv2</i> dataset. The input is only RGB image. Our method and method using only RGB images for prediction are shown in the last two columns in red color. From left to right: RGB image, predicted depth image, predicted surface normal, and predicted paths. In the top image, two predictions are similar. In the bottom images, our method performs better.	18
3.5	Some prediction examples in <i>Ram-lab</i> dataset images.	20
3.6	Some scenarios of the dynamic simulation. White area represents obstacles and red curve is the quadrotor path.	21
3.7	Some prediction examples in gazebo simulations.	21
3.8	Top view of path prediction in gazebo simulations. The red arrow represents the robot pose, the blue line is the predicted trajectory. The white areas in the image represents the obstacles.	22
3.9	(a) Real flight scenes including curved corridor, front obstacles and corridor following (b) Eight prediction examples from quadrotor’s view on the fly. For each image, the path image below it shows the predicted path. More results could be found in the supplementary video.	23
4.1	Pipeline of our APPLE algorithm : a) ranking neurons as per measures described in Section 3.1, b) identifying image patches corresponding to the top 5 neurons from (a), c) our patch classifier classifies the image patches from (b), only top-5 classifications are shown here.	26
4.2	Sample training data for the Patch classifier.	30
4.3	(a): Input image. (b): High Importance Patches selected by APPLE algorithm using <i>Activation Matrix Sum</i> measure.	31
4.4	APPLE sorts the labeled patches by confidence to present to a human in order to explain the CNN’s image classification. Two example images are shown with their important patches selected using <i>Activation Matrix Sum</i> measure.	32
4.5	Comparing weakly supervised localization between a) APPLE and b) CAM. In APPLE, red boxes indicate layers 3 and 4, green indicates layers 5 - 7 and blue indicates layers 8 and 9. On the CAM images, the heatmap visualizes its important pixels.	33
4.6	APPLE sorts the labeled patches by confidence to present to a human in order to explain the CNNs image classification and corresponding action (important patches are selected using <i>Weight Matrix Sum</i> measure).	34
4.7	Weakly supervised localization of APPLE (red boxes indicate layers 3 and 4, green indicates layers 5 - 7 and blue indicates layers 8 and 9.)	35

List of Tables

3.1	Trajectory label distribution on <i>NYUv2</i> dataset.	16
3.2	Comparison of path prediction on <i>NYUv2</i> dataset.	18
3.3	Confusion matrix for paths prediction on <i>NYUv2</i>	19
3.4	Comparison of path prediction on <i>Ram-lab</i> datasets.	20
4.1	Evaluation of important patches	32

Chapter 1

Introduction

1.1 Motivation

The ability to sense the surrounding environment is a fundamental requirement for any autonomous system. Sensors may be divided into two classes: internal state sensors, such as accelerometers or gyroscopes, which provide internal information about the robot’s movements, and external state sensors, such as lasers, infrared sensors, sonars, and visual sensors, which provide external information about the environment [16]. Compared to sensors such as lasers [1], stereo cameras [28], and RGB-D depth cameras [11], a monocular camera is small, low weight, low-cost, and relatively less power-consuming. Therefore, in this thesis, we focus on vision-based techniques to support robot mobile autonomy. Specifically, we contribute two vision-based algorithms to enable UAVs to perform a search task and to avoid obstacles in real-time.

Recently, there has been a wide-spread adoption of various deep-neural network architectures for vision-based robotic applications like unmanned aerial vehicle (UAV) navigation because of the empirical success these architectures have seen in various image classification tasks. However, deep learning models are often thought of as ‘black boxes’ in reference to the difficulties of tracing a prediction back to important features to understand how an output was arrived at. When the robots operate in human environments, their lack of interpretability is a major problem for human users to understand the deep-learning-based robot’s actions. To this end, we contribute a technique for explaining the deep-learning algorithms to humans in their environments.

1.2 Contributions of the thesis

In this thesis, we investigate vision-based techniques to support robot mobile autonomy in human environments, including also understanding the important image features responsible for image classification. Given this wide goal of transparent vision-based autonomy, the work includes three main contributions:

1. We propose a visual navigation technique that enables the Parrot ARDrone 2.0 to localize with respect to a CoBot [34], a ground service robot, to perform search tasks. CoBots autonomously localize and navigate in our office environment, while effectively avoiding obstacles using a modest variety of sensing devices, including a vision camera, a Kinect

depth-camera, and a small Hokuyo LIDAR [3]. However, our CoBots do not have the capability to navigate small spaces to view the objects on desks and other tall surfaces, and as a result, they fail to efficiently search for objects of interest without human assistance. Our approach leverages the robust localization and navigation capabilities of CoBot and the ARDrone’s capability to maneuver easily through indoor environments and search for object of interest.

2. We propose a deep-learning-based perception system for UAVs to avoid obstacles in real-time. We first propose a new method to get ground truth labels from RGB-D images automatically without human demonstrations. Then we propose a two-stage convolutional neural network (CNN) for obstacle avoidance. The first stage predicts the depth and surface normal from images, which are two important geometric properties related to 3D obstacle avoidance. The second stage predicts a path from the depth and normal maps using another CNN model. We demonstrate that, using our system, UAVs can navigate safely in various challenging environments.
3. We propose an algorithm to label the features of an image that the network focuses on in order to explain why the network made its prediction. We accomplish this by analyzing the neurons that are most important to the output classification of an image as well as the patterns that activate those neurons. Our proposed approach, Automatic Patch Pattern Labeling for Explanation (APPLE), first automatically analyzes the signal propagation through each layer of the network in order to find neurons that contribute highly to the signal in subsequent layers. Then, it deconvolves the important neurons at each layer to determine the parts or patches of the image that these neurons use as their input. Finally, our algorithm automatically labels the image patches using a separately-trained classifier and ranks the neurons based on confidence.

1.3 Outline of the thesis

This thesis is organized as follows. Chapter 2 introduces CoBot collaboration with the off-the-shelf Parrot ARDrone 2.0 unmanned aerial vehicle (UAV) to perform service-based object search tasks. In Chapter 3, we propose a CNN-based navigation system with intermediate perception to predict trajectory. In Chapter 4, we propose APPLE (Automatic Patch Pattern Labeling for Explanation), an algorithm to label the features of an image that the network focuses on in order to explain why the network made its prediction. Finally in Chapter 5, we summarize the contributions of this thesis, and the capabilities of the developed algorithms.

Chapter 2

UAV and CoBot coordination for indoor object search tasks

In this chapter, we propose a vision-based moving target approach for the navigation of ARDrone in which the drone uses a camera-based coordinate system to track the direction the robot and object has moved, find and hover above objects, and reverse those trajectories to return to its starting location under uncertainty. This approach requires little computation and yet allows the drone to perform its search task from any location in any environment. In Section 2.4, we demonstrate that our algorithm efficiently finds objects in our environment during search tasks and can return to its starting location after it is finished searching.

2.1 Related Work

Most of the current visual-servoing techniques are computationally expensive and precision is often traded with real-time computation on UAVs. The ARDrone was previously used for visual SLAM based navigation [8], autonomous navigation of hallways and stairs [2] and reactive avoidance in natural environments [26]. Inferring 3D structure from multiple 2D images is challenging because aerial vehicles are not equipped with reliable odometry and building a 3D model is computationally very expensive. [2] instead compute perspective cues to infer about the 3D environment. Unfortunately, most indoor environments don't possess distinct corner-type features to provide the desired perspective cues. Inspired by the accuracy of visual tracking methods, we designed *vision-based moving target navigation algorithm* for drones that does not rely on any environment features and is computationally minimal. In our approach, if an object is visible in the robot's visible range, the robot aims to minimize the distance between itself and the center of the object. If no object is visible, the algorithm generates way-points considered equivalent to virtual markers outside of its visible range, using which the robot navigates to it in a similar way. Using the knowledge of the robot's current speed and travel time, the robot navigates in any arbitrary search pattern by computing where virtual markers should be placed.

2.2 Multi-Robot Coordination for Object Search Tasks

Our CoBot service robots have been deployed in our environment for many years performing tasks such as delivering messages, accompanying people to meetings, and transporting objects to offices [34]. However, our CoBots do not have capability to navigate confined space such as offices, to view the objects in them, and as a result they require human assistance to find required objects in search tasks. We propose a multi-robot coordination in which a second platform - the Parrot ARDrone - can perform the object search task for CoBot, while at the same time relying on CoBot for its localization and navigation. We describe the robot's capabilities and the joint task before focusing on the remaining challenge of drone localization and navigation.

2.2.1 CoBot Capabilities

CoBot is a four-wheeled omni-directional robot, equipped with a short-range laser range-finder sensor and a depth camera for sensing (Figure 2.1). An on-board tablet provides significant computation for the robot to localize and navigate autonomously, as well as a method of communication with humans in the environment. The CoBot robots can perform multiple classes of tasks, as requested by users through a website [35], in person through speech [17], or through the robot's touch screen. All tasks can be represented as pick-up and delivery tasks of objects or people. Because CoBot is not able to navigate in confined space safely, it lacks the ability to visually search for objects to pick up and drop off, instead relying on humans.

2.2.2 ARDrone Capabilities

Parrot ARDrone 2.0 has a 3-axis accelerometer, 3-axis gyroscope, pressure sensor, ultrasonic sensors, front camera and a bottom/vertical camera. It relies on WiFi to transfer all of its sensor data onto a larger computational platform. It sends its video feed for simple object detection within the local camera frame (e.g., color thresholding for colored marker detection). Velocity commands are sent back to the drone for vision-based navigation.

2.2.3 Multi-Robot Coordination Task

Considering the capabilities of CoBot and ARDrone mentioned previously, we notice that the ARDrone can perform object search tasks with small computational loads but without reliable localization while CoBot can accurately localize and navigate in its safe regions. We propose coordination between them to effectively search for an object of interest in an indoor environment. In particular, CoBot carries the drone to a region of interest to search and then the drone can search locally by tracking its relative motion after taking off from CoBot. After it finishes its search, it can reverse the trajectory or perform another search to land on CoBot and move to another location.

For the search task to be performed efficiently, the drone should be able to navigate in its local coordinate space. Lack of reliable camera-based localization algorithms for the resource-constrained ARDrone forced us to opt for visual-servoing techniques. While performing the search task, the image provided by the bottom camera of the drone can only provide information



Figure 2.1: CoBot carrying ARDrone.

about presence or absence of the marker being searched for, but doesn't provide any cues that facilitate search. We contribute our vision-based moving target navigation algorithm to overcome the challenges of localizing and navigating without any visual cues.

2.3 Vision-Based Moving Target Navigation

In moving target navigation algorithms, a robot continuously aims to minimize the distance between its current location and another target point in its coordinate space. Maintaining the target in the same place over time allows the robot to navigate directly to it (i.e., for hovering over an object of interest). By moving the target point in a trajectory at a constant velocity, the robot follows the same trajectory. We use this moving target navigation algorithm in order for the ARDrone to search the environment and track a marker when it finds one, noting that the challenge of this algorithm is determining the local coordinate space to move the point in. We next describe our coordinate space for the moving target algorithm.

