2019

Guided Autonomy for Quadcopter Photography

Saif Alabachi
University of Central Florida

GUIDED AUTONOMY FOR QUADCOPTER PHOTOGRAPHY

by

SAIF ALABACHI
B.S. Nahrain University, 2001
M.S. Nahrain University, 2007
M.S. University of Central Florida, 2017

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical and Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2019

Major Professor: Gita Sukthankar

ABSTRACT

Photographing small objects with a quadcopter is non-trivial to perform with many common user interfaces, especially when it requires maneuvering an Unmanned Aerial Vehicle (UAV) to difficult angles in order to shoot high perspectives. The aim of this research is to employ machine learning to support better user interfaces for quadcopter photography. Human Robot Interaction (HRI) is supported by visual servoing, a specialized vision system for real-time object detection, and control policies acquired through reinforcement learning (RL). Two investigations of guided autonomy were conducted. In the first, the user directed the quadcopter with a sketch based interface, and periods of user direction were interspersed with periods of autonomous flight. In the second, the user directs the quadcopter by taking a single photo with a handheld mobile device, and the quadcopter autonomously flies to the requested vantage point. This dissertation focuses on the following problems: 1) evaluating different user interface paradigms for dynamic photography in a GPS-denied environment; 2) learning better Convolutional Neural Network (CNN) object detection models to assure a higher precision in detecting human subjects than the currently available state-of-the-art fast models; 3) transferring learning from the Gazebo simulation into the real world; 4) learning robust control policies using deep reinforcement learning to maneuver the quadcopter to multiple shooting positions with minimal human interaction.

To my mother...

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Dr. Gita Sukthankar, for all of her guidance and persistence throughout the process of completing my PhD research. Thank you for bearing with me and put me on the track. I'm really proud to be one of your students.

I would also like to thank Dr. Rahul Sukthankar for the valuable feedback, time, and effort. I feel proud and fortunate to have the opportunity to work with him throughout my graduate research and for that I am deeply grateful.

I extend my thanks to the PhD. dissertation committee members: Dr. Aman Behal, Dr. Mingjie Lin, Ladislau Boloni, and Dr. Joseph J. Laviola for the valuable comments and their time to evaluate this dissertation. I also would like to express my appreciation to Diana Camerino for her kind help throughout my course of study.

Last but not least, I would like to express my gratitude to my family for their continuous support, especially my father for the love and support that helped make my dreams came true, and my wife who always continues to support me and believe in me even in the toughest of times.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

During the last decade, Unmanned Aerial Vehicles (UAV) (commonly referred to as "drones") have emerged from rarity into ubiquity; they are now commonly used in a wide variety of tasks including construction, agriculture, military scouting, transportation, and rescue missions. This dissertation focuses on the application of UAVs for photography and film making. UAVs can be embodied using many types of rotor configurations and aerial formations. Quadcopter or quadrotor is one of the most popular drones which is commercially available as a photography system or as a professional racing platform. The quadcopter possesses a high stability and movement flexibility in 3D space, and flies by the thrust produced by four rotors. We can monitor the drone by its linear and angular states in 3D space and control it via four input velocities (roll, pitch, vertical incline and yaw). Although some UAVs fly completely autonomously, most of them are guided by users with a variety of user interfaces including graphical, speech, natural, or gestural interaction modalities. Flying a quadcopter to accomplish a task requires specialized skills; when guided by inexperienced pilots they have a high incidence of collisions, crashes, and poorly framed photographs.

This dissertation focuses on the problem of automating quadcopter photography using drones with mounted cameras, capable of shooting scenes from viewpoints that are impossible for a human to capture with a ground-based camera. The rapid improvement in deep learning (DL) algorithms offers promise for solving many types of robot control problems, including aerial robot autonomy. One key ingredient for using these DL systems is having sufficient data. Most deep computer vision systems require large image datasets that capture the subjects' appearance in different poses and from multiple angles supported with ground truth labels in order to be useful for supervised learning. Although many systems are trained with images mined from the web, often it is hard to collect specific image categories that are less common online. Learning deep neural networks from synthetic data is a popular option in computer vision and robotics as it is cheaper to collect. How-

ever, completely synthetically generated datasets will have a high correlation in the constructed samples and lack visual effects that normally appear in real images (such as overlaps and blurring) which may negatively influence the learned model. Labeling and annotating are both expensive operations that most of the time require human-in-the-loop even with innovations in labeling tools.

Fortunately the drone is also a good tool for collecting data. For this dissertation, we created a Smart User Interface (SUI) that is specialized for photographing objects, is robust against navigation errors, and reliably collects high quality photographs. Once an object is selected, the drone autonomously photographs the object from multiple viewpoints and collects images with a wide range of frame offsets and sizes. With our SUI, building a large imagery dataset for custom objects can be done in real-time. Multiple image processing manipulations can be performed at the same time to create several versions of the frame associated with ground truth labels to increase the variations and reduce the correlation between successive samples for a better training experience. By retaining the human in the loop, SUI is faster and more selective than purely autonomous UAVs that employ simple coverage algorithms. The implemented approach operates in multiple modes, allowing the user to either directly control the quadcopter or fly in a semi-autonomous mode around a target object in the environment. Our interface is tested in a synthetic environment created by a 3D emulator and was evaluated through a number of user studies where the users had to complete a data set collection task in which they were asked to photograph objects from multiple views. Our sketch-based control paradigm facilitated task completion, reduced crashes, and was favorably reviewed by participants.

The second contribution of this dissertation is developing specialized computer vision algorithms to apply to the video feed coming from the drone frontal RGB camera for object detection and tracking. Object detection models based on Convolutional Neural Networks (CNNs) demonstrate impressive performance when trained on large-scale labeled datasets. While a generic object detector trained on such a dataset performs adequately in applications where the input data is similar

to user photographs, the detector performs poorly on small objects, particularly ones with limited training data or imaged from uncommon viewpoints and high perspectives. Also, a specific room will have many objects that are missed by standard object detectors, frustrating a robot that continually operates in the same indoor environment. We train our models with samples collected by a drone teleoperated with our SUI which can perform instant image augmentation to create sufficient variation in brightness, contrast, and rotation in addition to the photo-realistic image augmentation that is coming from the drone movement. With the auto-pilot agent engagement in our SUI, the user gains higher control and more concentration on performing the photography task. We present our approach, Dense Upscaled Network (DUNet), a real-time object detection model for learning customized object detectors from scratch, without the need for any pre-trained classifiers or transfer learning that is commonly used in limited data situations to increase the object detection inference performance. DUNet learns models from scratch from smaller datasets; this is an alternate approach to fine tuning with existing state of the art object detectors.

In photography, one of the most common shooting routines is taking selfies with mobile phones. To achieve a wider angle of view, users will often use a selfie stick. A UAV can improve on a selfie stick by capturing photos at distances unattainable for the human; however teloperating a drone to a good viewpoint is a non-trivial task. The third section of the dissertation introduces an approach for using the UAV as a replacement for a selfie stick, enabling the creation of personal photos that include more of the background scene. A natural interface for UAV photography is presented, the *Selfie Drone Stick* that allows the user to guide the quadcopter to the optimal vantage point based on the phone's sensors. The user points the phone once, and the drone autonomously flies to the target viewpoint based on the phone camera and the Inertial Measurement Unit (IMU) sensor data. Visual servoing is achieved through the combination of a dense neural network object detector that matches the image captured from the phone camera to a bounding box in the scene which is then used as input for a controller learned with deep reinforcement learning. Our architecture

is trained with a combination of real-world images and simulated flight, and reward shaping is used to accelerate the learning process. With our novel and carefully-selected parameterized state space, our agent can apply experience learned in simulation to the real-world; reward shaping is used to accelerate training. Maneuvering and photography test results show that integrating the RL controller produces stable movement and good photographs.

1.1 Thesis Statement

Even with minimal human supervision, limited image data and no real-world experience, a carefully constructed deep learning architecture trained with reward shaping can achieve good visual servoing performance on human photography tasks.

1.2 Overview

This dissertation presents a cohesive solution for improving quadcopter photography using a combination of intelligent user interface design and deep learning both object detectors and control policies. Chapter 2 presents a detailed overview of related work on quadcopter photography. Chapter 3 describes the hardware and software infrastructure. The relationship between the different components of this dissertation is illustrated by Figure 1.1. The three main contributions are:

- A semi-autonomous Human Robot Interaction (HRI) system which is robust at performing real-time data collection and operates with minimal human guidance as a co-pilot using a new mode of interaction that combines gestural strokes for direct quadcopter maneuvers and target object assignment
- The state of the art drone-based system object detector, DUNet, which is customized for the objects in the user's current environment. With its novel architecture, DUNet can achieve

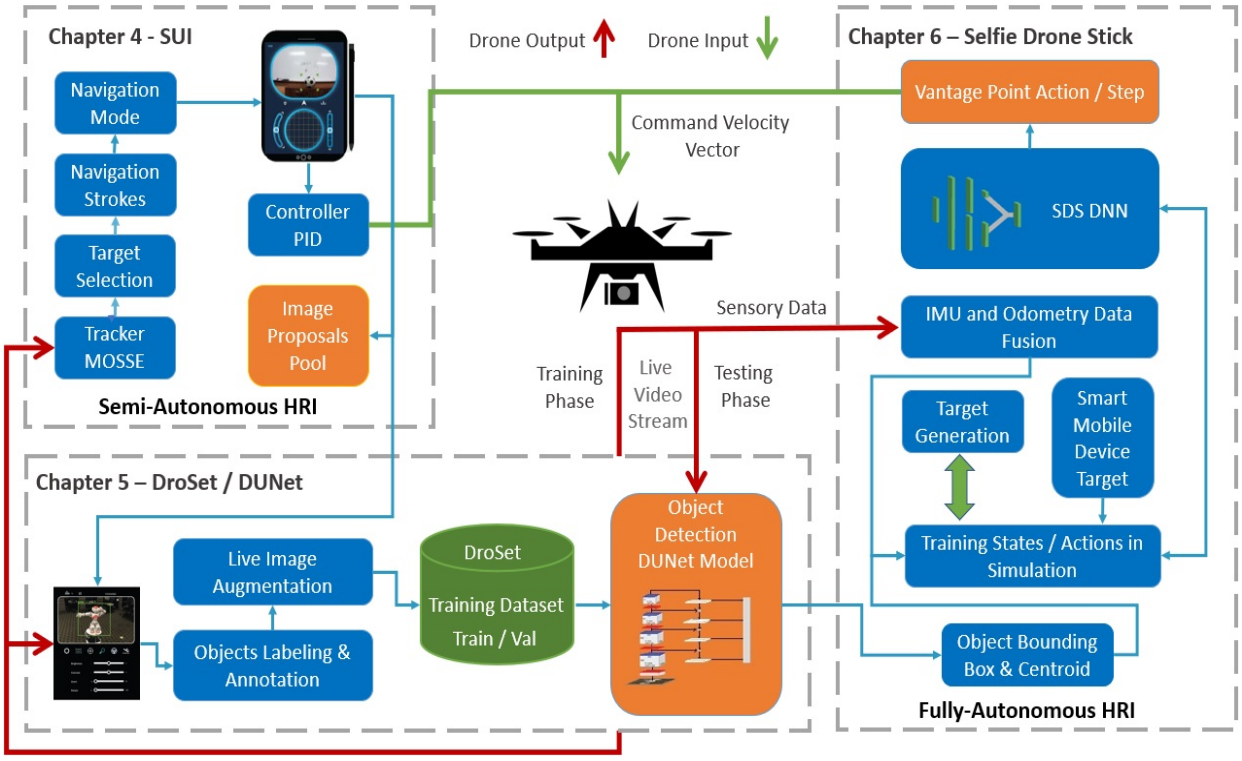


Figure 1.1: Dissertation roadmap. The results of our three main contributions are shown in the orange boxes and the supporting modules inside the blue boxes. The dashed boxes group the algorithms by the chapter.

real-time detection of small objects with high accuracy that functions well on an aerial robot.

- Selfie Drone Stick, which autonomously navigates the UAV to the vantage point assigned by the user guided by a selfie captured with any smart mobile device. We demonstrate how reward shaping can be used to accelerate learning a control policy with deep reinforcement learning.

CHAPTER 2: LITERATURE REVIEW

This chapter presents a survey of key related research areas including autonomous photography, object detection, quadcopter user interfaces, and learning aerial control with reinforcement learning.

2.1 Autonomous Photography

A variety of approaches have been employed by mobile robots performing autonomous photography tasks. Human detection is commonly used to guide the robot towards the subjects. For instance, combining face and skin detection models with pre-defined cinematography criteria [21, 22, 109] or analyzing depth from motion for well-framed group photographs [23]. In some cases, authors (e.g., [48]) have relied on SLAM (Simultaneous Localization and Mapping) to localize the robot before capturing photos. Aesthetics can be important; Kai et al. [64] implement an autonomous photography system that searches for the optimal target viewpoint for capturing a pleasing photo with a ground-based robot. With the rapid growth of commercial drone markets, focus has shifted away from ground robots towards aerial systems. The quadcopter has become the platform of choice due to its good hover mode and ability to flexibly change direction [17, 25, 113]. Even with aerial robots, human detection still plays an important role in automating photography. Photographing human athletic events and exercise routines from high perspectives is a popular application area [47, 80]. Several systems have also employed face detection such as Roberto et al. [115] who use the Haar cascade classifier [116] for head detection on a quadrotor. [28] detects head position and analyzes the landmarks of the detected face with the millisecond face alignment method [56] to infer orientation. However, neither group tackled the problem of multiple vantage points or background aesthetics.