2.3.1 Image Coordinate System

We use ARDrone's bottom camera's coordinate frame as the local frame of reference. The ARDrone's image is represented as pixels in its bottom camera time (640×360 in this work). When a marker is found in an image (Figure 2.2a), the computational platform sends velocity commands proportional to the distance from the center of the robot (image) to the center of the

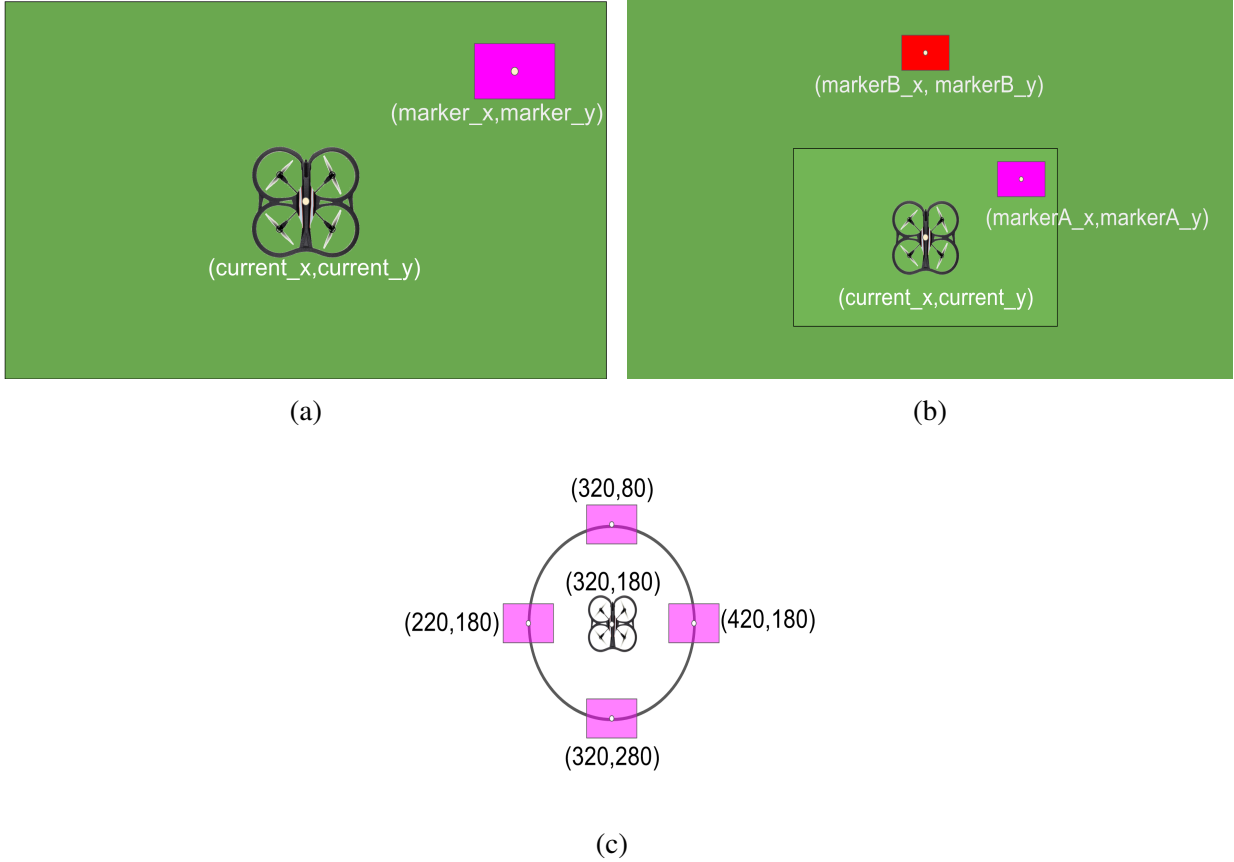


Figure 2.2: (a) Pink marker as seen through bottom camera. (b) The black rectangle represents the image captured by bottom camera of drone. One of the colored markers (pink marker) is in the field of view where as the other (red marker) can be reached through forward search. (c) Sample Coordinates corresponding to the four primary directions and center of the drone marked on a 640×360 image.

target marker:

$$error_x = (target_x - current_x) \quad (2.1)$$

$$error_y = (target_y - current_y) \quad (2.2)$$

$$vel_x = k \cdot error_y \quad (2.3)$$

$$vel_y = k \cdot error_x \quad (2.4)$$

In the above equations, $error_x$ refers to the difference in x-coordinates of the UAV's current position ($current_x$) and that of virtual marker's center ($target_x$). Similar notation applies to $error_y$. vel_x refers to the linear velocity in x-direction and vel_y refers to the linear velocity in y-direction. k is empirically tuned to be 0.0005.

Figure 2.2c demonstrates markers in the 4 cardinal directions of the robot's image frame. Since the marker exists in the image coordinate system, the error between the center of marker and the position of UAV tends to reduce as the UAV approaches the marker. Since the velocity is directly proportional to the error, the UAV starts hovering once the error becomes zero.

2.3.2 Imagining Beyond the Image Frame

When the target marker is not in the image frame, we would like the robot to search for it using moving target navigation. By referring to pixel locations outside of the image frame (Figure 2.2b), the robot imagines where the target should be and navigates towards it. Imagined coordinates could be based on knowledge of direction in which object went out of view or generated corresponding to a trajectory. In the current work, we generate coordinates representing imagined markers based on the required trajectory robot has to follow. However since it is an imagined marker at a constant distance from the current position of the robot, the error remains constant and the robot maintains a constant velocity using the equations 2.1-2.4. For instance, if we want the robot to perform search operation in a square path, we generate coordinates corresponding to imaginary markers at the four corners of a square. By creating trajectories of imagined markers outside (or even inside) of the image area, the robot navigates using those coordinates and eventually detects object of interest.

2.3.3 Moving Target Navigation Algorithms

In order to demonstrate the applicability of the moving target algorithm, we designed several search trajectories as finite state machines in addition to hovering behavior over a marker in the vision frame. We describe each in turn. Then, we will show results for experiments for each algorithm.

Navigation to marker in vision frame

Navigation to a marker detected in the field of view of the UAV relies entirely on the image coordinate system. After the marker is detected, its center becomes the target coordinate for the UAV. The UAV navigates towards the center of the marker with velocity proportional to the error between its current position and the target coordinate.

2.3.4 Forward Search and hover above marker

As depicted in Figure 2.2c, (320,80) refers to the center of imagined marker corresponding to forward direction. As the UAV navigates forward, the video feed from the bottom camera is used to search for the marker. Since the marker is distinctively colored from the background, a thresholding algorithm suffices to let the UAV know if it sees the marker in its field of view. Once the marker is detected, UAV navigates towards the marker to hover over it.

2.3.5 Forward Search, Returning Home, Landing

Forward search is performed as mentioned previously. After detecting the marker, the UAV starts navigating backward to the home (place where it started search from) by reversing the trajectory of imagined target points. Note that in this case, reversing the trajectory is equivalent to the forward search problem where home is another distinctively colored marker. After detecting the marker corresponding to home, drone navigates to the center of marker using the image

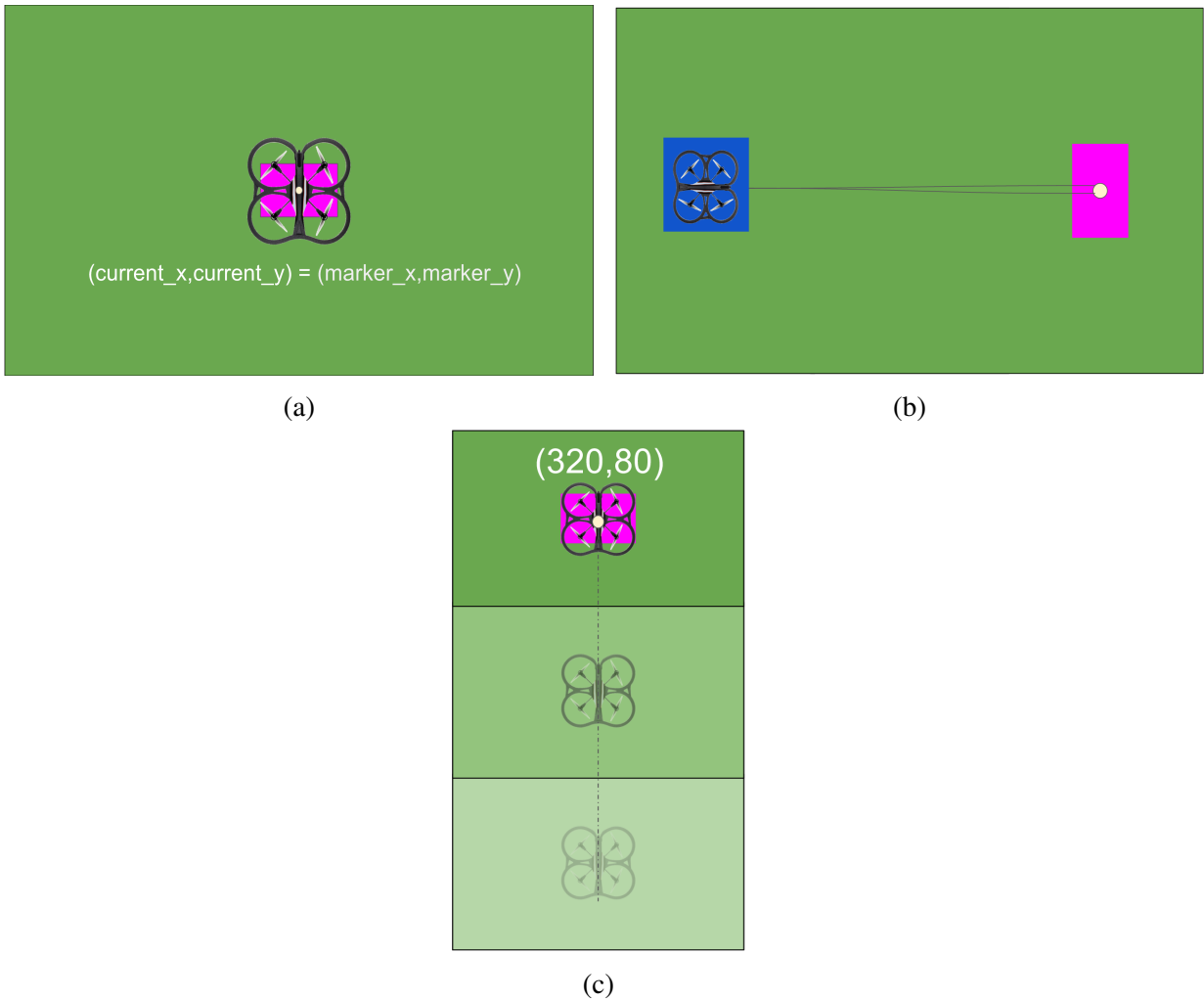


Figure 2.3: (a) Through error minimization, UAV has reached center of the marker and hovers above it.(b) UAV landing on CoBot after performing forward search and detecting the marker.(c) UAV navigating forward searching for the marker. Above three parts include sequential captures from the bottom camera of UAV.

coordinate system. It lands on the home marker after centering itself using error-minimization between the center of marker and its own position.

2.3.6 Coordination between CoBot and UAV to perform visual search

In this task, CoBot carries the UAV down the hallway to the destined search area. The UAV performs forward search, returns home, and lands on CoBot. As shown in Figure 2.3b, the blue marker represents the CoBot. The ARDrone performs two search operations: one for the search object (pink marker) and other for the CoBot (blue marker).

2.4 Experiments

We evaluated the performance of our proposed vision-based moving target navigation algorithm on ARDrone 2.0. through a series of experiments presented in this section. ARDrone has 1GHz ARMv7 Processor rev 2 (v7l) with 1Gbit DDR2 RAM and a VGA downward camera of 360p resolution at 30fps. Images captured are of 640x360 resolution. It is controlled via Wi-Fi through a laptop with Intel Core i7-6700HQ CPU and 16 GB RAM.

Each of the experiments presented in this section correspond to the tasks detailed in Section 4. We have performed each experiment 20 times to test the consistency of the proposed algorithm. Mean and standard deviation of the values corresponding to time taken for each iteration is calculated for each of the experiments and is depicted in the charts. Results suggest that the proposed algorithm is reliable for search tasks.

2.4.1 Navigation to marker in vision frame

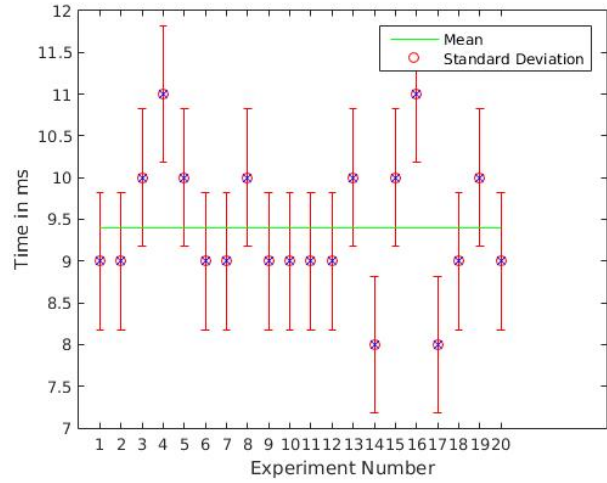
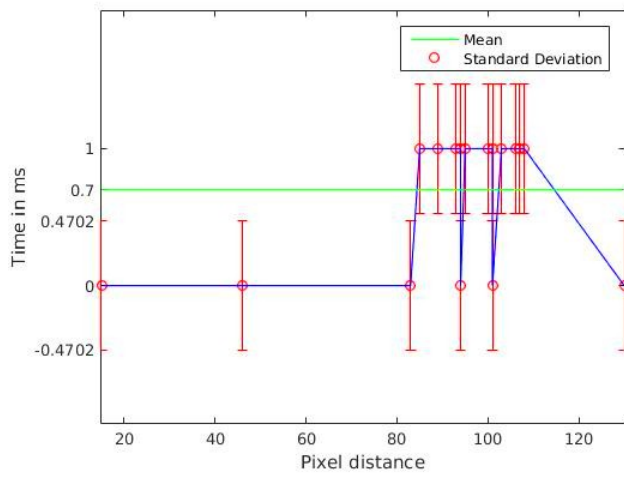
In this experiment, drone navigates to the center of the marker corresponding to the platform from where it takes off. Though drone takes off from the same place in all experiments, due to the unavoidable drift associated with the drone while taking off, pixel distance between the center of marker and the position of drone (center of the image) changes as represented on the x-axis of plot shown in the Figure 2.4a. Due to the precision of timestamp being 1ms, it is possible that 0.9ms is considered as 0ms. 14 out of 20 times, drone takes 1 ms and 6 out of 20 times, it takes 0 ms to reach the center of marker. As depicted in the Figure 2.4a, mean is 0.7ms and standard deviation is 0.4702ms.

2.4.2 Forward Search and hover above marker

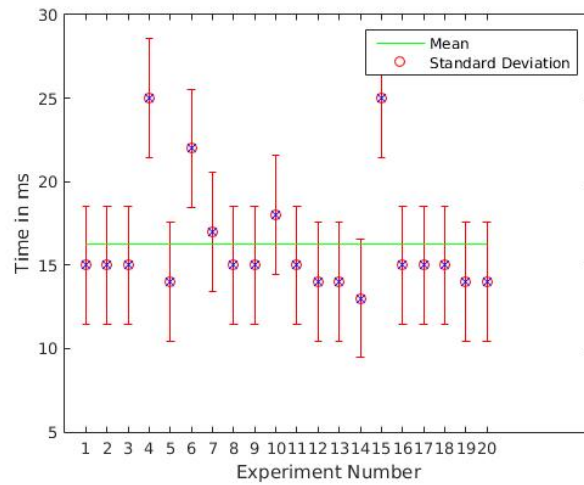
In this experiment, a marker is placed 2.0 meters in front of the drone. The drone performs forward search and hovers above the marker once it reaches within a threshold of 50 units of pixel distance from the center of marker. Each experiment is performed 20 times. Results show standard deviation of 0.8208ms with a mean of 9.40ms as depicted in Figure 2.4b.

2.4.3 Forward Search, Returning Home, Landing

In this experiment, drone performs forward search and returns back to the home after detecting the marker which is placed 2.0 meters in front of the drone. Over 20 experiments, results show a standard deviation of 3.5522ms and mean of 16.2500ms as depicted in the Figure 2.4c. For 17 times, time reported is similar with a standard deviation of 1.1663, but there are two cases where the time taken is 25 ms and one case where it is 22 ms, including which increased standard deviation to 3.5522. Through the data log, it is observed that in those three cases, the drone drifted after detecting the marker and required additional time to return to the home. Figure 2.5a shows approximate sketch of the path followed by the drone in cases where similar time was consumed, where Figure 2.5b shows path that looks more widened near the marker due to the drift of the drone.



(a) Task1: Navigation to marker in vision frame. (b) Task2: Forward Search and hover above marker.



(c) Task3: Forward Search, Returning Home, Landing.

Figure 2.4: Plots indicating standard deviation and mean of the observed data from experiments for different tasks

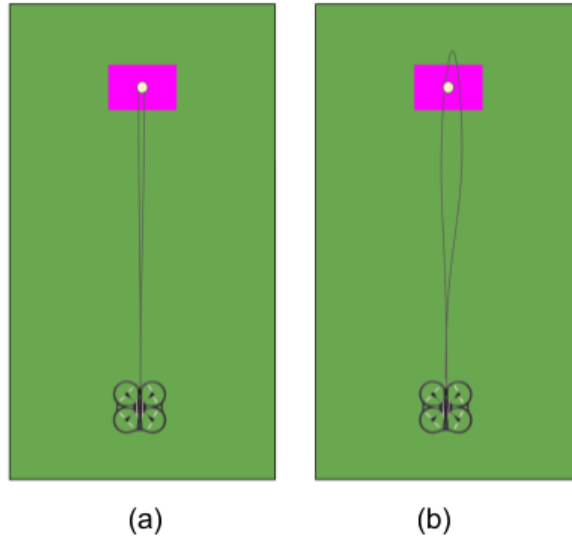


Figure 2.5: Approximate sketch of the path traversed by drone in two scenarios: (a) shows unaffected navigation during forward and backward motion, (b) shows the effect of drift on the course of navigation.

2.4.4 Coordination between CoBot and UAV to perform visual search task

We bring together all the above building blocks and accomplish the visual search task by establishing coordination between CoBot and UAV. As illustrated in Figure 2.6, CoBot carries the drone using its localization and navigation capabilities to the desired search area. The drone takes off from the CoBot as shown in Figure 2.6a and starts navigating forward as shown in Figure 2.6b. As it navigates forward, the drone simultaneously searches for the pink marker as shown in Figure 2.6c. Once it locates the marker in its field of view, it navigates to the center of marker through error minimization. Figure 2.6d shows the drone hovering above the marker. After reaching the center of marker, drone starts navigating backwards to the CoBot as shown in Figure 2.6e. As it navigates backwards, it keeps searching for the blue marker indicative of the CoBot. Once the blue marker is detected, the drone lands on the CoBot as shown in Figure 2.6f.

To summarize, our proposed vision-based moving target navigation algorithm utilizes visual markers for location feedback, allowing it to recover from drift problems frequently encountered by UAVs that do not use absolute positioning. In particular, we utilize the drift as part of the target search navigation and utilize the symmetry in the robot to simply playback the path assuming that the drift will be similar on the way back to CoBot. Additionally, we note that the CoBot and UAV multi-robot collaborative solution actually reduces overall drift compared to the UAV navigating alone since CoBot localizes well and carries UAV to known search locations. We conclude that the UAV’s collaboration with CoBot can accomplish search tasks that neither can do alone, while still maintaining accurate localization.

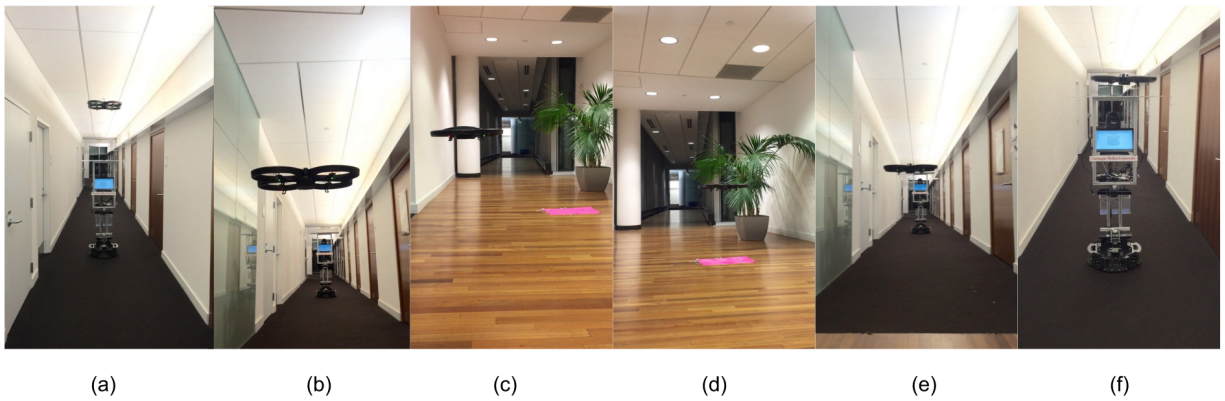


Figure 2.6: Coordination between CoBot and UAV to perform visual search task. (a) Drone taking off from the CoBot, (b)-(c) Drone performing forward search for target marker, (d) Drone hovering above marker after reaching it, (e) Drone performing backward search for CoBot, (f) Drone landing above CoBot.

2.5 Future work

Future work includes incorporating computer vision capabilities to avoid obstacles while still maintaining approximate trajectories in indoor environments. By implementing more efficient search strategies and also safely navigating around obstacles and people in the environment, the robot teams can indicate to humans where found objects are. Such work could also be extended further to use the robots to query about availability of specific objects or people in the environment.

Chapter 3

Obstacle avoidance for UAVs using deep learning

In this chapter, we propose a method to predict trajectories from images instead of the commonly used steering command (angular velocity). An overview of the proposed perception method is presented in Figure 3.1. Details of the proposed method, including the dataset creation and CNN architecture are presented in Section 3.2. In Section 3.3, we present experiments on several public datasets, dynamic simulations, and real flights and demonstrate that our method is safer than CNN trajectory prediction on raw RGB images.