The mounted high resolution cameras of the new generation UAVs [85] have motivated work on automating photography and videography [26], creating dynamic scenes from high and hard multi-view points [83], object recognition [89], and recording cinematography videos [55]. However many tasks are too ill-specified to be performed autonomously, necessitating the development of specialized user interfaces. One specific problem, tracking and photographing humans, is particularly interesting since humans are good subjects for photography and can be easier to track. However, there are a variety of visual inspection and surveying tasks that require photographing arbitrary objects; our prototype user interface is specialized for handling those types of problems.

2.2 Object Detection with Convolutional Neural Networks

Object recognition has a long history in computer vision [99]. The field saw major advances due to the resurgence of neural networks, specifically deep convolutional neural networks, initially for the task of image labeling [63] and subsequently for detection [111]. The image labeling task such as ImageNet [30] and LabelMe [100] are most relevant to information retrieval and requires assigning to each input (image), a class label corresponding to the dominant semantic category visible in that image. By contrast, object detection or localization, consists of drawing a bounding box around each object (from a set of relevant categories) in the image, along with its semantic label – and is thus of direct relevance to robotics. We briefly cover both the approaches and relevant datasets below.

Image datasets are typically sourced from the Internet but there is also a growing trend of datasets, particularly for robotics applications, collected directly from the real world. For instance, the KITTI dataset [42] consists of roadway images taken from a car driving in an urban environment. There has also been significant recent progress in efficiently collecting large quantities of visual data using robots, including smart user interfaces for semi-automated data collection using drones

(e.g., [8]) and indoor mobile robots (e.g., [74]). In addition to its labeling tools, ImageNet consists of almost a million images from 1000 categories, each labeled manually by human raters. However, most modern object recognition relies on large labeled datasets. By contrast, the PASCAL VOC [34] (21 classes) and MS COCO [70] (80 classes) consist of fewer images, but each image is laboriously labeled by crowd-sourced workers with the location of each object appearance in the frame from the selected categories.

Convolutional neural networks (CNNs) [66] were initially applied to handwritten digit recognition but were shown to outperform traditional techniques such as deformable part models [35] on image labeling in AlexNet [63]. Since then, there have been consistent improvements to the state-of-the-art based on extensions to CNN-based architectures, such as VGG [108], GoogLeNet [110] and DenseNet [49]. CNNs were also instrumental to recent progress on object localization, starting with MultiBox [111]. Inspired by classification models, R-CNN [44] used cropped boxes from the original image as input to a neural network classifier. Unfortunately, R-CNN was computationally expensive since it repeatedly processed the same pixels whenever they appeared in different overlapping regions. Fast R-CNN [43] addressed this defect by first pushing the entire image through a feature extractor, thus amortizing the computation across the set of anchor boxes. This set of ideas has culminated in Faster R-CNN [97], where region proposals are efficiently generated using a fully convolutional network. While Faster R-CNN can process several images per second, it is typically still too slow for most mobile or robotics applications that demand real-time performance on compute-constrained platforms. This has motivated a series of object detection models, such as SSD [73] and YOLO [96] that aim for high quality detection at near real-time speed.

Our work is informed by the comprehensive experiments on object detection speed/accuracy trade-offs conducted by Huang et al. [50], where SSD + MobileNet emerges as a very strong baseline for our application. However, we saw opportunities for improving customized object detectors, drawing inspiration from recent work on feature extraction in DenseNet [49], fully-convolutional

approaches to semantic segmentation such as Tiramisu [54] and recent multi-scale approaches for object detection, such as FPN [69] and TDM [106]. The standard approach to customizing an object detector is via domain transfer — e.g., replacing the final layer in a strong pre-trained model and fine-tuning it on the new data. However, we see significant advantages to training custom object detectors from scratch, such as DSOD [105], which demonstrates competitive performance, albeit not in real-time.

Thus, our proposed approach for customizing real-time object detectors, termed Dense Upscaled Network (DUNet), is an architecture inspired by SSD, DSOD, FPN, TDM and trained from scratch on data collected semi-autonomously. In addition, both realistic and synthetic visual augmentation techniques are applied to captured frames producing hundreds of variations in the data samples. These variations are normally applied to the image samples of a randomly assigned batch before feeding it to the network for training the object detection model in order to increase its performance.

Using quadcopter mounted cameras, many researchers have applied tracking and object detection algorithms to address different problems associated with autonomous navigation. For instance, [57] integrated an on-board low-frequency vision system with parallel PID controls to perform object tracking by analyzing the offset and size. Most quadcopter vision systems have employed the front mounted camera (as ours does) but some have also leveraged the downward facing camera for landing tasks. By mapping the calculated homography from the visual marker, the drone distance to the landing surface can be estimated either with a PID controller [24] or fuzzy logic [11].

2.3 Quadcopter User Interfaces

Natural user interfaces (NUI) rely on innate human actions such as gesture and voice command for all human-robot interaction [36, 76, 84, 93]. Alternatively, more precise navigation in indoor and outdoor environments can be achieved through structured waypoint designation strategies [8, 41, 72]. Wearable sensors were employed in a point-to-target interaction scenario to control and land a drone using arm position [45]. Our system removes the need to employ gestures, hand crafted strokes, or wearable devices. Any mobile device equipped with a camera and IMU sensors can be used to direct the quadcopter using our *SelfieDroneStick* interface.

We considered many candidate interface modalities when designing our system, including gesture, voice, gaze and EEG, which have been successfully used in other quadcopter systems. Unlike many sketch-based robot control systems (e.g., [29, 98, 102]) in our system the user designates the target on a canvas displaying the robot's view rather than on the global map. A subset of the human-robot interaction research has specifically addressed the problem of user interfaces for drone-mounted cameras. For instance, [8] tracks user-specified objects with an adaptive correlation filter in order to create photo collections that include a diversity of viewpoints.

Ziquan et al. [65] designed a touch-based system for photo taking in which the user concentrates on adjusting the desired photo rather than on specifying the quadcopter flight path. Their framework, XPose, is a semi-autonomous system that offers an innovative, powerful user interface for taking a variety of photos, supported by integrating an ORB SLAM system [82] for localization and 3D trajectory planning in a GPS-denied environment. Unlike XPose, our system does not require a SLAM system and hence is robust to localization errors.

2.4 Aerial Control Policies

Automating flight control in an unknown environment has been achieved using both machine learning and non machine learning methods. In our literature review we do not include systems that use GPS or motion capture (e.g., [71, 75, 77, 81]). Most of the success in autonomous navigation has been based on analyzing frames that are live broadcast from the UAV camera in addition to the odometry information that is fed in real-time through mounted IMU sensors [14, 32, 40, 61, 103]. Localizing the aerial robot in 3D space can be performed by SLAM systems that leverage the tracked feature descriptors captured in the video [37, 38, 60]. [33] used PTAM on monocular camera frames, a PID controller, and an extended Kalman filter to automate quadcopter navigation in both indoor and outdoor environments, whereas [82] uses ORB-SLAM to track keyframes for localization. However, SLAM systems are very sensitive to the changes that may occur in the environment. Hence, our system eliminates the need for an expensive SLAM matching technique. Instead a robust object detection model is used to detect the subject to be considered as the reference which the quadcopter uses to localize itself.

Reinforcement learning has been employed to learn flight control [16, 51] as an alternative to complex feedback linearization [117, 118] to stabilize the non linear dynamic behavior of aerial robots. It has also been applied to learn Markov Decision Process models for navigating in an indoor environment [119]. A PID controller and Q-learning algorithm are used together to guide the quadcopter autonomously to navigate to the target in an unknown environment [88]. Deep RL has been used to learn specialized flight controllers since many researchers have been inspired by recent successes in learning control policies in both discrete and continuous action space [31] in games and simulated robots [20]. For instance, Deep Q-Network (DQN) was used to learn autonomous landing policies for a quadcopter with a downward facing camera [91]. It also has been utilized to capture frontal facial photos; [86] developed a realistic simulation for the task and

trained their system on a database of head positions. In order to use DQN, both these systems adopted a discretized action space. In our system, we evaluated the performance of both DQN and Deep Deterministic Policy Gradient (DDPG) and found that DDPG produced trajectories with less oscillation, particularly when combined with our reward shaping method.

Most machine learning controllers are learned in simulation or game worlds and need to transfer the learned experience to the real-world [18, 46]. For these systems, adding dynamics randomization [87, 95] is known to be important for successful training. Synthetic data can be randomly ordered during training; in the real-world noise can be added to increase robustness to the system [53, 114]. One notable exception where only real-world data was used is Gandhi et al. who learned UAV obstacle avoidance policies by collecting a large dataset of crashes [39]. [112] learned policy controllers for a quadruped robot through formulating locomotion control as a POMDP and solving it using a policy gradient method, Proximal Policy Optimization (PPO) [104]. Our controller is learned using both real-world and simulated environment images to train DUNet; then its output is combined with Gazebo flight trajectories for building the generic state-space to simplify and smooth the transformation of the learned experience from the synthetic environment to the real-world. For this project, we selected DDPG to learn our flight control. It exhibits faster learning due to the use of a deterministic policy in the critic network to guide the optimization of the stochastic policy in the actor network. We support our learning method with off-policy learning and target networks [78] as we have a high-dimensional continuous action space. In order to capture the perfect selfie, our learned controller must be able to execute multiple types of flight paths and is not limited to a single maneuver.

CHAPTER 3: PLATFORM

Building a system that insures the performance and simplicity of human-robot interaction with a multi-purpose UAV is a challenging problem. Therefore, it is important to design a reliable platform where the communication between different entities can occur. This chapter describes the hardware and software components that are used in this project.

3.1 UAV and 3D Environment

In this project, a quadcopter (or a quadrotor) UAV is used which has four rotors horizontally attached at the endings of equal-length arcs of a cross-body shape. Each rotor is attached to a brushless engine which is controlled by a micro-controller at the drone center. With this structural body, the wind resistance is improved resulting in more hovering stability and smoother movement in any direction.

3.1.1 Augmented Reality (AR) Drone 2.0:

The experimental work is performed on the Parrot Augmented Reality AR.Drone 2.0 [90] which is one of the most popular drones in the robotics research community. The second generation quadcopter was released in 2012 with several improvements over the previous version that was presented at International Consumer Electronics Show (CES) in 2010. The AR.Drone 2.0 with its indoor and outdoor hulls is shown in Figure 3.1. Below are the main hardware specifications, sensors, and software of AR.Drone 2.0:

- **Processing unit**: The main board is mounted at the center of the drone, and it holds a 32 bit ARM 1-GHz Cortex A8 processor, a DDR2 1-GB of RAM. and a Camera image signal

processor for image format conversions and sensor interfaces.

- **Navigation board**: This board holds the micro-controller that controls the drone motors by converting the DC power from the battery into 3-phase AC power to run the brushless motors that turn the rotors, and the IMU sensors that provide estimations of the drone motion and state as listed below.
  - 3-axis accelerometer is used to estimate the orientation of the drone with respect to the world coordinate system.
  - 1-axis yaw-precision gyroscope is used to measure the yaw angle with a drifting noise of up to 60°per minute, and a 2-axis gyroscope is used to obtain the roll and pitch rotation angles with respect to the drone coordinate system as illustrated in Figure 3.1.
  - Ultrasound altimeter is used to estimate the drone altitude up to 6 meters.
  - Pressure sensor is used to measure the altitude beyond the ultrasound altimeter sensor limit at any height but with ± 10 Pa precision in its altitude estimation.
  - 3-axis magnetometer can be used in the absolute control mode.
- **Cameras**: AR.Drone 2.0 has two cameras which differ in their properties and usage. The frontal camera streams up to 1280×720 pixels HD quality video at a rate of 30 fps and covers 92°field of view. The down-facing camera streams QVGA video with 320×240 pixels resolution at 60 fps frame rate and has 64°diagonal wide angle lens. The second camera is mostly used to measure the horizontal displacement and ground speed through tag detection, otherwise the resolution can be scaled up to $360p$ or $720p$ for video streaming.
- **USB Port**: Firmware update, flashing and debugging can be done using a supported USB socket. This version 2.0 high speed USB port is also used to record real-time video streaming from either of the two cameras.
- **Network**: The network between the AR.Drone and the ground station or a smart mobile device is supported by WiFi b/g/n via the supported wireless LAN on the following ports:
  - **UDP-5554 port:** The AR.Drone engines rotation speed and the estimated velocity

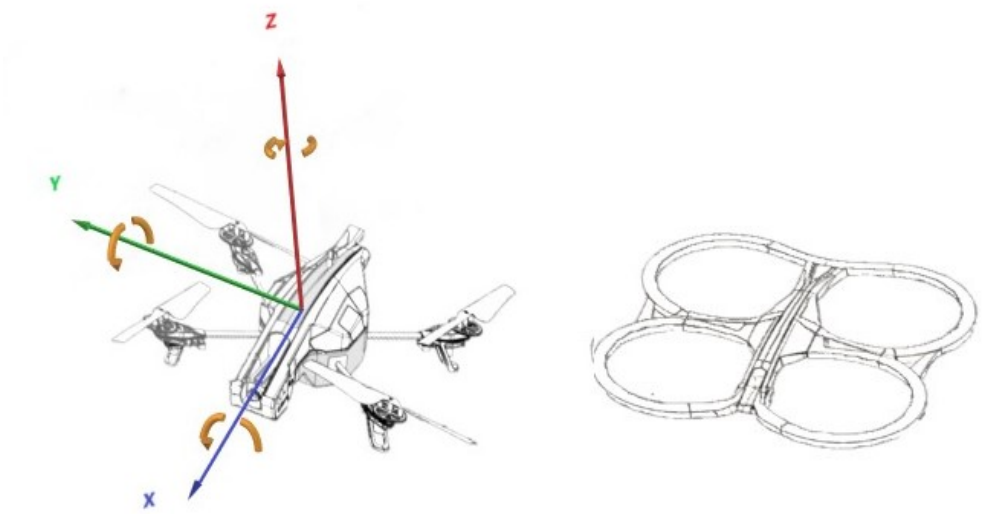


Figure 3.1: Quadcopter AR Parrot Drone 2.0 and hull

and position are sent through the User Datagram Protocol (UDP) on 5554 port. This information comes with *navdata* header and is sent by the drone to the connected client. *navdata* also includes tag detection records which are used for creating augmented reality games and sent in a $rate \approx 15$ times per second in demo mode, and $rate \approx 200$ times per second in debug mode.