3.1 Related Work

Monocular-image-based obstacle avoidance systems could be divided into two categories. The first category is to use monocular cues or motion parallax to analyze the object position and size using low-level features such as image saliency SURF features [22] and optical flow [31]. These methods are usually suitable for simple tasks such as centering in a corridor without many obstacles. The second category is to use learning algorithms to directly predict 3D obstacle position or motion commands especially using CNNs. 3D scene layout understanding [14] [37] could be an efficient way of representing 3D maps but it is usually limited to Manhattan environments or corridor with clear boundaries [38] which might not be satisfied in all environments. CNN depth estimation has also been used to represent 3D obstacles [21]. Some works already apply CNNs to autonomous vehicles. Chen *et al.* [4] propose a 'deep driving' system to predict some features to generate steering commands (angular velocity) using a feedback controller. Tai *et al* [33] apply CNNs to ground robot navigation but their input is a depth image. Giusti [12] *et al.* apply it to MAV autonomous trail following. One common problem of the above methods is that they all use human demonstration to collect ground truth datasets which have some drawbacks. Firstly, it is difficult to get a large-scale and diverse collection of training data. Secondly, their data usually is biased to certain scenarios where demonstrations happen and also biased by human understanding. Finally, they only predict the instantaneous steering command which doesn't contain information of where to go in the future so it requires the learning algorithm to give steering commands very frequently otherwise it might hit obstacles using the old angular

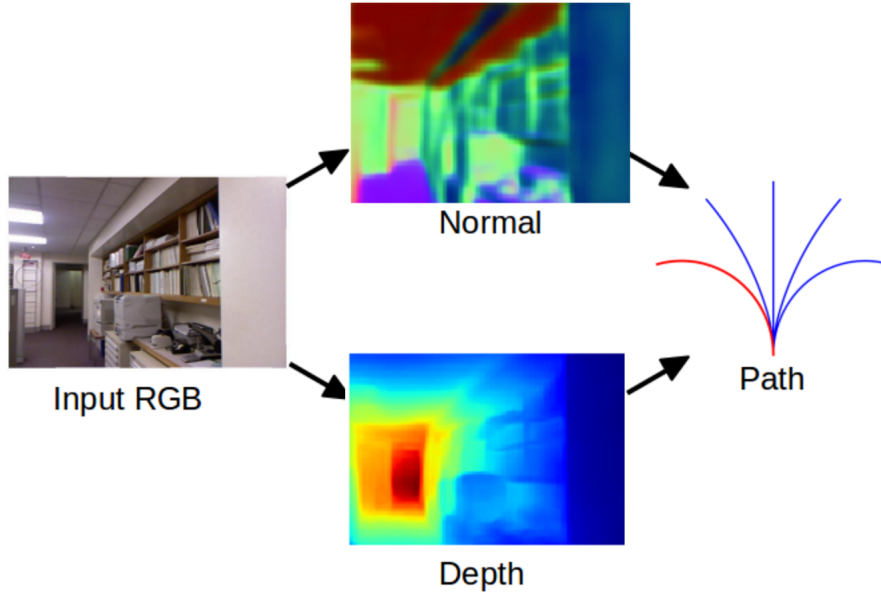


Figure 3.1: Method Overview. Instead of directly predicting path from RGB image, we propose intermediate perception: first predict depth and surface normal, which are closely related to 3D obstacles, then predict the path from the depth and normal. Both steps utilize CNNs.

velocity. It is also difficult to evaluate whether robots hit obstacles or not as it depends on how long the steering command lasts. We first propose an automatic image labelling method, then utilize CNN to predict a rough 3D model. This model might be noisy, inaccurate, and even invalid in reality so we cannot use a standard motion planner [15] [41] to generate trajectories. Instead, we utilize the predicted 3D model to predict the trajectory in another CNN model. Results show that this two-stage model performs better than directly predicting from RGB images.

3.2 Visual Perception

3.2.1 Dataset

We use the *NYUv2* RGB-D dataset, which contains many diverse indoor scenes. For trajectory labelling, in theory, robots could execute numerous continuous 3D trajectories, which will be a difficult high-dimensional regression problem. To simplify the problem, we change it into a classification problem by discretizing the continuous trajectory space into five categories: left turn, left forward, straight forward, right forward, right turn, shown in the right of Figure 3.1. Then for each image, the best one is selected as the ground truth label. These five trajectories are generated by the maximum dispersion theory of Green *et al* [13]. Each path is around 2.5m long. We avoid the ambiguity of manual labelling and design a cost function to automatically select the best path. We first project the provided depth image into 3D point cloud then build a 3D Euclidean distance map. The cost function is commonly used in optimization-based motion

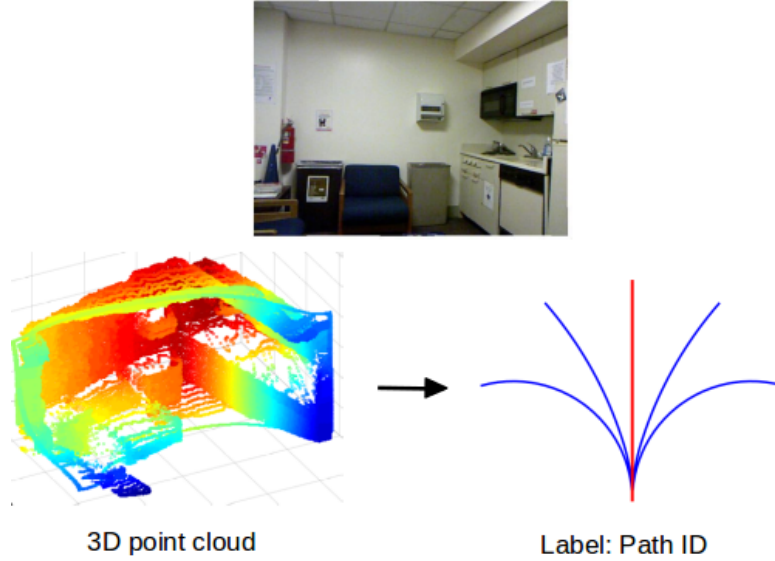


Figure 3.2: Generating ground truth path label. Using the provided depth image, a 3D point cloud map is built. Then the best path in red is selected based on the Euclidean obstacle cost and path smoothness cost.

planners [41] [11]. An example of the 3D point cloud and selected label is shown in Figure 3.2. In more detail, the cost function of a trajectory ξ is computed by:

$$J(\xi) = f_{obst}(\xi) + w f_{smooth}(\xi) \quad (3.1)$$

where w is a weighting parameter. $f_{obst}(\xi)$ is the obstacle cost function[41] penalizing closeness to obstacles:

$$f_{obst}(\xi) = \int_0^1 c_{obs}(\xi(t)) \left\| \frac{d}{dt} \xi(t) \right\| dt \quad (3.2)$$

where $c_{obs}(\xi(t)) = \|\max(0, d_{max} - d(\xi(t)))\|^2$. d_{max} is the maximum distance upon which obstacle cost is available. For *NYU* dataset, we set $d_{max} = 3.5\text{m}$. $d(\xi(t))$ is the distance to obstacles from the distance map. $f_{smooth}(\xi)$ measures the smoothness of the path and penalizes the high derivatives:

$$f_{smooth}(\xi) = \frac{1}{2} \int_0^1 \left\| \frac{d}{dt} \xi(t) \right\|^2 dt \quad (3.3)$$

The final ground truth label distribution for *NYUv2* is shown in Table 3.1. We can see that five categories are nearly equally distributed after selection and fair for the latter evaluation and comparison of different methods.

3.2.2 Intermediate Perception - depth and surface normal

Unlike existing CNN-based navigation methods [12] [33], which predict the command directly from RGB images, we first predict depth and surface normal. There are many 3D geometry

Table 3.1: Trajectory label distribution on *NYUv2* dataset.

Class ID	Distribution
Left turn	20.70%
Left forward	17.08%
Straight forward	22.15%
Right forward	18.01%
Right turn	22.08%

understanding methods that could help obstacle avoidance and we choose depth estimation due to its generality for various environments.

There has been a lot of work in depth and normal estimation from a single image [7] [36]. We utilize the CNN model from Eigen [7], which is a multi-scale fully convolutional network. It first predicts a coarse global output based on the entire image area, then refines it using finer-scale local networks. The cost functions for depth training are defined as follows: suppose d is the log difference between predicted and ground depth, then depth loss is:

$$L_{depth} = \frac{1}{n} \sum_i d_i^2 - \frac{1}{2n^2} \left(\sum_i d_i \right)^2 + \frac{1}{n} \sum_i \nabla d_i^2$$

where ∇d_i is the gradient magnitude of d . If the ground truth and predicted normal vector are v and v^* , then the dot product between vectors is used to define the normal loss function:

$$L_{normal} = -\frac{1}{n} \sum_i v_i \cdot v_i^*$$

3.2.3 Trajectory prediction

We design another CNN network shown in Figure 3.3, to utilize the predicted depth and normal to get the final path classification. It is a modification of standard Alexnet [18] with two inputs. For symmetry, we replicate the depth image (one channel) into three channels. Then depth and normal images learn convolution filters separately and are fused at the fourth layer. By doing this, the final prediction could merge two sources of information. For training, we minimize the standard classification cross-entropy loss:

$$L(C, C^*) = -\frac{1}{n} \sum_i C_i^* \log(C_i) \tag{3.4}$$

where $C_i = e^{z_i} / \sum_c e^{z_{i,c}}$ is the softmax class probability given the CNN convolution output z .

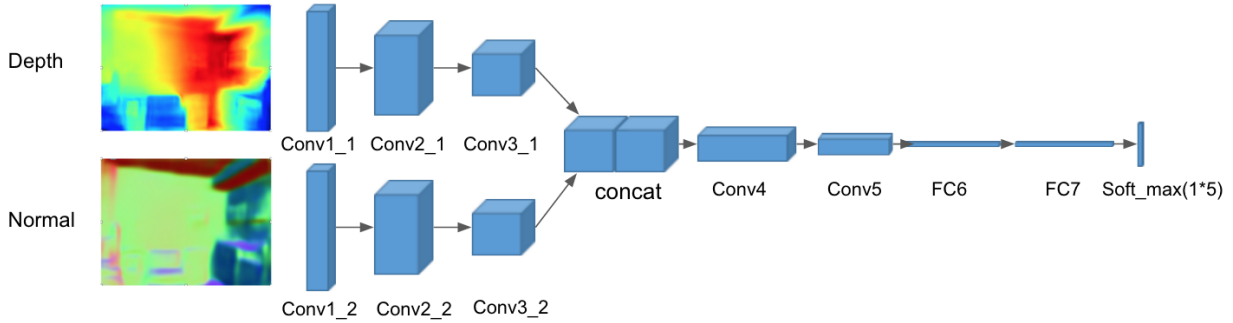


Figure 3.3: Proposed model architecture to predict path from depth and surface normal. It has two branches at the beginning to receive two input information. The prediction result is a label within five classes. For predicting depth and normal images, we use the model of [7].

3.3 Experiments

3.3.1 Training and testing

In the training phase, we use the dataset created in Section 3.2.1 to train our two-stage CNN separately. In the first stage, the ground truth surface normal is computed using the method of [29], which estimates surface normals by fitting local least-squares planes in the point cloud. In the second stage, we use the *ground truth* depth and surface normal as inputs to train CNN for path classification. We also augment the training data by flipping images horizontally. Note that for surface normal flipping, we need to reverse the horizontal normal component.

In the testing phase, we only use the raw RGB image as inputs of our two-stage CNN.

3.3.2 NYUv2 dataset Evaluation

The baseline for comparison is to directly train Alexnet CNN model to predict path label from raw RGB images, which has been adopted in many other CNN navigation works [4] [12][33]. We use the same settings of parameters for CNN weight initialization and training for comparison.

Qualitative Results

A qualitative comparison between our method and the baseline method is shown in Figure 3.4. We can see that our method generates safer trajectories most of the time.