- **TCP-5555 port:** The video streaming occurs on this TCP port for AR.Drone 2.0 where the successive frames pass through the H264 encoder if any are coming from the frontal camera or the down-facing camera.
- **UDP-5556 port:** Drone piloting commands can be passed through this control port including takeoff, land, rest, and navigation which are sent regularly around 30 times per second. The navigation commands are also communicated using this port as a six-element vector corresponding to three linear velocities and three angular velocities which the drone receives as a message on a specific topic. Four values are the most

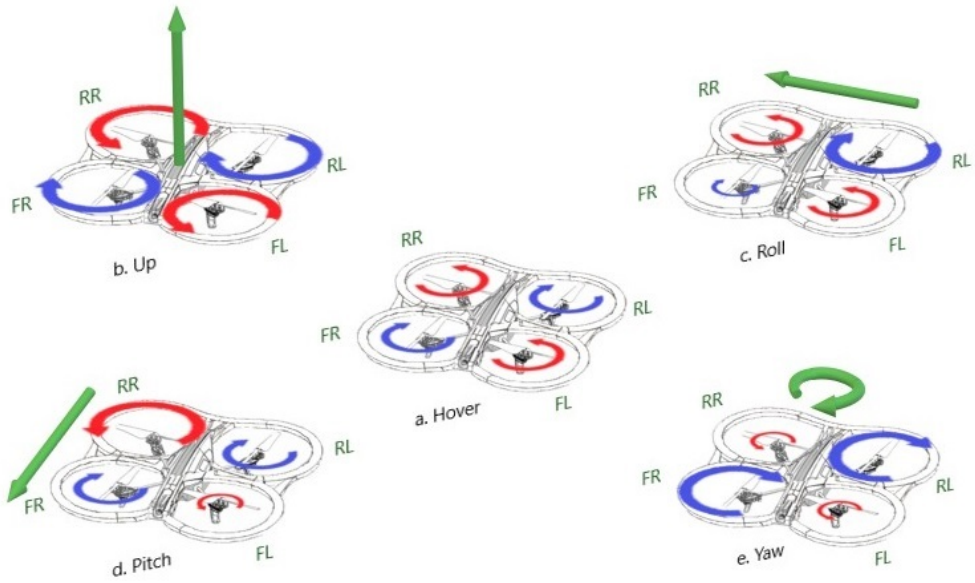


Figure 3.2: A quadcopter has four rotors; the Front Right (FR) and the Rear Left (RL) turn clockwise while the Front Left (FL) and the Rear Right (RR) turn counter-clockwise. By changing the motor velocities of these rotors we can: **a**. hover: all the rotors have the same constant speed. **b**. move up: all rotors speed up above the hover constant rate with the same amount. **c**. roll to the right: increasing RL and decreasing FR whereas the FL and RR remain with no change. **d**. pitch forward: increasing RR and decreasing FL. **e**. yaw clockwise: Equally increasing RL and FR. The velocities are changed based on how fast we want to travel.

important which are the signed (either -ve or +ve for directions) and normalized velocities of the roll, pitch, yaw, and vertical height. If all values are equal to zero, the drone will stay steady in hovering mode.

  - **TCP-5559 port:** Custom settings, parameters attenuation, and critical acknowledgement data are reserved for this channel such as such as changing the video streaming port.

- **Operating System (OS)**: AR.Drone 2.0 has a GNU/Linux 2.6 that runs by the ARM processor to manage the wireless connection, IMU data estimation, and control tasks.

- **SDK**: The Parrot developer community has official software named AR.FreeFlight which contains the AR.Drone driver and a touch based application that is available to download from iOS and Android stores and installed on a smart mobile device to be used to fly the drone. AR.FreeFlight shows either the frontal camera or the downward camera at once on the mobile screen and allows for photography capturing and video recording. Another ROS-based SDK is publicly available and is widely used by robotics researchers in order to develop intelligent systems for automating drone tasks.
- **Battery**: The official battery that comes with the standard edition of the AR.Drone 2.0 is 1000mAh and allows a flight time of approximately 10 minutes.

Basically, any quadcopter can roll both sides, pitch front or back, yaw clockwise or counter-clockwise, and throttle up and down. However, the quadcopter can move in 83 possible ways if we consider different unique combinations with a single assignment constant velocity. The flying mechanism of AR.Drone 2.0 or any other quadcopter is similar and based on the rotation rate and direction of the rotors. Each two diagonally opposite rotors run in the same direction. In hover mode, all rotors rotate at the same constant velocity which is set depending on the required altitude. Movement is done by changing the rotation rate of the rotors individually as each rotor has a separate motor, for instance, to fly up, we can increase the rotation rate of all the rotors equally by $+\Delta\nu$, and going down can be done through decreasing the rate by $-\Delta\nu$. Rolling AR.Drone to the right can be done by increasing the rotation speed of the rear left motor, which turns clockwise, by $+\Delta\nu_r$ and decreasing the velocity of the front right motor, which runs the diagonally opposite rotor and turns clockwise, by $-\Delta\nu_f$. Pitching AR.Drone to move forward can be done by decreasing the speed of the front left motor, witch turns counter-clockwise, by $-\Delta\nu_f$ and increasing the speed of the rear right motor, which turns counter-clockwise, by $+\Delta\nu_r$. Yawing AR.Drone to the right can be done by increasing the velocities of both of the front right and the rear left motors by $+\Delta\nu$ and decreasing the velocities of both of the front left and the rear right motors by $-\Delta\nu$. Examples

Figure 3.3: Simulated AR.Drone model with video streaming from the frontal camera along with readings from its sensors in real-time inside a 3D custom environment generated via the Gazebo simulator.

of the drone movements (linear velocity commands) and orientation (angular velocity command) in the 3D space are illustrated in Figure 3.2.

3.1.2 Gazebo 3D Robot Simulation:

Testing and debugging with physical quadcopters are hard since crashes are frequent and the battery life is short. Using a simulated testing environment can reduce a lot of time and effort during the development stages. Therefore, researchers use simulation to implement their algorithms before moving to real-world environment. Examples include the commercially available COSIMIR [2] and Webots [6], and the open source simulators such as Darwin2K [67], OpenSim [4], and Gazebo [62].

In this project, the *Gazebo* 3D simulator is used which is an open source framework supported by a data visualization user interface. With Gazebo, custom and complex worlds with next generation mobile robots possessing a variety of actuators and sensors can be created. Flexible integration with the ROS framework can be established through *gazebo_ros_pkgs* a ROS-dependent package that allows for sensory reading and command transformation. A 3D realistic world environment is created with Gazebo default models along with other designated models for the purpose of our learning algorithms. Human models are created via the makehuman [3] open source software and a simulated model of the AR.Drone [1] quadcopter is used as a custom model inside the Gazebo 3D dynamic emulator. The simulated AR.Drone reflects the behavior of the actual robot through interacting with its sensors and cameras. The interaction experience is collected from the real-time camera streaming of the synthetic visuals created from the simulated environment, and the odometry data is observed from the simulated version of the IMU. The simulated drone, human subject, and the environment are illustrated in Figure 3.3.

3.2 ROS: Robot Operating System

Robot Operating System (ROS) is an open-source framework hierarchically structured to work as a network backbone between different platforms. It has a flexible toolkit that can be used to collect diagnostic information, visualize real-time states, and plot the behavior of the connected robot for issues addressing, troubleshooting, and debugging with much less time and effort. The built-in conventions and protocols connect multiple software programs running on different computers with each other. In addition, it has multiple client libraries that facilitate the building of data manipulation programs and intelligent agents to automate the interaction with a wide range of robots via ROS-dependent Software Development Kits (SDK). For instance, one of the designated tasks of our SUI [8] is to pass certain low-level robot control commands from a non-ROS program directly to the UAV via ROS connectivity in a semi-autonomous interaction scenario with the

intelligent agent. The next sections describe ROS file system resources, ROS computation graph which is the peer-to-peer data processing network, and the ROS toolkit.

3.2.1 ROS Filesystem

The head of the ROS filesystem starts with the **workspace** directory (most commonly named as catkin_ws since ROS indigo version) which contains and arranges the software programs as **packages** in ROS. In the workspace, we can modify, install, and build catkin packages and separate the low-level robot control software (SDK) from the high-level intelligent runtime processing units. This architectural separation allows us to substitute the physical robot with a simulated version during debugging and development stages at any time without affecting the original robot SDK. Packages may also contain libraries, datasets, etc. Inside these packages, other sub directories and files can be created. The most important are:

- Metapackages or stacks which are a group of related packages that are specialized for accomplishing related or sequential tasks. The purpose of stacking packages is to share the common information.
- Msg subdirectory which holds custom package-related messages with flexible selection capability among various data types. For instance, we can publish **Binary** True or False, **String** text, **Twist** velocities command vector, or **Image** message type that holds a frame array for photography transmission, etc. It is also possible to publish a fixed or a variable length list containing different data types arranged as sub_fields inside the custom-built message file. ***.msg*** files work as templates with sub fields to hold the information published by program instances or through the command line directly executed in the shell. Our sds.msg message is shown in Figure 3.4. This message is used to publish the data obtained from the observations and located inside the *selfiedronestick* package that is discussed in details in chapter 6.
- Srv subdirectory holds the **.srv** files used for bi-directional requests and responses in the

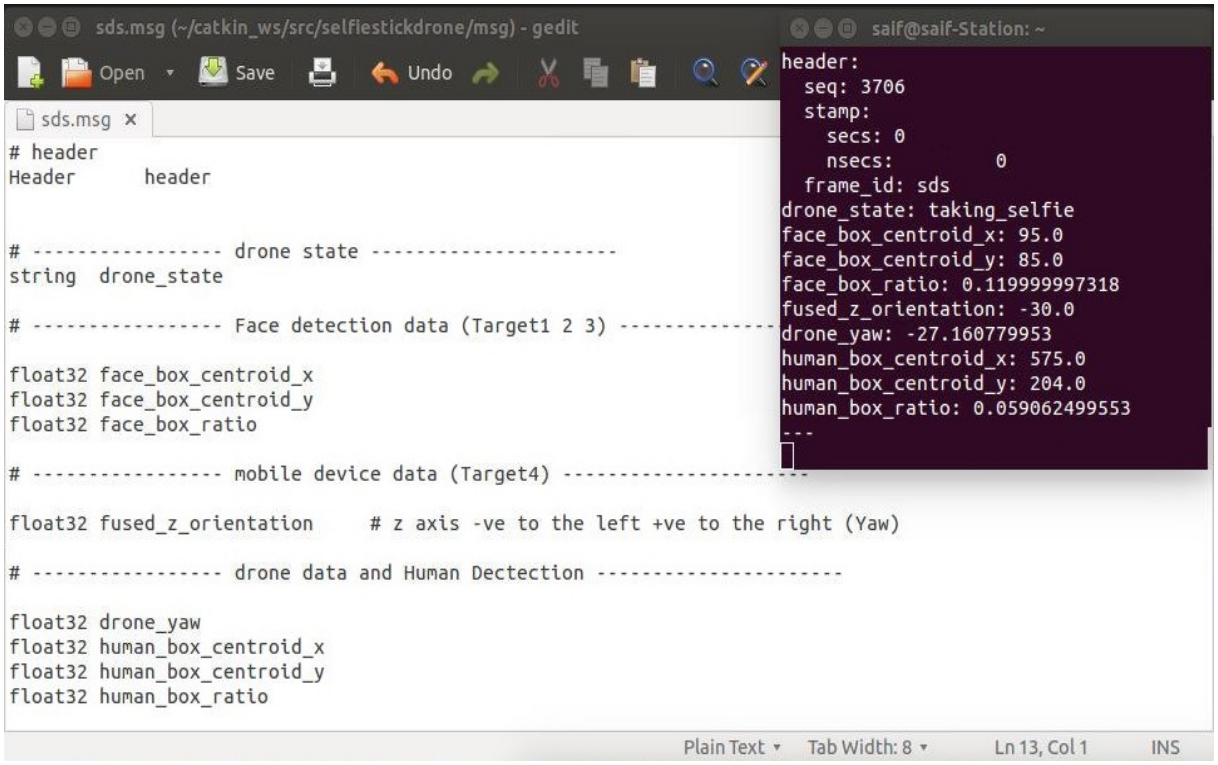


Figure 3.4: The **sds** message published by the *selfiedronestick* package through */sds/state* topic.

ROS distributed system.

- Configuration files ***.cfg*** hold parameters for the node. These parameters can be modified dynamically through the *dynamic_reconfigure* package without having to restart the node. ***.yaml*** files is another way used to configure ROS nodes. These files can be loaded, modified, or dumped using **rosparam** command line. In this project, yaml files are used to set the variables of the custom 3D model that is created by Gazebo simulation and used as a simulated environment for the AR.Drone to explore and learn SDS agent through the designed framework that is explained in details in chapter 6. Also, it is used to set SDS DNN hyper parameters and the learning configuration.
- Launch directory holds the XML-based ***.launch*** files and can be called using **roslaunch** command-line. Launch files enable launching multiple ROS nodes at the same time even if

these nodes are from different packages. It is also possible to set parameters inside these launch files. This project benefits from launch files in order to engage the related nodes (intelligent agents) from different packages all together at the time of the AR.Drone initialization.

3.2.2 ROS Graph Network

ROS has a **master** centralization unit that handles naming and registration of the **nodes** in the entire network. These nodes form the graph entities and are arranged inside ROS packages. Each node is a run-time processing instance and can allocate any other node within the graph by using master node services and communicate with any other registered node using three different types of communication procedures which are **services**, **parameter server**, and **topics**. Services are a Remote Procedure Call (RPC) interaction between nodes through request / response one-to-one bi-directional transportation. For instance a node may offer a service with a certain data type following the ROS unique naming convention and any other node requesting this service waits until receiving a reply. The parameter server is used by the nodes to share their global configuration parameters arranged in multivariate sub-fields with other nodes in the network. Topics are registered buses used to transport anonymous and asynchronous messages in the network by the publishing nodes; any node can register or select the appropriate topic and shares information so that any other node (or nodes) can directly receive this information by registering as subscribing nodes to that topic and receiving the published messages, For instance, a program (package) has a node that registers the appropriate topic by its name and publishes direct control commands. The node inside the robot SDK package can receive these commands after registering as a subscriber by the master node services. We can visualize the relationship between the publishers and subscribers nodes through viewing them with the ROS **rqt_graph** interface as illustrated in Figure 3.5. For instance, **selfie** node shares information on */selfie/state* topic with **phone_pub** node and other subscribers

and publishers in the graph.

3.2.3 ROS ToolKit

ROS provides powerful diagnostic tools and helper plugins that can be used in the development phase. One of the most important services is the data logging and playback that can be performed using ROS **bags** which are files with the ***.bag*** extension that allow us to record ROS serialized message data as it is received so we can play them back at any time in the same way as having real robots with nodes publishing these messages, For instance, sensory data readings that are transferred between the nodes can be recorded so we can develop our processing units having the same data inputs and that reduces a lot of time and effort than if testing were performed with physical robots. In addition, there is a variety of tools for processing, analyzing, and visualizing these bag files. Another built-in functionality is the **log** messages which are used for debugging and informative purposes including DEBUG, INFO, WARN, ERROR, and FATAL.

Multiple nodes can be initiated simultaneously by registering their unique names and their parental packages inside a ***.launch*** file. The purpose of simultaneous run is to perform multiple tasks in real-time such as localization, object recognition, and autonomous navigation before sending the resultant command velocities to the UAV SDK package. The implemented nodes make use of other ROS libraries which are publicly available as helper packages as listed below:

- vision_opencv: A real-time interface between ROS and openCV computer vision library [19]. This package is used to convert images between ROS Image messages and openCV images via its internal **cv_bridge** package.
- gazebo_ros_pkgs: A collection of ROS packages that provide the required interfaces to use Gazebo 3D simulator. The integration between these packages and Gazebo is performed through ROS messaging system, dynamic reconfiguration, and services.

- web_video_server: This package is used to stream the successive frames from the AR.Drone frontal camera as a message of type **Image** on */ardrone/frontal_camera/image_raw* topic via HTTP protocol on a specified port; It waits for the HTTP request on the opened port by subscribing to the corresponding topic to create the video encoder instance to be sent to the client side. Streaming can be provided with multi-variate video encoding formats with this package and it is possible to stream to multiple devices using via this web server.
- rosbridge_suite: A meta-package used to interact between ROS API and non-ROS frontend programs. In our project, the non-ROS programs are in the SUI which is implemented using roslibjs JavaScript-based library and HTML5. Rosbeidge has a protocol that can be used to send JSON commands to a robot with a ROS-dependent SDK package such as sending velocity command message of type **geometry_msgs/Twist** to AR.Drone using */cmd_vel* topic. The connection to the browser is performed by the **rosbridge_server** internal package on the transport layer by providing a **WebSocket** connection. In this project we use this package to interact with the JavaScript-based Roslibjs library.

ROS commands-lines can be executed directly in the shell. Here are some of the most common used commands:

- rosrun can be used to run any programming instance.
- roslaunch is used to set parameters and run multiple nodes simultaneously by compiling the *.launch* files. **roslaunch** is often used instead of executing nodes with **rosrun** command-line as roslaunch fires **roscore** automatically.
- rostopic is used to inspect ROS topics and visualize the messages, publishers, subscribers, and publishing rate.
- rosmsg is used to investigate the message type.
- rossrv command-line can be used to display service information.
- rosbag used for recording and playback .bag files.

Full ROS documentation is available in [94] and at the ROS official website with a wide range of publicly available packages. Two main packages are implemented to carry all the intelligence and data manipulation between the physical or simulated quadcopter and the client or the ground station as below:

- ucf_ardrone: This package contains the processing unit for the AR.Drone semi-autonomous navigation agent, along with other interaction and data manipulation operations with the designated SUI. More information about this package is presented in chapter 4.
- SelfieDroneStick: The main package of this project which links with the ROS libraries and other implemented packages and also with the other non-ROS components such as the web server, Gazebo simulation, and the training platform. More information about the functionality of this package and the implemented intelligent agents are presented in chapter **??**.

3.3 Machine Learning Tools

Developing intelligent agents to automate navigation and photography tasks can be achieved by training Deep Neural Networks (DNN) which requires a lot of numerical computations and time as we use Supervised Learning (SL) and Deep Reinforcement Learning (DRL) to learn the weight parameters of large-scale networks. These two deep learning methods with different embedded algorithms are implemented through the use of the **TensorFlow** library [7] because it has the ability to handle the continuous update procedure efficiently by creating data flow graphs with nodes that represent mathematical operations between the graph edges which represent the multidimensional arrays or **tensors** that hold the actual data. TensorFlow is provided by Google. It has APIs in different programming languages and is supported with a suite of visualization tools called **TensorBoard** which makes debugging and network optimization easier. The convergence of the classification loss, regression loss, and total loss functions of our autonomous photography agent

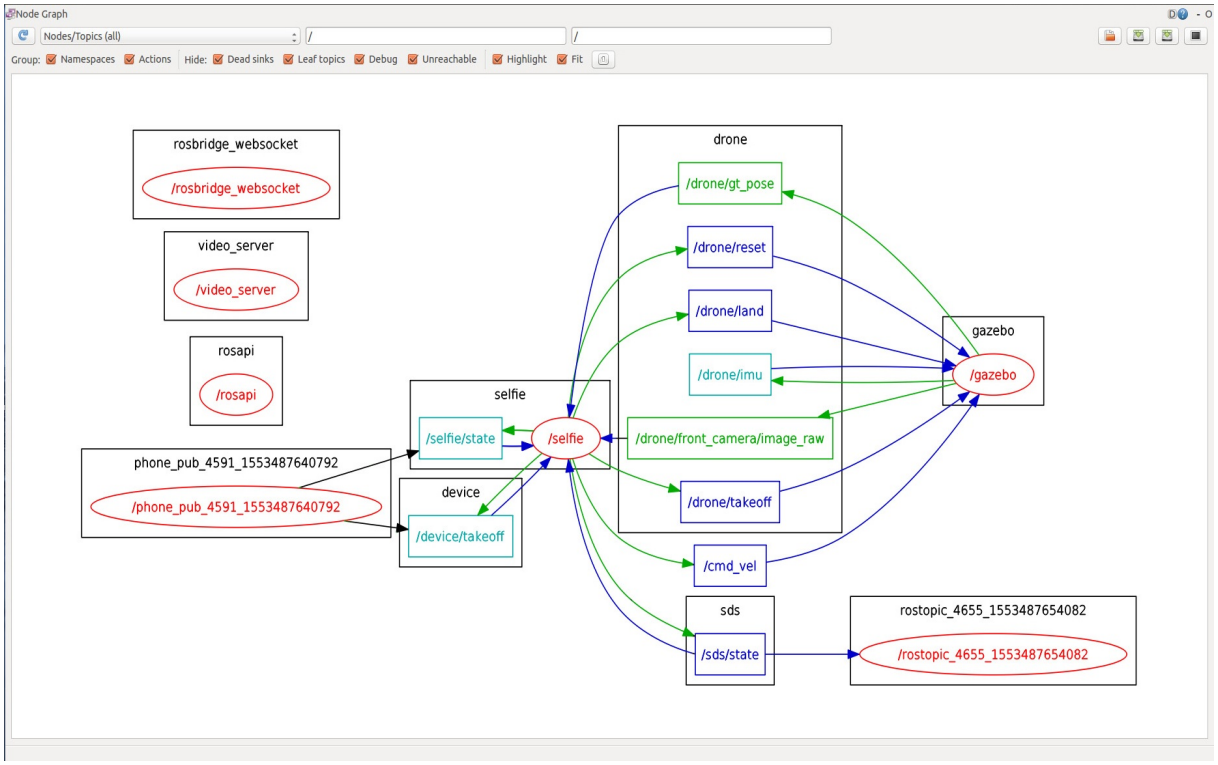


Figure 3.5: The rqt_graph of our SelfieDroneStick with simulated AR.DRone and Gazebo 3D emulator showing the nodes, topics, and messages between ROS packages.

are all shown in Figure 6.7. The object detection model is implemented via an open-source high-level neural networks API Keras [27] with the TensorFlow backend. Keras is developed as part of the Open-ended Neuro-Electronic Intelligent Robot Operating System (ONEIROS) project. Using Keras enables us to extend our model easily by reducing the cognitive load of altering and editing different components; for instance, it has the most common neural-network building blocks such as layers, optimizers, and activation functions. Thus, we adopt this API to perform fast experimentation with minimal actions.