Quantitative Results

The result of the two methods is presented in Table 3.2. The *Accuracy* column represents the standard classification accuracy. To make the evaluation more meaningful, during the dataset creation in Section 3.2.1, we also record the second best path and check whether the predicted path lies in the top two best paths shown in *Top-2 accuracy* column. From the table, our method

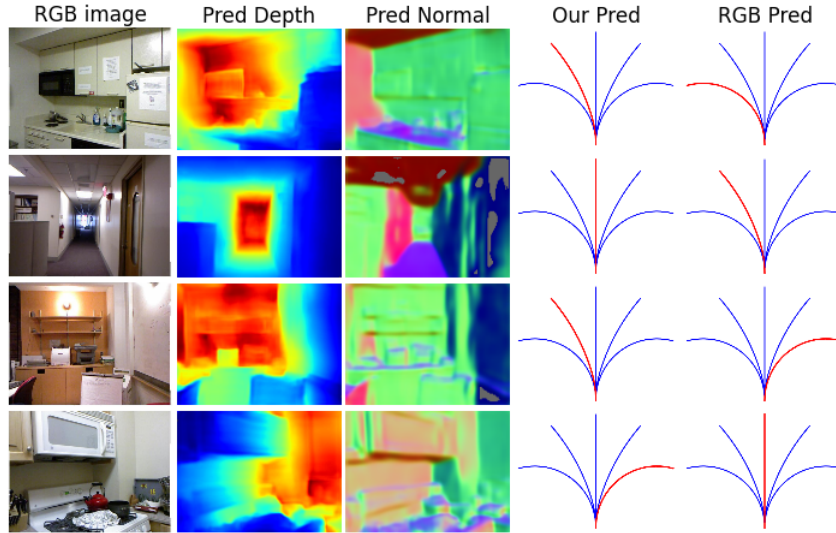


Figure 3.4: Example of path prediction on *NYUv2* dataset. The input is only RGB image. Our method and method using only RGB images for prediction are shown in the last two columns in red color. From left to right: RGB image, predicted depth image, predicted surface normal, and predicted paths. In the top image, two predictions are similar. In the bottom images, our method performs better.

with intermediate perception performs much better compared to direct perception from RGB image, with an accuracy increase of 20%.

We also report the percentage of predicted paths hitting the 3D obstacles shown in the *Safe prediction* column, which is an important metric related to robot’s safety. 92.08% of the predicted paths using the proposed method are safe. If we only consider obstacles within a certain distance (eg. $2m$) of the robots, the safe prediction ratio increases to 95.55%, whereas using random prediction, the safe prediction ratio is 62.38%. Note that since images are usually taken at some distance from obstacles, random prediction will not necessarily generate unsafe trajectories all the time.

Results of our staged CNN model with only depth prediction can be seen in Table 3.2. It

Table 3.2: Comparison of path prediction on *NYUv2* dataset.

Method	Accuracy	Top-2 accuracy	Safe prediction
Two-stage (Ours)	64.07%	82.11%	92.08%
Two-stage only depth (Ours)	60.34%	78.09%	90.68%
CNN on RGB	39.20%	60.19%	78.73%
Random	20.00%	40.00%	62.38%

Table 3.3: Confusion matrix for paths prediction on *NYUv2*.

Class	Predicted values				
	Left	Left+	Straight	Right+	Right
Left	0.65	0.11	0.11	0.03	0.10
Left+	0.14	0.64	0.12	0.04	0.06
Straight	0.07	0.14	0.60	0.10	0.08
Right+	0.07	0.05	0.21	0.56	0.11
Right	0.09	0.02	0.06	0.10	0.73

performs slightly worse compared to combining depth and surface normal. The confusion matrix of our method is shown in Table 3.3. We can see that most of the time, the prediction is correct and only a few times, the prediction will generate totally opposite direction. The reason why our network performs better is that we add ground truth depth information in the first stage’s CNN model training which will guide the network to learn 3D scene information related to obstacle avoidance. However, the baseline method might not be able to learn this 3D information just from a single image without ground truth depth instruction.

3.3.3 Other Public Indoor dataset Evaluation

Recently, Tai *et al*[33] proposed the *Ram-lab* RGB-D corridor dataset for CNN autonomous navigation. We didn’t directly compare with their method as they predict motion command from true depth image instead of RGB image. Besides, their dataset labeling comes from human demonstration, which is different from ours. The provided image labels are not well distributed. So we generate a new path label for their dataset using the same method and parameters in Section 3.2.1. Depth image is pre-processed by the cross-bilateral filter of Paris [23] to fill the missing regions in order to project to 3D space and compute surface normals. Note that we directly applied the trained model from *NYUv2* on this dataset without turning any parameters. Some prediction examples are shown in Figure 3.5. A metric comparison on all the images is presented in Table 3.4.

We can find that our method still outperforms the baseline methods in terms of classification *accuracy* and *Safe prediction* ratio. However, the improvement is not as large as the *NYUv2* dataset mainly because *Ram-lab* dataset contains only corridor images with similar scene structure and appearance.

3.3.4 Quadrotor simulation flight

We also evaluate our network for continuous robot flight in simulations shown in Figure 3.6. Our CNN model predicts a trajectory in real-time, then the robot will follow the trajectory until a new predicted trajectory comes in. Yaw heading of quadrotor is set as the tangent direction of the trajectory. We use the ardrone quadrotor dynamic simulator developed by Engel *et al.* [9] in the default gazebo willowgarage world model without texturing it. A simple PID controller is used

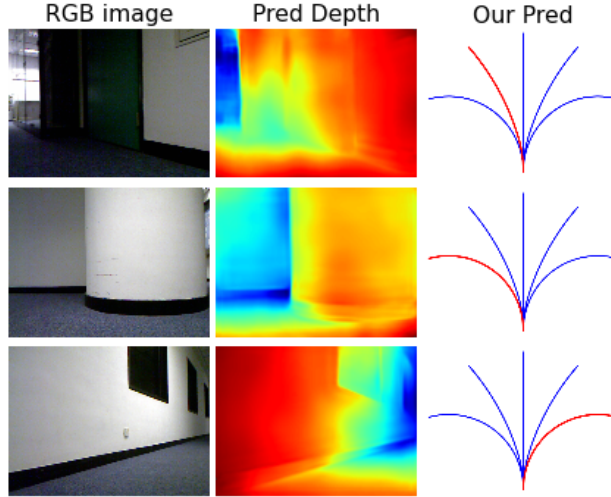


Figure 3.5: Some prediction examples in *Ram-lab* dataset images.

Table 3.4: Comparison of path prediction on *Ram-lab* datasets.

Method	Accuracy	Top-2 accuracy	Safe prediction
Two-stage (Ours)	50.05%	73.21%	91.10%
CNN on RGB	42.68%	65.67%	87.92%
Random	20.00%	40.00%	72.37%

for trajectory tracking. The final performance is reflected both by the CNN prediction error and also by the uncertainty in dynamic modelling and trajectory tracking. In the simulation, there are various kinds of door and wall configurations to test the robustness of our system. Some of the prediction examples are shown in Figure 3.7 and the top-view trajectory prediction is shown in Figure 3.8. We can see that even though the scene is quite different from *NYUv2* dataset, our CNN model still works well and the quadrotor is able to travel long distances and go through very narrow doors.

The ground truth pose is provided by gazebo. Our mean obstacle distance is 0.98m. The narrowest door is only 0.78m wide while quadrotor is 0.52m wide. The average prediction time of our whole system is 38.5ms on a GeForce GTX 980 Ti GPU, so it is able to run in real time over 25Hz.

We need to note that our CNN model can only be used for local obstacle avoidance, so the quadrotor can easily come to a dead end if there is no high level instruction shown in the top scene of Figure 3.6. Moreover, we currently only use five paths for classification, which might not satisfy the real world requirements. For example, if there is sharp turning, U-turn trajectory or stop motion to select, the robot flight could perform better. These techniques have been used in some actual flight system [11].

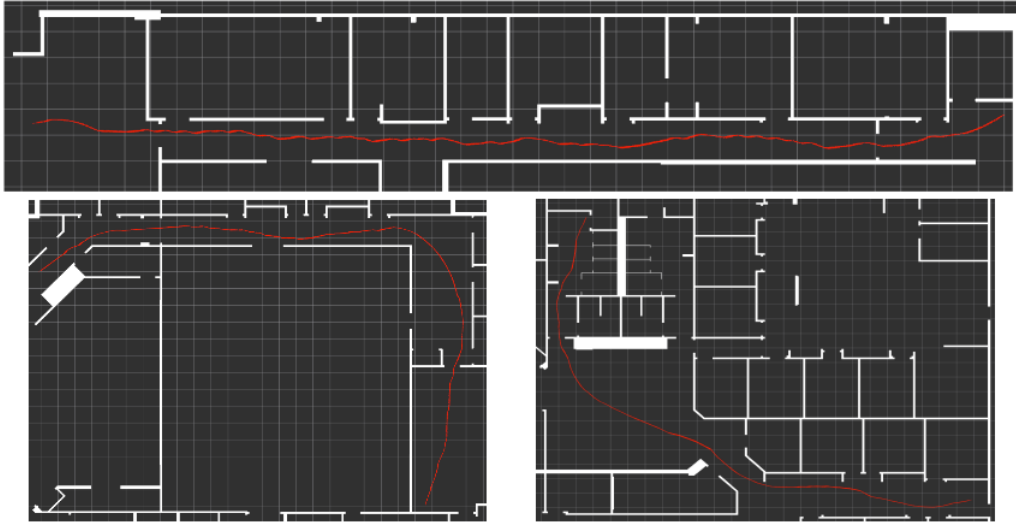


Figure 3.6: Some scenarios of the dynamic simulation. White area represents obstacles and red curve is the quadrotor path.

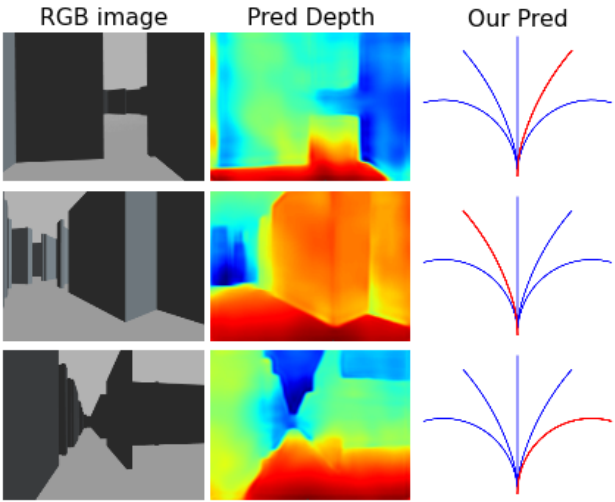


Figure 3.7: Some prediction examples in gazebo simulations.

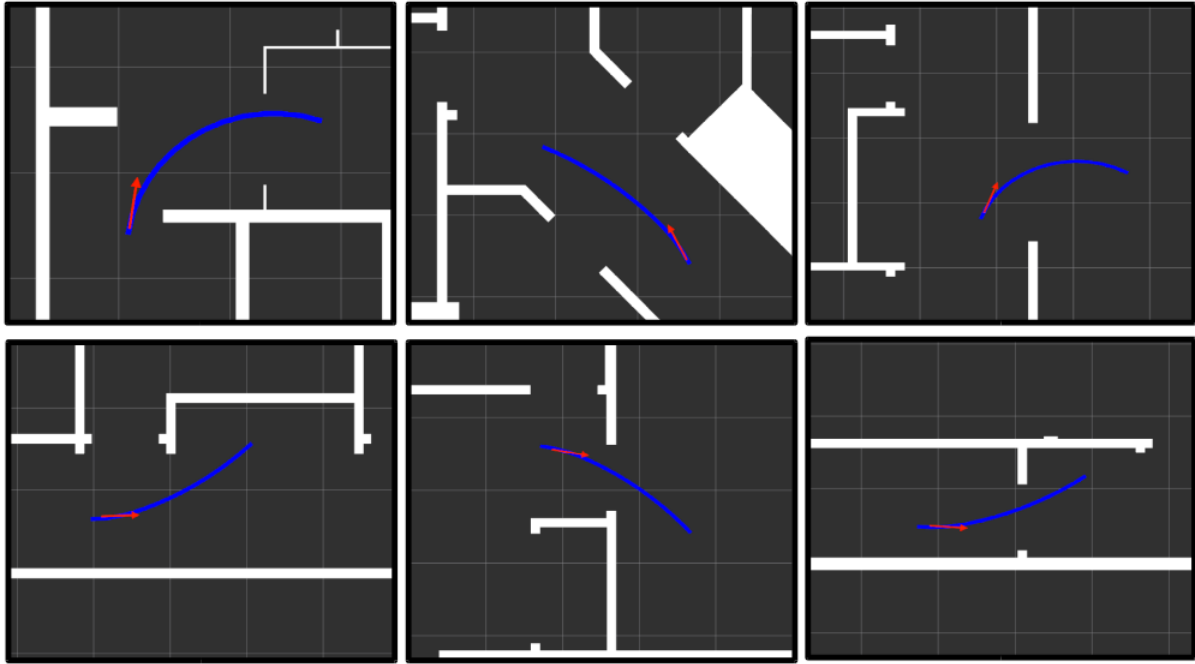


Figure 3.8: Top view of path prediction in gazebo simulations. The red arrow represents the robot pose, the blue line is the predicted trajectory. The white areas in the image represents the obstacles.