In order to train an AI agent to autonomously navigate the aerial robot, it needs to learn from observations that can be collected by exploring the surroundings of the environment. Therefore, the sensory readings and the camera-streaming frames coming from the AR.Drone are arranged

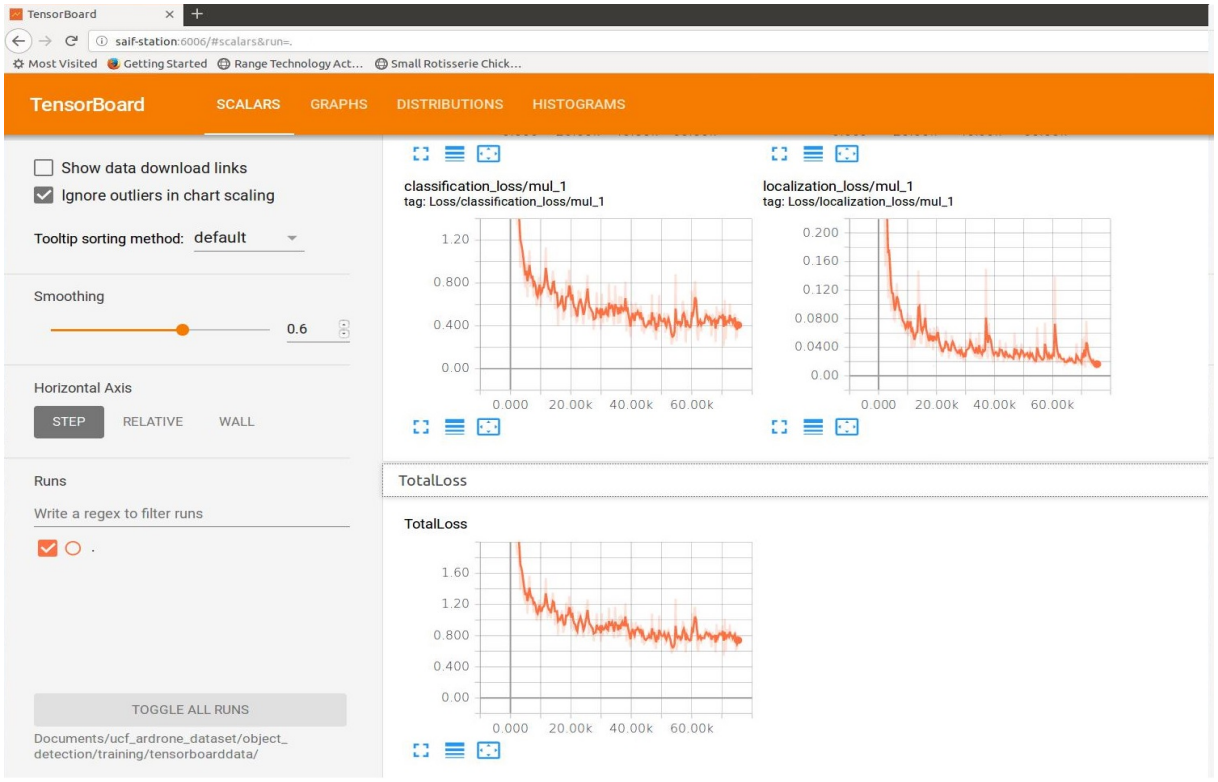


Figure 3.6: Tensorboard visualization interface plotting the classification, regression, and total loss of the learned photography agent.

episodically through integrating the **OpenAI Gym** toolkit [20] which can be integrated with TensorFlow to develop DRL algorithms. This open-source library provides several procedures and functions in addition to the big Atari games library [78]. Also, OpenAI Gym has a library specialized for robotics with 2D and 3D environments and robots. However, we integrate our Gazebo custom environment and the simulated AR.Drone with OpenAI Gym toolkit, and we develop the reward function to include our reward and punishment strategy. Action space sampling can be modified to serve the exploration and exploitation decision at each state in the step function in order to reach the vantage point.

3.4 Web Server

The implemented user interface is a web-based application that doesn't require any prior installation as it can be run through using an internet browser from any mobile device. However, the official ARFreeFlight application that was created to control the AR.Drone quadcopter needs to be installed from the Apple Store or Google Play. A web server is created using Apache 2.0 [13] server on the same machine station that is connected to the AR.Drone via WiFi through an additional USB wireless adapter. Apache compiles SUI application after receiving navigation requests from the connected client and flies the quadcopter. At the same time, it sends back the camera streaming with the result of applying the appropriate processing as output to be viewed on the SUI canvas panel. SUI is created and published over the secured HTTPS protocol and any authorized mobile device is able to control the AR.Drone by accessing the application through the URL in the web browser. The video streaming transfers over HTTP protocol, whereas the commands transfer between the client and the AR.Drone ROS-dependent package through Websocket protocol with the specified port. More information about our SUI can be found in chapter 4.

ROS web tools [5] provide the roslibjs library which is used to transfer commands between ROS and the client through SUI over the Websocket protocol. The rosbridge package works as a Websocket server that holds ROS messages, services, and any other essential functionality transformation sent and received by the publishers and subscriber. Our platform architecture with all the nodes and communications between the multiple software components implemented with different frameworks are illustrated in Figure 3.7.

3.5 Hardware Server

The AR.Drone has a limited processing capability, and does not have sufficient memory to handle the random batches which are needed for learning our deep neural network (DNN). Therefore, an external machine is used, and it is connected to the AR.Drone through WiFi for data manipulation and transfer. The machine has a core i7 CPU and an NVIDIA GeForce GTX Titan X graphics card in order to handle building a DNN specially for learning the object detector and reducing the time required to learn the network parameters for the optimal policy. *SelfieDroneStick* agent uses significantly less memory as its architecture is created with a smaller network, less parameters, and without the need for CNN. Also, the designed state space is based on the analyzed data from both of the visual and odometry observations to eliminate processing unaffected pixels and ease the model transformation after training from the simulated environment to the real-world drone hardware.

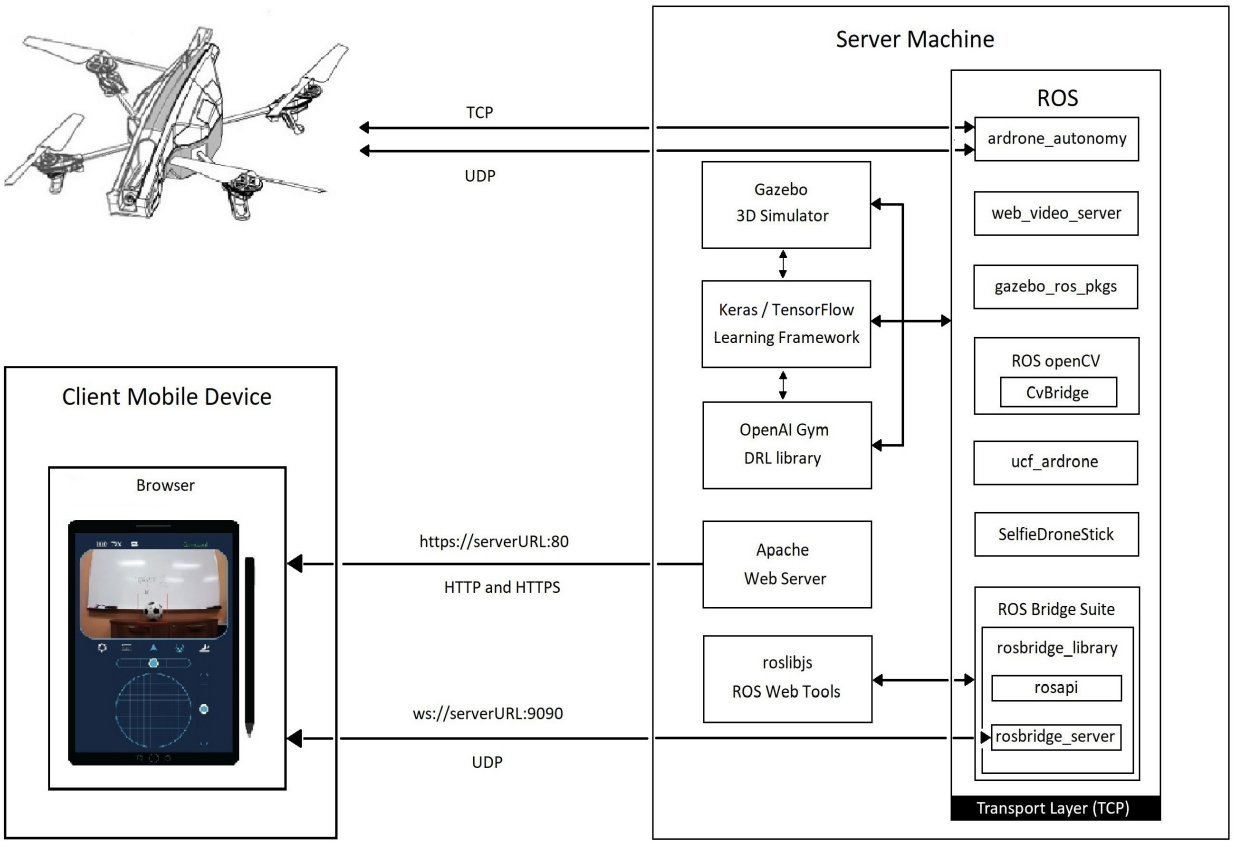


Figure 3.7: The designated platform connects different frameworks through ROS messaging server and the web server. ROS nodes communicate with each other on the transport TCP layer and with nodes from different framework on the Websocket protocol. The connection to the AR.Drone is established on UDP network layer. HTTPS protocol is used to ensure a secure connection with the SUI through the web server. Any smart mobile device with IMU sensors and camera can be used to establish a connection with the web server via WiFi and with the AR.Drone quadcopter as well.

CHAPTER 4: INTELLIGENTLY ASSISTING HUMAN-GUIDED QUADCOPTER PHOTOGRAPHY[1]

Under the supervision of a careful, experienced pilot, quadcopters can be used to capture amazing photographs; however, the typical user's experience is marred by crashes and poor quality photos.[2] The question remains—-how to provide a rewarding human-robot interaction for inexperienced users working with hobbyist quadcopters? This dissertation proposes a sketch-based interface which is designed to hide certain degrees of freedom from user control and prevent crashes by monitoring the scale of the targeted object (Figure 4.1).

The interface offers the following functionality: 1) three canvases for manual navigation that capture user sketches and translate them into control commands in eight directions; 2) a canvas for displaying the real-time image from the quadcopter frontal camera that can be used to select an object of interest for the quadcopter to track autonomously; 3) a dataset collection mode in which the quadcopter autonomously collects images suitable for a variety of applications, including infrastructure inspection, 3D reconstruction, and training machine learning classifiers. The system is implemented as a web-based application that can be run on a variety of platforms with no installation required. It can be accessed by multiple clients, allowing several users to cooperatively direct the quadcopter. Rapid and reliable object tracking is achieved through the use of an adaptive correlation filter (MOSSE, Minimum Output Sum of Squared Error [15]); previous systems have relied on color-based tracking strategies [58]. This dissertation compares the performance of our interface vs. two commercial drone control systems. Based on the participants' performance on indoor image collection tasks, we believe that our system improves on commercial options and

[1]Related publication: S. Alabachi and G. Sukthankar. Intelligently assisting human-guided quadcopter photography. In Proceedings of Florida Artificial Intelligence Research Society , Melbourne, FL, May 2018.

[2]See the NYTimes article "Santa Delivered the Drone. But Not the Safety and Skill to Fly Them.".

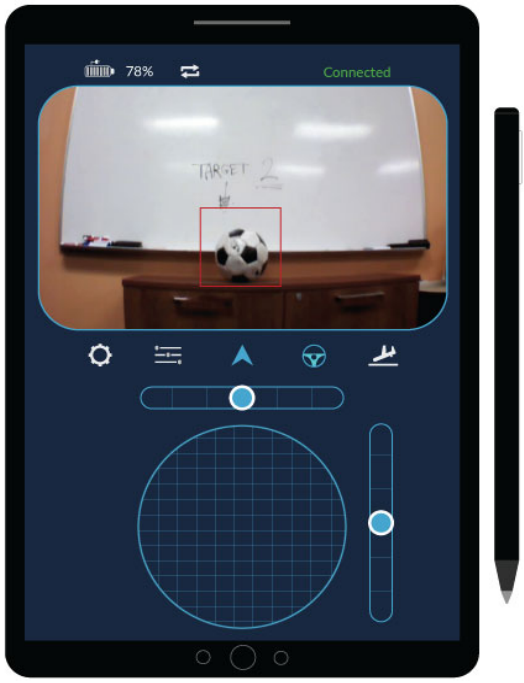


Figure 4.1: The upper part of the interface broadcasts the image stream from the quadcopter frontal camera. The user can select a target by drawing a bounding box over the camera view. After selecting the required object and starting the semi-autonomous mode, the upper part switch to stream the processing frames and shows the auto-selected part to the user. Switching between the manual pilot and the human-guided pilot can be done at any time to and the object selection as well. The human can directly control the quadcopter by sketching on the lower control panel.

provides precise control at low cost without additional hardware extensions.

4.1 Proposed Method

The system architecture is shown in Figure 4.2. The back end was constructed on top of the Robot Operating System (ROS) which can handle communication between several entities without experiencing significant latency. We created a web server using ROS Web Tools [10] and assigned a specific HTTP port to emit ROS video streaming messages, while using network web socket tools to communicate to our user interface. Our experiments were performed on the commercially

available Parrot Augmented Reality (AR) Drone Version 2 for the reasons that have been explained in chapter 3.

4.1.1 Sketch-based User Interface

For the front end part of our system, we designed a web-based interface, which can be simultaneously accessed by multiple users, using a variety of mobile devices. The AR.Drone utilizes UDP with a specified port to communicate with the ground analysis and command unit (back-end), and HTTP is used for data transmission between the front-end and the ground (back-end) unit. There is a dedicated canvas showing the view from the quadcopter front camera, along with three areas that control translation, yaw, and altitude. To enter autonomous object tracking mode, the user can circle a region in the front view canvas.