3.3.5 Real quadrotor flight

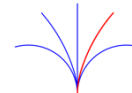
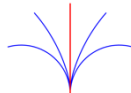
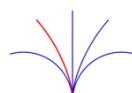
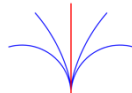
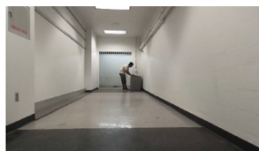
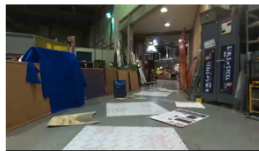
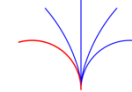
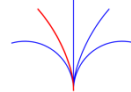
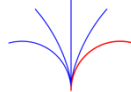
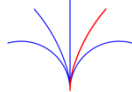
We also test our algorithm on real quadrotor flight. The platform is a parrot Bebop quadrotor, shown in Figure 3.9a. It can send the forward-facing camera image via WiFi to our host laptop, which predicts the trajectory and sends back velocity command to the drone. The laptop is equipped with a GeForce GTX 960M GPU for CNN prediction which takes about 0.143s per frame. There is also a image transmission delay of about 0.2s. We still use the trained model from *NYUv2* dataset. Due to unstable and sometimes jumping state estimation from the bebop drone, we cannot use the position feedback in the trajectory tracking, which deteriorates the whole performance. We test our vehicle in three indoor environments: straight corridor, turning, and front obstacles, as shown in Figure 3.9a. Some images and predictions taken from real flights are shown in Figure 3.9b.

3.4 Future work

Future work includes integrating the proposed obstacle avoidance system with high-level motion planning, accurate state estimation and also adding more trajectories into it for example U-turn, to better avoid obstacles. Integrating prediction from previous frames should also increase the performance.



(a)



(b)

Figure 3.9: (a) Real flight scenes including curved corridor, front obstacles and corridor following (b) Eight prediction examples from quadrotor's view on the fly. For each image, the path image below it shows the predicted path. More results could be found in the supplementary video.

Chapter 4

Automatic Patch Pattern Labeling for Explanation (APPLE)

In this chapter, we propose an algorithm to label the features of an image that the network focuses on in order to explain why the network made its prediction. Our goal is to explain the output of a CNN image classifier by automatically evaluating the neurons and their corresponding patches of the image that contribute the most to the classification. For example, when classifying images of polar bears, neurons that detect patches containing the eyes, nose or paws of the animal might contribute significantly to the classification. We would like to automatically find the neurons that are important (i.e., neurons that contribute significantly to the classification), and label the features in the image that they correspond to. In order to do this, we propose our Automatic Patch Pattern Labeling for Explanation (APPLE) algorithm to:

1. find high importance neurons within the CNN,
2. deconvolve the network to determine the patch of the image that each important neuron looks at, and
3. automatically label those patches using a secondary classifier to determine object features that the patch contains.

4.1 Related Work

Prior research pertaining to understanding a CNN’s predictions can be divided into two categories: weakly supervised localization and network structure analysis. In weakly supervised localization, the objective is to highlight the object features (pixels) within an image. Prominent works include Class Activation Mapping (CAM) [40], Grad-CAM [27] and Local Interpretable Model-agnostic Explanations (LIME) [25]. While these techniques use different measures for determining which pixels in the image are most important for classification, they do not analyze how the pixel features are propagated through the network to arrive at a prediction. We build on prior work that aims to understand how information propagates through a network (e.g., [10, 20, 32, 39]). Although most of the network structural analyses, including [39], provide insights into a CNN’s classification at the neuron-level, they require human intervention to man-

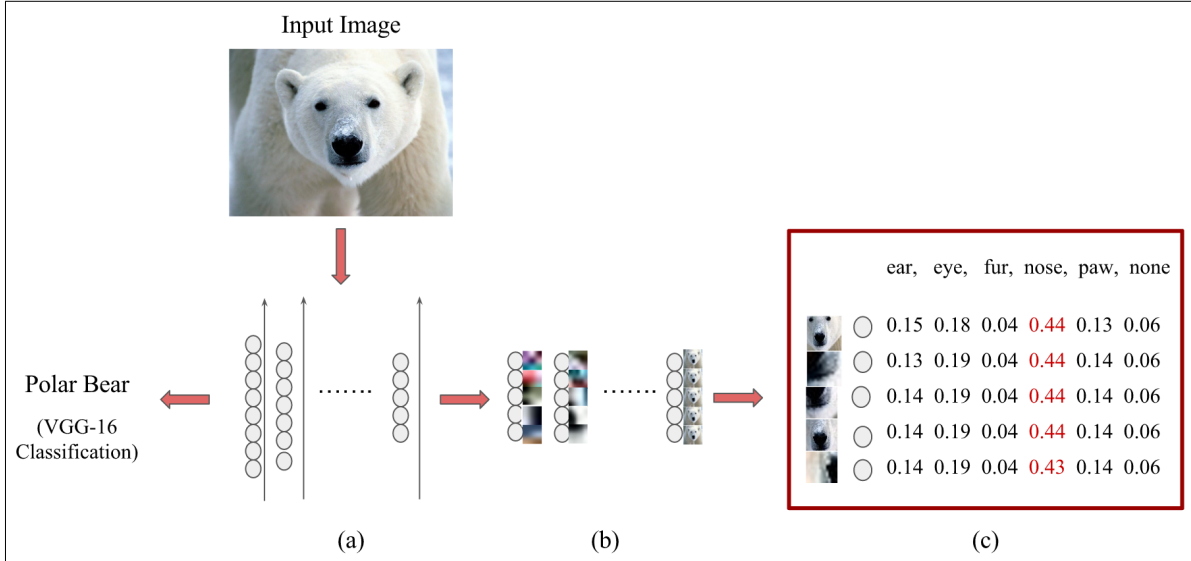


Figure 4.1: Pipeline of our APPLE algorithm : a) ranking neurons as per measures described in Section 3.1, b) identifying image patches corresponding to the top 5 neurons from (a), c) our patch classifier classifies the image patches from (b), only top-5 classifications are shown here.

ually analyze the activations or the important image patches to interpret how a network made a prediction. This manual process does not scale as the network gets bigger or as the number of images to analyze increases. Our Automatic Patch Pattern Labeling for Explanation (APPLE) algorithm is built on top of Fergus & Zeiler [39], but does not require any manual probing to understand the image patches that activate individual neurons, and we eliminate the need to look at all the neurons. In particular, we focus only on analyzing neurons that are important to the network. By assessing the image patches corresponding to important neurons, APPLE simultaneously localizes the object within the image while also automatically labeling those patches with object feature descriptions.

4.2 Approach

4.2.1 High importance Neurons

Each neuron (x, y) in layer l in a deep network takes as input a set of signal matrices $z_{i,j}^{l-1}$ from neurons at indices (i, j) in the previous layer and propagates a new signal matrix $z_{x,y}^l$ to the next layer using the following equation:

$$z_{x,y}^l = \sum_{i,j} w_{i,j}^l \phi(z_{i,j}^{l-1}) + b_{x,y}^l \quad (4.1)$$

where

- $z_{x,y}^l$ is the activation signal matrix of the neuron at index (x, y) in layer l
- $w_{i,j}^l$ is the weight of the signal from neuron (i, j) in layer $l - 1$ to (x, y) in l

- $\phi(z)$ is the activation function applied to the activation output of the neurons from the previous layer, and
- $b_{x,y}^l$ is a bias function.

The propagation of information through neuron (x, y) is related to 1) its final activation z and 2) the weights w that subsequent neurons place on the output signal of that neuron. Neurons that have high activation signal may contribute highly to the activation of subsequent neurons and the classification prediction. Similarly, neurons whose activations are multiplied with high weight in subsequent neurons may also contribute highly to the classification. In addition to high overall values of weights and activations, high variance within a matrix could also signify that the neuron is detecting patterns with intensity changes (e.g., edges of objects)

We propose four measures of importance based on the weights and activations of each neuron (x, y) in layer l :

- **Activation Matrix Sum:** The sum of all values in the output activation signal matrix z :

$$\sum_{row,col} z_{x,y}^l[row][col], \quad (4.2)$$

- **Activation Matrix Variance:** The variance of all values in the output activation signal matrix z :

$$Var_{row,col} z_{x,y}^l[row][col], \quad (4.3)$$

- **Weight Matrix Sum:** The sum of all values in the weight matrix w^{l+1} :

$$\sum_{row,col} w_{x,y}^{l+1}[row][col], \quad (4.4)$$

- **Weight Matrix Variance:** The variance of all values in the weight matrix w^{l+1} :

$$Var_{row,col} w_{x,y}^{l+1}[row][col] \quad (4.5)$$

The neurons in each layer are ranked from highest to lowest values for each measure (Figure 4.1a). The highest ranked neurons in each layer are those that are most important.

Note that since the CNN weights remain same for all images, it is possible for some neurons to be highly ranked with respect to our weight measures yet have no activation for a particular image. When ranking neurons for weight-based importance measures in a test image, we only consider those neurons that have activation $z_{x,y}^l > 0$.

4.2.2 High Importance Patches

Given the ranked neurons, we select the top N neurons per layer (i.e., we use the top 5 neurons). For each of the top N neurons, we are interested in identifying the image patches that they convolve (Figure 4.1b). We determine the image patches by deconvolving the network as in Zeiler and Fergus [39]. Although different layers use different sized patches, these patches are all identified as of high importance.

4.2.3 Patch classifier

In order to label the object features in each high importance patch, we construct and train a secondary classifier. Given a set of object features as classifier labels (e.g., eyes, nose, ear, fur, and paws for polar bears), we manually crop image patches as training data for each of these labels. We also include a None label which represents our background scene and parts of the object that may be difficult to distinguish.

Once the patch classifier is trained, it can be run on any patches that are identified by the CNN image classifier as important in order to determine the likely labels for those important image patches. We rank the likelihood of each label on each patch (Figure 4.1c) to determine the most likely label. Compared to [39] which requires manual probing to understand patches that activate neurons, our patch classifier automatically determines the *labels* that can be used to explain what important areas of the image the network focused on.

4.2.4 Putting it all together

Given an image and a CNN model, our APPLE algorithm forward propagates the image to determine its classification. Then, it automatically analyzes the CNN to find the top N most important neurons and constructs a list of image patch regions to label. APPLE runs the patch classifier on the image patches to determine the most likely labels. Currently, we construct one patch classifier per CNN classification (e.g., a patch classifier of polar bear features for images classified as polar bears) though patches could be labeled using more generic classifiers if possible. Labeled patches can then be further sorted based on the maximum confidence. The important patches are a visual representation of the explanation of how the network determined the image’s classification. The labels on the patches represent a semantic representation of the same explanation without requiring humans to interpret the image.

Next, we demonstrate the use of our APPLE algorithm on an object recognition task.

4.3 Experiments

In order to demonstrate the ability of our APPLE algorithm to find important neurons and corresponding image patches and then automatically label them, we used the pre-trained VGG-16 classifier trained on animals to find important features of polar bears in test images - eyes, ears, fur, nose, and paw. Polar bear images are particularly challenging as the background of the images is often snowy and contains similar colors as the polar bears themselves. The ability of an algorithm to explain why the CNN classified the object as a polar bear may help humans understand how to improve training or recover from errors. We describe our experimental setup and our results that test APPLE’s ability to find important patches and label them. Additionally, we compare APPLE’s important patches to CAM for object localization within the image.

4.3.1 CNN Classifier

The neural network architecture chosen for this investigation was the VGG-16 architecture [30] with pre-trained weights as provided [5]. The VGG-16 architecture consists of 13 convolution layers followed by 3 dense layers, with max pooling after the 2nd, 4th, 7th, 10th and 13th layer. The pre-trained weights used were not further tuned to the input images that we analyzed.

We chose VGG-16 because it is a deep network with enough intermediate layers to find important patches. However, only image patches for neurons between layers 3 and 9 were evaluated in the analysis below. The image patches in the first few layers (1 and 2) were too small to train a patch classifier for. Furthermore, results from Ranzato et al.[24] already show that the first layers learn stroke-detectors and Gabor-like filters. Similarly, the image patches corresponding to the last few layers (10 and above) focus on a majority of the image and thus they contain too many of the object features in each patch to provide any useful insight.

4.3.2 Patch Classifier

In order to train our patch classifier, we listed distinguishing features of polar bears - ears, eyes, nose, fur, and paws - and manually cropped those important polar bear features from images in the dataset. These polar bear images were chosen from the Imagenet dataset from the ILSVRC12 challenge. For the None class we collected background patches. Each class had around 80 images, totalling the training data size to be 480. Figure 4.2 contains sample patches from our classifier dataset. Other methods could be used in the future to automatically generate the list of features for each object (e.g., using web search) as well as automatically crop features from images (e.g., using crowd-sourcing).

We used a multi-class Support Vector Classifier (AdaBoostSVM [19]) with $c=0.771$, and $\gamma = 0.096$, with an rbf kernel, determined by running K-fold cross validation on the patch data. The classifier obtained an average test set accuracy of 80% when the data was split into 80% training-20% test. SVMs were chosen for the task because of their ability to handle non-linear data: an RBF kernel was used as described in [19]. As the patch classifier accuracy is not the focus of the contribution of this work, we present our findings using this classifier but note that it is possible to train a more accurate classifier for better results.

4.3.3 Demonstration of APPLE

We ran our experiments and demonstrations of APPLE on a laptop with 15.5 GiB RAM and 4GB GeForce GTX 960M with Intel Core i7-6700HQ CPU @ 2.60GHz x 8. Given an input image of a polar bear image (Figure 4.3a), the pre-trained VGG-16 network, and the AdaBoostSVM patch classifier, APPLE forward propagates the image to ensure that it is correctly classified as a polar bear. If it is classified as a polar bear, APPLE collects the neurons' activations obtained from the forward propagation, as well as the pre-trained VGG-16 neuron weights in order to find the five high importance neurons per layer. Neurons that have high weight but are not active on the input image are discarded from the top 5 lists for Weight Matrix Sum and Variance.

Deconvolutions are then performed on the important neurons to determine their corresponding image patches. Figure 4.3b shows the 35 image patches selected by the Activation Matrix





































Ear	Eye	Fur	Nose	Paw	None
					
					
					
					
					
					

Figure 4.2: Sample training data for the Patch classifier.

Sum measure (top 5 neurons from the seven layers - layers 3-9 - of interest). APPLE then labels the image patches using the patch classifier to determine the polar bear features (eyes, nose, ears, paw, fur) that they capture. For example, most of the image patches in the first two rows of the figure contain the polar bear’s nose. Our algorithm finally sorts the labels by confidence and can display them to a human to explain the classification (Figure 4.4). The most confidently labeled image patches selected for Figure 4.4(left) are all labeled as the nose, which makes sense given the distinctive color of the nose against the background of the white bear and snow. Similarly, the most important patches for the image on the right also contain the bear’s nose.

4.3.4 Evaluation of Important Patches

Next, we evaluated APPLE’s ability to select important neurons and corresponding image patches. Our experiments were conducted on 10 images containing polar bears with varying backgrounds like snow, rocks and water. We selected new images by searching for polar bears on the Internet because the small number of polar bear images in Imagenet made it likely that we would select training images. Additionally, because our goal is to demonstrate the ability of APPLE to find important patches in any image, we chose to manually search for images of polar bears in a variety of environments, poses, and conditions. The selection process involved criteria such as pose, number of classes visible in the image, lighting, and image resolution. By varying the different settings we could determine how different conditions influence the results of both our high importance neuron ranking method, as well as our patch classifier.

Patch Localization

We first evaluated APPLE’s ability to find image patches that were localized on the object of interest that the CNN is classifying (i.e., polar bears in our experiments). *Patch localization* is measured as the ratio of number of patches containing pixels of the object (polar bear) to the total number of patches.



(a)



(b)

Figure 4.3: (a): Input image. (b): High Importance Patches selected by APPLE algorithm using *Activation Matrix Sum* measure.

In order to compute this ratio, we manually evaluated each image patch by comparing the patch against the original input image. In total, for each bear image and each evaluation measure, 35 important patches (top 5 patches across 7 layers) were evaluated as to whether they contained portions of the bear (using our input image as reference). Figure 4.3b shows 35 high importance patches picked by our algorithm under the *Activation Matrix Sum* measure. Each of these patches contains pixels belonging to polar bear, and hence they are all deemed as correct patches.

Table 4.1 shows the precision of each of our importance evaluation measures on the test images. All the metrics perform well, with *Weight Matrix Sum* performing slightly better than the rest. Since all the metrics chosen impact the information flow across the layers, it is not surprising to note that they show precision greater than 0.9. In 8 out of 40 experiments (4 metrics across 10 images) that we ran, the precision value was 1, suggesting that if the background of the image is distinct from the patches, APPLE algorithm could identify only the patches corresponding to the class under consideration. We specifically chose a challenging image classification task in which polar bears are not often distinct from their background to stress APPLE's ability to find the polar bear patches. Our results demonstrate that any suggested measure could be used for the purpose of extracting neurons and thereby important patches, for the purpose of understanding the reasoning behind classification.

Patch Label Precision

While the focus of our evaluation is not the patch classifier, we did evaluate its precision at labeling the image patches. *Patch label precision* is measured as the ratio of correctly classified



is classified as **polar bear** because of



is classified as **polar bear** because of

Layer	Neuron	Patches	Predictions: ear, eye, fur, nose, paw, none
Layer : 7	Neuron : 4		0.15 0.18 0.04 0.44 0.13 0.06
Layer : 3	Neuron : 2		0.13 0.19 0.04 0.44 0.14 0.06
Layer : 4	Neuron : 2		0.14 0.19 0.04 0.44 0.14 0.06
Layer : 6	Neuron : 3		0.14 0.19 0.04 0.44 0.14 0.06
Layer : 3	Neuron : 0		0.14 0.19 0.04 0.43 0.14 0.06

Layer	Neuron	Patches	Predictions: ear, eye, fur, nose, paw, none
Layer : 3	Neuron : 1		0.11 0.21 0.05 0.42 0.15 0.06
Layer : 3	Neuron : 4		0.11 0.20 0.05 0.41 0.17 0.07
Layer : 4	Neuron : 1		0.11 0.21 0.05 0.41 0.15 0.06
Layer : 4	Neuron : 2		0.11 0.22 0.05 0.41 0.15 0.07
Layer : 4	Neuron : 0		0.11 0.20 0.05 0.39 0.16 0.08

Figure 4.4: APPLE sorts the labeled patches by confidence to present to a human in order to explain the CNN’s image classification. Two example images are shown with their important patches selected using *Activation Matrix Sum* measure.

Table 4.1: Evaluation of important patches

Evaluation Measure	Patch Localization	Label Precision
Weight Sum	0.94	0.571
Weight Variance	0.91	0.551
Activation Sum	0.92	0.626
Activation Variance	0.91	0.608

patches to the total number of patches. A patch classification is considered correct if the top label output by the patch classifier matches our manually labeled ground-truth. Table 4.1 shows the precision of the AdaBoost SVM on the important image patches. The *Activation Matrix Sum* performs better than the other measures of accuracy. In general, the metrics involving the activations perform better than metrics involving the weights. However, precision is less than 0.65 in all the cases, due to the choice of patch classifier and the quality of training data provided. For example, we only obtained 80 patches per label and the low-resolution of the images led to high confusion between eyes and nose as well as fur and background (None). The precision for patches belonging to the background None is less than 0.1 in all the cases. Despite the challenges in building a patch classifier, our results demonstrate the patches identified as important by APPLE can be labeled accurately to help a human understand the CNN information propagation.

Comparison to Weakly Supervised Localization

Finally, we evaluated our important image patches compared to Class Activation Maps (CAM) for weakly supervised localization. We outlined APPLE’s important patches using the *Weight*

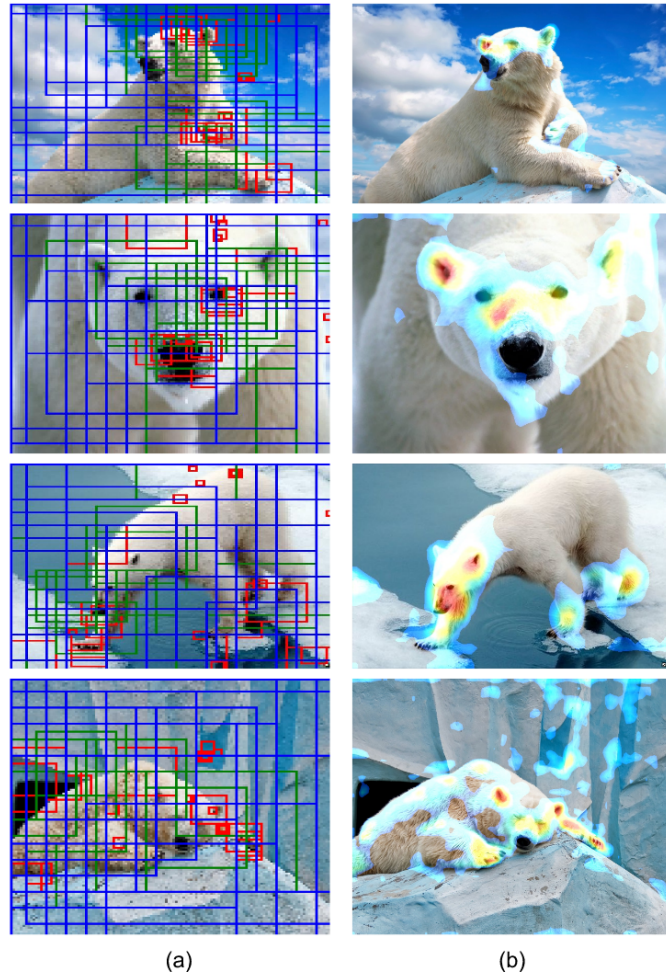


Figure 4.5: Comparing weakly supervised localization between a) APPLE and b) CAM. In APPLE, red boxes indicate layers 3 and 4, green indicates layers 5 - 7 and blue indicates layers 8 and 9. On the CAM images, the heatmap visualizes its important pixels.