The user can assert direct control by sketching in the lower panel; there are separate controls for stop, go, takeoff, and landing. The sketches are then translated into linear and angular velocities in the quadcopter coordinate system and normalized by the total length of the corresponding canvas. as follows:

$$z_{linear} = (max_{alt} - min_{alt})/len_{alt} \tag{4.1}$$

$$z_{angular} = (max_{yaw} - min_{yaw})/len_{yaw} \tag{4.2}$$

$$x_{linear} = (max_{hor} - min_{hor})/len_{hor} \tag{4.3}$$

$$y_{linear} = (max_{hor} - min_{hor})/len_{hor} \tag{4.4}$$

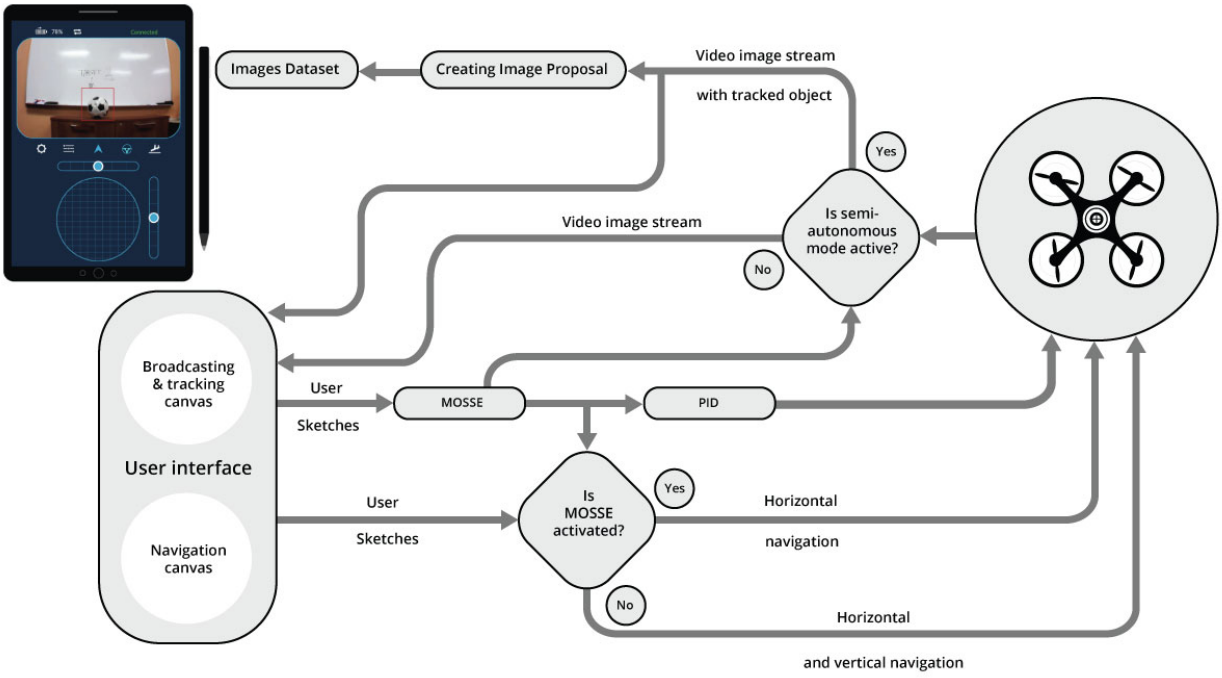


Figure 4.2: The user interface accepts two categories of user sketches: 1) navigation strokes from the three designated canvases which specify the direction and velocity of the quadcopter and 2) the boundary strokes on the broadcasting canvas that enclose the area of interest. During semi-autonomous operation, the MOSSE adaptive correlation filter outputs the object centroid point and the corresponding bounding box. The broadcasting canvas receives the raw image with embedded tracking results at each time interval t. If the object is tracked successfully, the navigation agent locks the yaw angle and altitude of the quadcopter and calculates the measured centroid error to transmit to the PD controller.

where yaw = yaw canvas, hor = horizontal canvas, alt = altitude canvas, and len = total length of the specified canvas in the user interface.

4.1.2 Autonomous Object Photography Mode

For the vision system, we evaluated several object detection and tracking approaches before deciding to use an adaptive correlation filter (MOSSE) to track the region enclosed by the user on the front-view canvas. MOSSE [15] employs convolution to perform the tracking, after creating an

appearance model with adaptive correlation filters. The simplicity of the procedure allows MOSSE to track objects in video captured at high frame rates (> 600 frames per second (fps)). The appearance model is trained in the Fourier domain using a set of random affine transformations, and the aim is to minimize the sum squared error between the desired and actual convolution outputs. During the tracking process, three ROS messages are created for each t period: 1) the centroid point of the tracked object, 2) a tuple-type message for streaming the bounding box coordinates, and 3) an image-type message containing both the front camera image and the bounding box to be viewed on the user interface. x_{min}, y_{min}, x_{max}, and y_{max} are extracted from the circle stroke, and that region of the image is used to initialize the adaptive correlation filter.

After the initial bounding box is drawn, the quadcopter starts flying autonomously, and the system enters a visual dataset collection mode, acquiring data at a rate of 1 fps. The quadcopter modifies its yaw angle and altitude to track the object designated by the user. The x-axis error between the object centroid and canvas center is used to estimate the orientation angle, and the y-axis error is used to estimate the quadcopter's altitude.

$$\text{error}_x = (x_{\text{centroid}} - x_{\text{center}})/x_{\text{max}} \tag{4.5}$$

$$\text{error}_y = (y_{\text{centroid}} - y_{\text{center}})/y_{\text{max}} \tag{4.6}$$

The errors are transmitted to a PD (proportional-derivative) controller with gains K_p and K_d set to 0.25. The quadcopter uses its inertial sensors to monitor roll Φ, pitch Θ, yaw ψ, rotational speed Ψ and the vertical velocity ζ; controls are issued using a series of ROS Twist commands $u = (\bar{\Phi}, \bar{\Theta}, \bar{\zeta}, \bar{\Psi}) \in [-1, 1]^4$ at a frequency of 100Hz. Our interface is capable of eliminating undesired photos by comparing the correlation percentage to a predefined threshold; as long as this percentage exceeds the specified threshold, the agent continues photographing the tracked object,

else it stops.

4.2 Evaluation

We sought feedback on our user interface design from three groups of users. First, to evaluate the ease of learning our sketch-based control paradigm, an observation study was conducted on a group of elementary/high school lab visitors who were asked to fly the quadcopter to a target and land it. In the second study, the performance of the sketch-based user interface was compared to the performance of joystick control for piloting the drone. In the third study, the autonomous visual data collection was evaluated vs. AR.Free Flight image capture. All experiments were performed in an indoor environment, and users were trained in the usage of each control paradigm for five minutes before commencement of testing. Pre and post questionnaires were administered during the second and third studies. The participants' ratings of the difficulty of aerial control under each control modality (joystick, AR.Free Flight, and our smart user interface (SUI)) is illustrated in Figure 4.3; our interface was rated by ten users as being significantly easier to use ($p < 0.05$ on a single tailed paired t-test). A video demo of our system can be viewed at: `https://youtu.be/ErA2111xjzMl`.

4.2.1 Study 1: Elementary/High School Observation

Our elementary/high school guests included four males and two females between the ages of 12 and 16 years old. Our main goal for this study was to observe how younger users would perform with the sketch-based control. The participants were given five minutes of practice and then were asked to try two flying procedures. The first procedure was to fly the quadcopter in a circuit by sketching strokes on the navigation canvases. In the second procedure, they were asked to select a target by sketching a bounding box, as part of flying the circuit. The quadcopter then tries to center

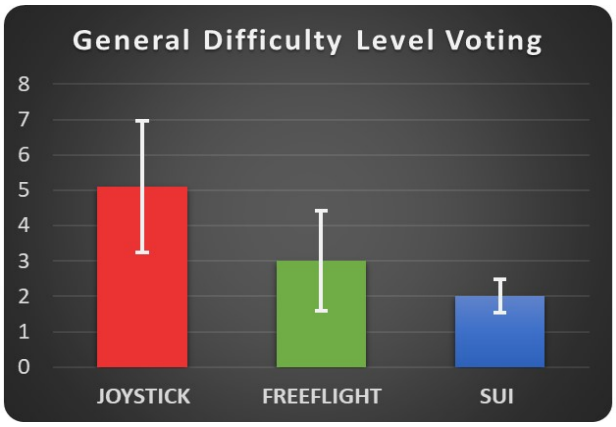


Figure 4.3: Mean and standard deviation obtained from ten participants' rating of the difficulty of each control modality (Study 2 and 3), where 7=most difficult to use and 1=easiest to use. Our interface (SUI) was rated by the users as being significantly easier to use according to a single tailed paired t-test ($p < 0.05$).

Table 4.1: Performance of elementary/high school students using the sketch-based interface. Six volunteers participated, and we tallied how many of the quadcopter control tasks they were able to perform, as well as their interest in the system. In the first condition they were asked to just use the stroke control. Then they were allowed to use a simple bounding box control system, similar in concept to the vision-based tracking but without the filtering or the image capture. Since the children performed well on most of the elements using the bounding box, we decided to incorporate it into our final design.

Study 1		
Commands	Stroke	Bounding Box
Take Off	6/6	6/6
Navigation	4/6	4/6
Reach the Goal	3/6	4/6
Landing	2/6	5/6
Interest	6/6	5/6

the target using a simple visual servoing algorithm.

The participants' performance in achieving the required tasks for both procedures is shown in Table 5.1. The AR.Drone 2.0 elite edition has a maximum speed of about eleven meters per second, which is quite high when navigating in an indoor environment. For more safety, we added an option to our system in which the user can limit the speed by curtailing the length of drawn strokes. This study enabled us to test whether using the bounding box to guide the quadcopter was an intuitive control choice. We determined that the addition of that control option improved navigation, particularly for promoting successful landings.

4.2.2 Study 2: Navigation Control

Our second study focused on evaluating navigation performance with the user interface. Our participants were assigned two objectives to reach with the quadcopter (Table 4.3). The first object was a fire-alarm mounted on the wall, and the second one was a soccer ball placed on a cabinet. They were asked to fly the AR.Drone, face each target while maintaining a safe one meter distance, and return to the start point. For evaluation purposes, we employed a SLAM system [60] to estimate the position of the quadcopter during flight (Figure 4.4).

Users flew the scenario once using the joystick control and the other time using our interface (in randomized order). Two participants rated themselves as expert gamers in the pre-questionnaire. They were able to fly acceptably well using the joystick, but many of the other users either crashed or exceeded the allotted time. However, with our interface, all participants were able to fly the quadcopter without crashing and complete the task in under the three minute time limit (Figure 4.5). The PR (percentage of targets reached) was measured, along the time required, including overtime trials and crashes (Table 4.4). Participants using the interface had higher success rates at reaching the targets, compared to joystick control. Expert users experienced slightly slower

Table 4.2: Participants' Post-Questionnaire (Rating out of 7)

Question	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Do you agree with the idea of controlling the drone with a stylus?	7	6	6	6	6	7	7	4	7	7
How do you feel about the proposed user interface?	7	6	6	6	5	7	7	5	6	6
How would you rate using the assistant agent in directing the drone to track a specific part of the frame in dataset collection?	7	7	6	6	7	7	7	4	7	6
Do you think engaging the assistant agent in such kind of dataset image collection is crucial?	7	6	6	7	6	6	7	1	7	7
Would you recommend having more control such as specifying part of the entire frame while capturing the images?	Yes	Yes	Yes	Maybe	Maybe	Maybe	Yes	Maybe	Yes	Yes
Would you like to have an assistant agent helping you out with capturing images of the selected object automatically?	Yes	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes
Would you like to have an assistant agent helping you out with navigation while capturing the images?	Yes	Yes	Yes	No	Maybe	No	Yes	Maybe	Yes	Yes
Would you use the proposed user interface to collect images for your own project?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes

flight times, however when accounting for unsuccessful trials the overall time required for our user interface was improved.

Table 4.3: Participant information for the second and third study. None of the users had prior experience flying drones. Many of them were intermediate or expert game players. All participants but one felt most comfortable with either console or touch pad game controllers.

Participants Information							
P	PC Game Skill Level			Controller Prior Knowledge			Drone Prior Knowledge
	Beginner	Intermediate	Expert	Console Controller	Touch Pad	Others	
P1		✓		✓			
P2		✓				✓	
P3			✓	✓			
P4			✓	✓			
P5		✓			✓		
P6	✓			✓			
P7	✓				✓		
P8		✓			✓		
P9	✓				✓		
P10		✓			✓		

Table 4.4: Our user interface makes navigation much more reliable for the users. In Study 2, only 40% of the targets were reached (across all users), whereas 100% of the targets were reached by participants employing our user interface.