Matrix Sum measure on top of the original image to compare with CAM's heatmap output. Figure 4.5a illustrates the localization abilities of our APPLE algorithm (red boxes indicate patches belonging to layers 3 and 4, green indicates layers 5 - 7 and blue indicates layers 8 and 9). Figure 4.5b shows CAM's output on the same images.

Because our image patches contain background in addition to polar bear pixels, measures for comparing the overlapping regions bias our results lower than reality. By inspection, our results for important patches from layers 3-7 roughly cover the same areas as CAM. Although we ignore patches from layers higher than 9, the patches for 8 and 9 (shown in blue) still cover large parts of the image. Layers 5-7 seem to capture important parts of the image containing the polar bear without as much environment background in the patch.



Figure 4.6: APPLE sorts the labeled patches by confidence to present to a human in order to explain the CNNs image classification and corresponding action (important patches are selected using Weight Matrix Sum measure)

4.3.5 How does APPLE apply to UAVs?

As demonstrated in the above sections, APPLE works well for polar bear images. Consider a modified version of deep-learning-based obstacle avoidance system described in Chapter 3, where the drone navigates forward if there is no person detected in the scene and stops otherwise. It is important for the user to understand if the drone stopped due to the presence of a person or if the system picked cues from other parts of the scene. APPLE could be extremely helpful in such scenarios. We trained a person vs non-person classifier using VGG-16 network on the INRIA dataset [6]. While training, all the images were resized to 68x128, and 20% of the images were used for validation. In order to train our patch classifier, we listed five distinguishing features of human beings - head, torso, hand, leg, and foot - and manually cropped those features from images in the INRIA dataset [6]. For the *none* class, we collected background patches. Each class had around 120 images, totalling the training data size to be 720. Figure 4.6 demonstrates APPLE applied to person vs non-person classifier. Figure 4.6 shows the input image classified as ‘Person’ with corresponding action as ‘Stop’. Output of APPLE that contains top-5 patches selected using Weight Matrix Sum measure and their corresponding classifications performed by patch classifier can also be seen. Figure 4.7 shows weakly supervised localization of APPLE, which demonstrates that the image is classified as ‘Stop’ or ‘Person’ due to right reasons i.e. bounding boxes are centered around persons in the image.



Figure 4.7: Weakly supervised localization of APPLE (red boxes indicate layers 3 and 4, green indicates layers 5 - 7 and blue indicates layers 8 and 9.)

4.4 Future work

Future work could include improving patch classifier performance and testing our approach on other CNN architectures. Extending our work to processing camera feed in real-time would have numerous practical applications. Additional metrics like drop-in-classification accuracy for perturbed images, would help in better evaluation of the proposed measures of neuron importance.

Chapter 5

Conclusion

In this thesis, we contributed two vision-based algorithms to enable UAVs to perform search task and to avoid obstacles in real-time, and a technique for explaining the deep-learning algorithms to humans in their environments.

In chapter 2, we proposed and presented a coordination between CoBot and a drone leveraging the robust localization and navigation capabilities of CoBot with ARDrones capability to maneuver easily through indoor environments and search for an object of interest. As to enable reliable navigation of drone to perform search task, we proposed a vision-based moving target navigation algorithm. Observed results affirm that the proposed algorithm could be used effectively for the service search tasks.

In chapter 3, we proposed a CNN based navigation system that was shown to be navigating safely in various challenging environments. We proposed a method to get ground truth labels from RGB-D images automatically without human demonstrations. We proposed a two-stage CNN with intermediate perception. The first stage predicts the depth and surface normal from images, which are two important geometric properties related to 3D obstacle avoidance. The second step predicts a path from the depth and normal maps using another CNN model. Results on the NYUv2 dataset and other public datasets showed that our system performs better than directly predicting path from RGB image. We also applied our CNN model to simulation and actual flight without retraining which is different from existing methods where training and testing data usually comes from the same environments.

In chapter 4, we contributed our algorithm Automatic Patch Pattern Labeling for Explanation (APPLE) to analyze the neuron-level information propagation and help humans understand the regions of interest. We contributed four different measures of neuron importance, such as sum and variance across the activation matrix and weight matrix of neurons. We demonstrated that in an object recognition task, our algorithm is able to use the measures to identify neurons within the CNN that focus on important features of the recognized object (i.e., body parts of animals). In particular, we demonstrated that all of APPLEs importance measures find regions of the images that contain the object of interest. We then used a patch classifier to label the features of the object. Beyond simply highlighting important pixels for visualization purposes, automatic image patch labels provided by our patch classifier can be used to explain the object recognition without requiring a human to decipher the image or manually probe the reasoning behind a prediction.

Bibliography

- [1] Abraham Bachrach, Ruijie He, and Nicholas Roy. Autonomous flight in unknown indoor environments. *International Journal of Micro Air Vehicles*, 1(4):217–228, 2009. 1.1
- [2] Cooper Bills, Joyce Chen, and Ashutosh Saxena. Autonomous mav flight in indoor environments using single image perspective cues. In *Proceedings of ICRA 2011*, pages 5776–5783. IEEE, 2011. 2.1
- [3] J. Biswas and M. Veloso. Wifi localization and navigation for autonomous indoor mobile robots. In *Proceedings of ICRA 2010*, pages 4379–4384, May 2010. doi: 10.1109/ROBOT.2010.5509842. 1
- [4] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015. 3.1, 3.3.2
- [5] François Chollet. Keras. <https://github.com/fchollet/keras>, 2015. 4.3.1
- [6] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005. 4.3.5
- [7] Fergus R Eigen D. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015. (document), 3.2.2, 3.3
- [8] Jakob Engel, Jürgen Sturm, and Daniel Cremers. Camera-based navigation of a low-cost quadcopter. In *Proceedings of IROS 2012*, pages 2815–2821. IEEE, 2012. 2.1
- [9] Jakob Engel, Jürgen Sturm, and Daniel Cremers. Scale-aware navigation of a low-cost quadcopter with a monocular camera. *Robotics and Autonomous Systems*, 62(11):1646–1656, 2014. 3.3.4
- [10] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. 2009. 4.1
- [11] Zheng Fang, Shichao Yang, Sezal Jain, Geetesh Dubey, Silvio Maeta, Stephan Roth, Sebastian Scherer, Yu Zhang, and Stephen Nuske. Robust autonomous flight in constrained and visually degraded environments. In *Field and Service Robotics*, pages 411–425. Springer, 2016. 1.1, 3.2.1, 3.3.4
- [12] Alessandro Giusti, Jerome Guzzi, Dan Ciresan, Fang-Lin He, Juan Pablo Rodriguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jurgen Schmidhuber, Gianni Di Caro, et al.

- A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letter*, 2016. 3.1, 3.2.2, 3.3.2
- [13] Colin Green and Alonzo Kelly. Toward optimal sampling in the space of paths. In *13th International Symposium of Robotics Research*, 2007. 3.2.1
- [14] Varsha Hedau, Derek Hoiem, and David Forsyth. Recovering the spatial layout of cluttered rooms. In *Computer vision, 2009 IEEE 12th international conference on*, pages 1849–1856. IEEE, 2009. 3.1
- [15] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011. 3.1
- [16] Matt Knudson and Kagan Tumer. Adaptive navigation for autonomous robots. *Robotics and Autonomous Systems*, 59(6):410–420, 2011. 1.1
- [17] T. Kollar, V. Perera, D. Nardi, and M. Veloso. Learning environmental knowledge from task-based human-robot dialog. In *ICRA 2013*, pages 4304–4309, May 2013. doi: 10.1109/ICRA.2013.6631186. 2.2.1
- [18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems (NIPS)*, pages 1097–1105, 2012. 3.2.3
- [19] Xuchun Li, Lei Wang, and Eric Sung. Adaboost with svm-based component classifiers. *Engineering Applications of Artificial Intelligence*, 21(5):785–795, 2008. 4.3.2
- [20] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *2015 IEEE conference on computer vision and pattern recognition (CVPR)*, pages 5188–5196. IEEE, 2015. 4.1
- [21] Michele Mancini, Gabriele Costante, Paolo Valigi, and Thomas A Ciarfuglia. Fast robust monocular depth estimation for obstacle detection with fully convolutional networks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2016. 3.1
- [22] Tomoyuki Mori and Sebastian Scherer. First results in detecting and avoiding frontal obstacles from a monocular camera for micro unmanned aerial vehicles. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1750–1757. IEEE, 2013. 3.1
- [23] Sylvain Paris and Frédo Durand. A fast approximation of the bilateral filter using a signal processing approach. In *European conference on computer vision*, pages 568–580. Springer, 2006. 3.3.3
- [24] Marc’Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In *Proceedings of the 19th International Conference on Neural Information Processing Systems, NIPS’06*, pages 1137–1144, Cambridge, MA, USA, 2006. MIT Press. URL <http://dl.acm.org/citation.cfm?id=2976456.2976599>. 4.3.1
- [25] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. ”why should I trust you?”: Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016. URL <http://arxiv.org/abs/1602.04938>. 4.1

- [26] Susan Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debabrata Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive uav control in cluttered natural environments. In *Proceedings of ICRA 2013*, pages 1765–1772. IEEE, 2013. 2.1
- [27] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016. URL <http://arxiv.org/abs/1610.02391>. 4.1
- [28] Shaojie Shen, Yash Mulgaonkar, Nathan Michael, and Vijay Kumar. Vision-based state estimation for autonomous rotorcraft mavs in complex environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1758–1764. IEEE, 2013. 1.1
- [29] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision*, pages 746–760. Springer, 2012. 3.3.1
- [30] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 4.3.1
- [31] Kahlouche Souhila and Achour Karim. Optical flow based robot obstacle avoidance. *International Journal of Advanced Robotic Systems*, 4(1):13–16, 2007. 3.1
- [32] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014. 4.1
- [33] Lei Tai, Shaohua Li, and Ming Liu. A deep-network solution towards model-less obstacle avoidance. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016. 3.1, 3.2.2, 3.3.2, 3.3.3
- [34] M. Veloso, J. Biswas, B. Coltin, S. Rosenthal, T. Kollar, C. Mericli, M. Samadi, S. Brando, and R. Ventura. Cobots: Collaborative robots servicing multi-floor buildings. In *Proceedings of IROS 2012*, pages 5446–5447, Oct 2012. doi: 10.1109/IROS.2012.6386300. 1, 2.2
- [35] Rodrigo Ventura, Brian Coltin, and Manuela Veloso. Web-based remote assistance to overcome robot perceptual limitations. In *AAAI-13, Workshop on Intelligent Robot Systems.*, 2013. 2.2.1
- [36] Xiaolong Wang, David Fouhey, and Abhinav Gupta. Designing deep networks for surface normal estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 539–547, 2015. 3.2.2
- [37] Shichao Yang, Daniel Maturana, and Sebastian Scherer. Real-time 3D scene layout from a single image using convolutional neural networks. In *Robotics and automation (ICRA), IEEE international conference on*, pages 2183 – 2189. IEEE, 2016. 3.1
- [38] Shichao Yang, Yu Song, Michael Kaess, and Sebastian Scherer. Pop-up SLAM: a semantic monocular plane slam for low-texture environments. In *Intelligent Robots and Systems (IROS), 2016 IEEE international conference on*. IEEE, 2016. 3.1

- [39] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014. 4.1, 4.2.2, 4.2.3
- [40] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. *arXiv preprint arXiv:1512.04150*, 2015. 4.1
- [41] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013. 3.1, 3.2.1, 3.2.1