Study 2 (Navigation Performance)	
User Interface	Targets Reached (%)
Joystick	40%
Interface	100%

4.2.3 Study 3: Visual Dataset Collection

For this study, users were asked to collect an image dataset using our interface vs. capturing images using the AR.FreeFlight piloting application. The Parrot developer community has created versions of the AR.FreeFlight user interface for iOS, Android, and Windows platforms; it is the official form of software control for the AR.Drone quadcopter. The piloting section of the AR.FreeFlight

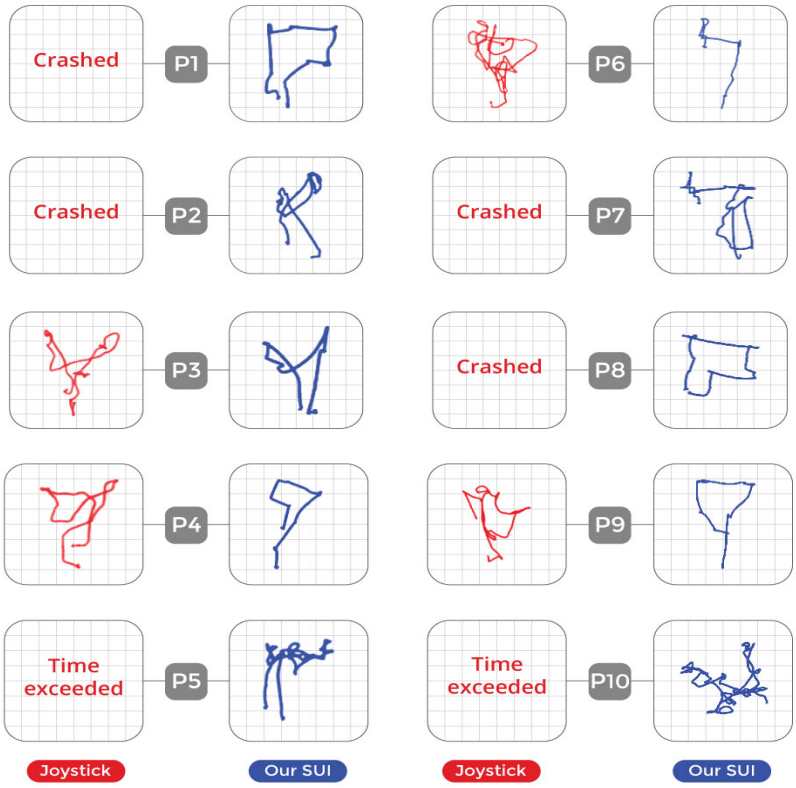


Figure 4.4: In study 2, participants were asked to fly to two target objectives and return to the start point using joystick control and our sketch-based user interface. We exported the flight paths that the quadcopter measured using its SLAM system. An ideal path would be shaped like an isosceles triangle. The red paths are the ones executed under joystick control, and the blue ones were done with our user interface. Paths from participants P1, P2, P5, P7, P8 and P10 were not captured because either they were unable to reach the targets using a joystick within the specified time or crashed the drone three times.

UI has a screen that shows the frontal and downward cameras, along with takeoff/land buttons, photo/video capture buttons, and two joysticks (Figure 4.8). Moving the quadcopter horizontally can be done through using the left joystick or tilting the tablet/phone. The autonomous image capturing option offered by our system frees the participants from doing it manually. This option along with the target tracking feature ease the operation of image capturing. The users were able to rapidly collect more images using our interface (significantly more according to a paired single

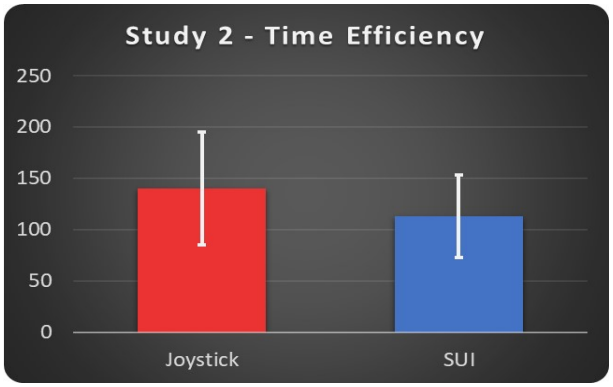


Figure 4.5: Average time required (seconds) for participants to complete both navigation tasks in study 2. Participants that crashed the quadcopter were assumed to have taken the maximum required time (180 seconds). Despite a few fast joystick runs by the expert users, our user interface led to a faster average completion time ($p < 0.05$ according to a paired single-tailed t-test)

.

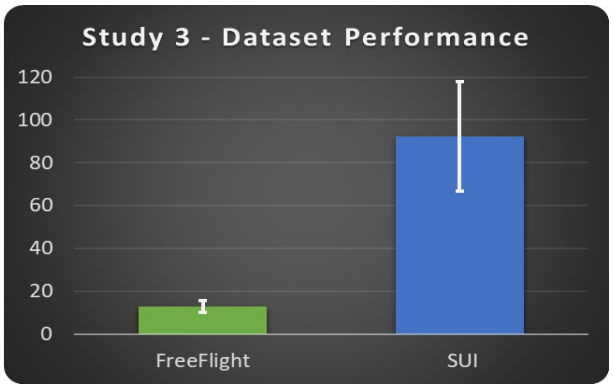


Figure 4.6: Number of images collected by users in study 3. Our interface supports more prolific image collections which are useful for training machine learning classifiers.

tailed t-test at the $p < 0.05$ level) as illustrated in Figure 4.6. From Figure 4.7, we can see that when participants used our interface they were able to acquire more diverse image views. This variety in image characteristics is particularly valuable for training machine learning classifiers.

After the experiments, we administered a post-questionnaire to the participants seeking their opinion and likeliness about using our user interface in achieving the requested task. Key questions

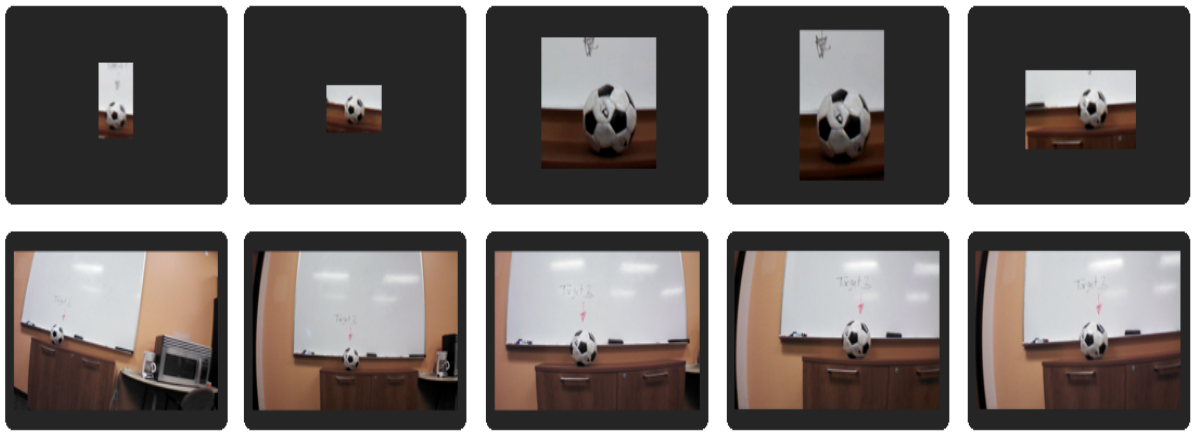


Figure 4.7: Examples of images captured using the visual dataset collection mode. The top row shows images captured with our user interface; the bottom row shows images captured with AR.FreeFlight.

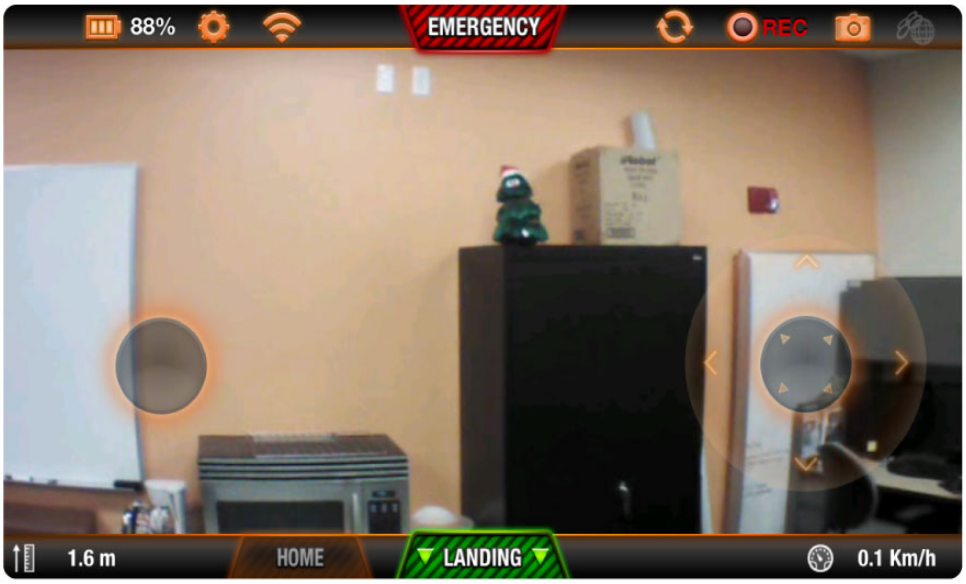


Figure 4.8: Piloting the quadcopter using AR.FreeFlight user interface. Mobile device IMU and touch-based gestures are used to control the quadcopter.

Table 4.5: Participants' Post-Questionnaire (Opinions and Comments)

Question	P1	P2	P3	P4
What would you recommend adding or changing to enhance your navigation experience?	More control over the speed of the drone	Make better hovering action, sticker indicators on the quadcopter	Stop the drone immediately	It would be cool if there was "pinching" support, so you pinch with a thumb and another finer to control the yaw
What would you recommend adding/changing to enhance your dataset collection experience?	More control by the ardrone when capturing the images to have a more stable, clear images	An option to lock on the target would be helpful	Make the control easier	If possible, a square "capture box" that centers around the object
What would you recommend adding/changing to enhance your navigation experience?	Nothing	We can use the rate of pen movement to control the speed of the drone instead of the length of the drawing	Nothing	The circle worked well, an alternative could be just pressing a dot to focus on one point on an object
What would you recommend adding/changing to enhance your dataset collection experience?	To show the number if images taken on the main screen		I liked it, nothing in mind	
For which future purposes do you believe that this user interface would be most beneficial?	Could be used to capture clear images on large fields while remaining on the same spot		Capturing people faces	Remote viewing screen and capturing, especially ones in nests

included:

1. Would you like to have an assistant agent helping you out with capturing images of the selected object automatically?

2. Would you like to have an assistant agent helping you out with navigation while capturing the images?
3. Would you use the proposed user interface to collect images for your own project?

The majority of the participants responded positively to all these questions, indicating a high level of satisfaction with the concept of the intelligent user interface. The rating of 10 participants about the user interface out of 7 is shown in Table 4.2. Examples of their opinions about the overall experience are shown in Table 4.5.

4.2.4 Summary

We introduced a smart user interface (SUI) that uses sketch-based control to facilitate drone navigation and visual dataset collection tasks. Our implementation is platform-independent and can be accessed from any mobile device without prior installation. Our experiments demonstrate that our interface outperforms standard commercial solutions, such as joystick and AR.Free Flight. A key contribution is the use of adaptive correlation filters for visual tracking of objects in the semi-autonomous target selection mode. The MOSSE filter is robust against many appearance changes and capable of executing at high frame rates. We tested our platform in three different scenarios with participants from different age groups. The participants were able to robustly execute navigation patterns and collect visual datasets without crashing and expressed satisfaction with the user experience.

CHAPTER 5: CUSTOMIZING OBJECT DETECTORS FOR INDOOR ROBOTS[1]

Indoor mobile robots operate in spaces that are not as rigorously controlled as manufacturing areas, nor as rich with diversity as outdoor scenes. The open source release of pre-trained object detection models such as the TensorFlow Object Detection API [50] has been a boon to robotics, but in indoor spaces, many objects, particularly small ones, are omitted from the common object datasets. This is a hindrance for creating indoor robots that can be tasked to find or manipulate objects on tables, walls, and desks. Our aim is to develop a system that can be used to rapidly create customized detectors for vision-based robots that require real-time object detection. This is related to the challenge of using deep learning to perform visual SLAM [120] but with the objective of tasking the robot to use the objects rather than learning landmarks for visual navigation. This dissertation addresses the following challenges:

1. Collecting and annotating images of novel relevant objects with minimal human effort.
2. Developing a CNN architecture that trains with limited data and performs real-time inference on videos.

Our proposed system is illustrated in Figure 5.1. It includes an interactive user interface that enables the user to task a quadcopter to autonomously collect images of target objects from multiple viewpoints and label them without additional manual effort. Quadcopters as imaging platforms have become ubiquitous in recent years in a wide variety of applications including surveillance [12], real-estate photography, agricultural and industrial inspection [101].

To address the second challenge, we introduce a new architecture, DUNet (Densely Upscaled

[1]S. Alabachi, G. Sukthankar, and R. Sukthankar. Customizing object detectors for indoor robots. IEEE International Conference on Robotics and Automation (ICRA) , 2019.

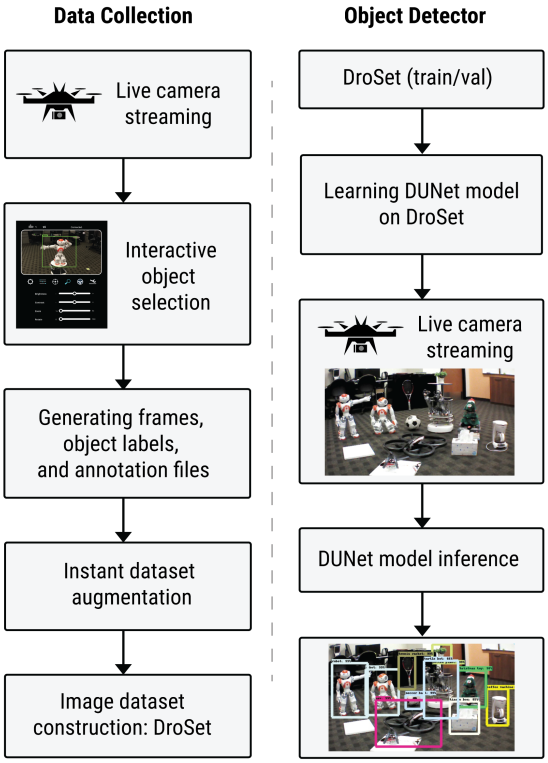


Figure 5.1: Our system consists of two parts: 1) a semi-autonomous data collection system and 2) our neural network architecture for rapidly training custom object detectors. The user teleoperates the quadcopter toward the object using an interactive user interface. After the user selects the object, the quadcopter autonomously captures multiple views of the object, which are then augmented with synthetically filtered images. This dataset (DroSet) was then used to train DUNet (Dense Upscaled Network). At the conclusion of the procedure, the quadcopter can fly autonomously around the environment rapidly and reliably detecting all objects initially specified by the user.

Network), that is inspired by the DenseNet [49] image classifier, the feature pyramid network (FPN) [69], and the meta-architecture of the SSD [73] object detector. By combining dense layers and upscaling, DUNet can reliably detect small objects and requires fewer classification layers to achieve the desired speed-quality balance.

Our data collection platform was used to collect a dataset (DroSet) of ten real-world objects along with labeling and bounding box annotations. It includes both images captured at different view-

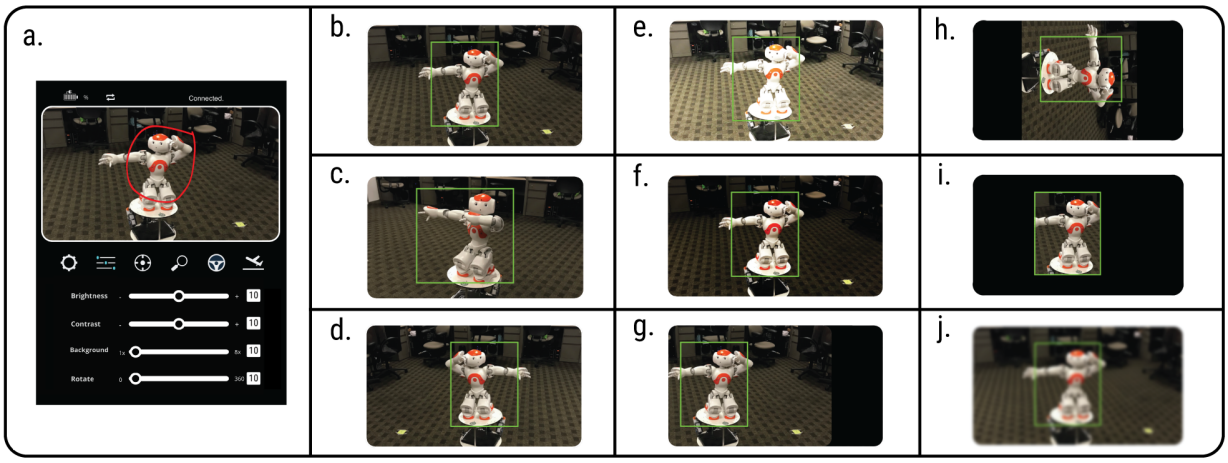


Figure 5.2: Real-time Data Generator (Stage 1). SUI includes many types of data augmentation. Some filters from the list include: a. Object selection during the flight. b. Autonomous labeling and bounding box initialization. c. Ability to create different offsets. d. Flipping e. Brightness. f. Contrast. g. translation h. Rotation and flipping. i. Changing background. and j. Blurring

points and ranges, along with augmented data created by applying filters to create contrast, background, and brightness variations as shown in Figure 5.2. A standard approach would be to use DroSet to fine tune an existing network trained on a large dataset such as PascalVOC [34]. Fine-tuning can be used to reduce the training time and improve convergence; if the new objects do not share sufficient feature representations with the original dataset then fine tuning performs poorly. In contrast, the dense layers of DUNet promote convergence on small customized datasets. Both our DUNet framework[2] and dataset are publicly available.

Our experimental results demonstrate that the DUNet architecture can be trained from scratch on a small dataset, achieves higher accuracy on small-sized objects and achieves frame-rate object detection on image streams. DUNet is also practical as a generic object detector, achieving competitive performance on standardized object detection datasets as state of the art models.

[2]Download DUNet from:`https://github.com/cyberphantom/Customizing-Object-\Detectors-for-Indoor-Robots`

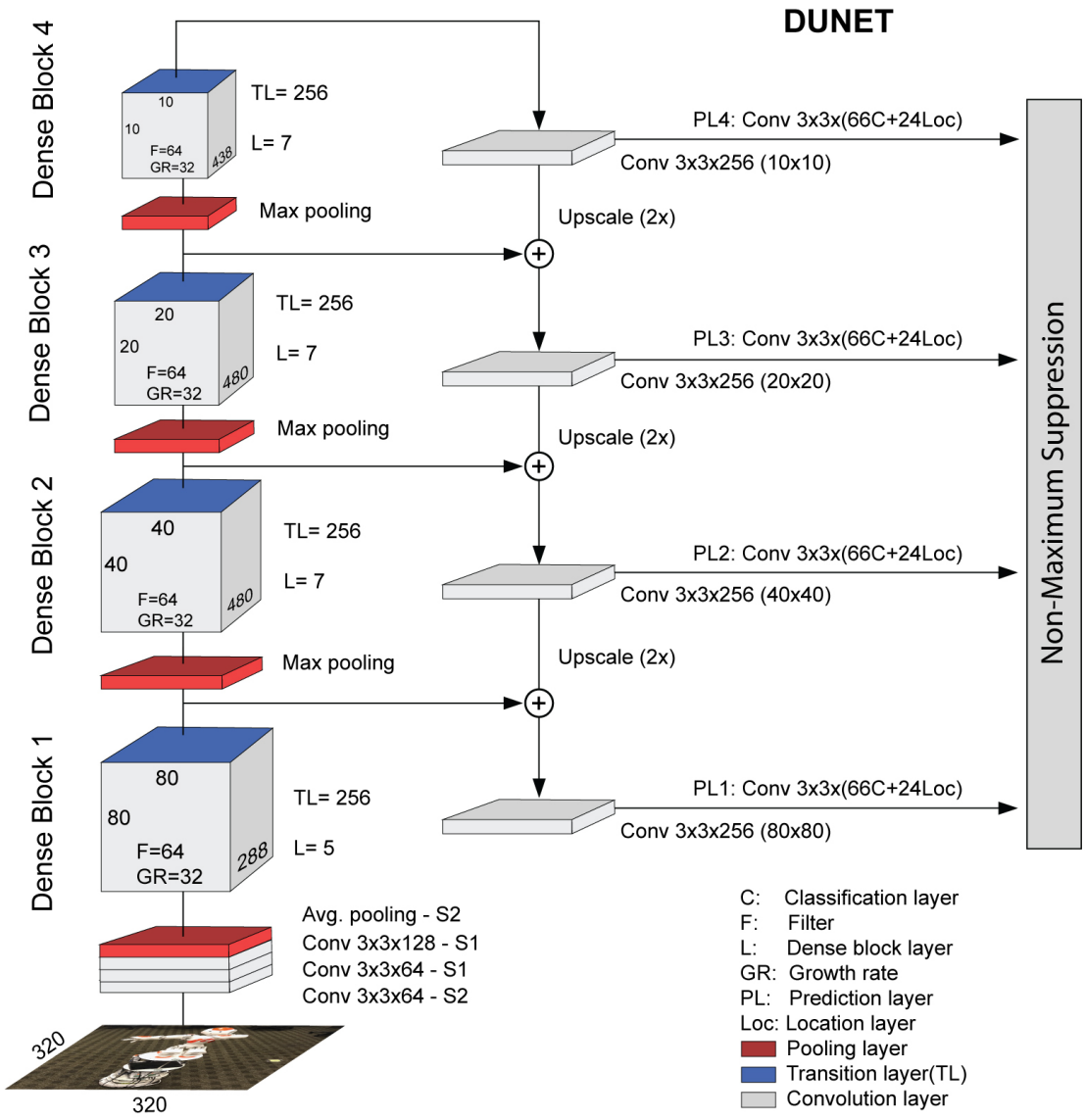


Figure 5.3: DUNet architecture

5.1 Dense Upscaled Network (DUNet)

A typical modern Deep CNN-based object detection system consists of a feature extraction stage combined with an approach for generating bounding box proposals, followed by appropriate clas-

sification and regression layers and strategies for non-maxima suppression. For example, an object detector built using the SSD [73] meta-architecture may employ VGG16 [108] as a feature extraction network, followed by six convolutional classification layers, with four localization points for regressing the ground truth proposals for each class.

The meta-architecture for our model, Dense Upscaled Network (DUNet) is summarized in Figure 5.3. At a high level, DUNet consists of a sequence of dense blocks that process the input image at different scales, connected to a sequence of prediction layers, each of which independently generate detection results. The two sequences are connected both laterally (at appropriate scales) and vertically (through max pooling and upscaling). We detail each of these aspects below.

5.1.1 Feature Extraction

As discussed above, many object detectors employ a base network for feature extraction; for instance, SSD uses the VGG16 network (pre-trained on ImageNet) for this purpose. In DUNet, we eliminate the use of VGG16 and instead start with a fully convolutional "initial layer" sequence followed by average pooling that serves as our feature extractor; this is not pre-trained using ImageNet but is simply randomly initialized and jointly trained from scratch. We employ batch normalization [52] before every convolution in DUNet.

DUNet then processes these initial features using a bottom-up pathway of four dense blocks, rather than the ResNet architecture employed by SSD. Like ResNet, dense blocks enable us to avoid the problem of vanishing gradients and we are able to train these for a customized detector from scratch on a relatively small dataset.

The first dense block consists of five layers while the remaining dense blocks use seven layers each with 64 filters and a growth rate of 32. Each layer of a dense block includes normalization, ReLU and convolution layers, and each layer's input consists of the concatenated outputs of every

feature-map from each of the preceding layers as illustrated in Figure 5.4. The pooling layers are removed from the TLs and max pooling layers with stride = 2 are placed after each of the first three dense blocks to reduce the spatial dimension by half.

The top-down pathway (inspired by feature pyramid networks [69] and top-down modulation [106]) consists of prediction layers interspersed with $2\times$ upscaling operations. The intuition is that this configuration improves detection of small objects based on their context because each of the prediction layers can exploit both high-resolution features and top-down context. Additionally, the lateral connections serve as skip connections that create short pathways from input to output. Using dense blocks inside FPN and adding their feature maps with the upsampled ones from the top-down pathway of FPN for proposing the initial boxes coordinates enhances feature extraction as each layer in the dense block contains all the previous information as a concatenation layer. The primary difference is that in dense blocks, the outputs are concatenated instead of summed, so that each dense block has $L(L+1)/2$ direct connections.

In FPN, a 1×1 convolution must be applied to the feature maps of the bottom-up pathway in order to be added element-wise to the feature maps of the top-down pathway. However, each TL is a 1×1 convolution layer, so we don't need to apply the FPN $4\times(1\times1)$ convolutions again to reduce the computational cost. That is why we removed the pooling layer from each TL in the dense blocks of the bottom-up pathway. This is one of the strengths of DUNet: sharing computational cost between the convolutions. Detection performance on small objects is improved through the summation of the concatenated feature maps with the upsampled high semantic layer feature maps. Batch normalization is used prior to convolutions in all network layers even in our reduced SSD meta-architecture filter layers.

To the best of our knowledge, DUNet is the first architecture to exploit both DenseNet-style concatenation (via dense blocks) in the bottom-up pathway and ResNet-style summation (via the upscaling) in the top-down pathway.

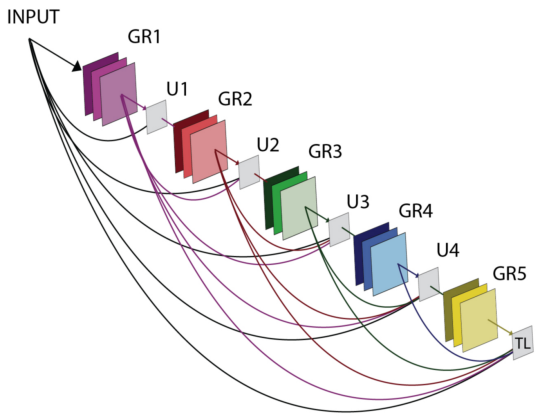


Figure 5.4: A five-layer Dense block. Each layer has its own number of feature maps based on the growth rate (GR). The output of each layer concatenated with all previous feature maps to be the input for the next transition layer (TL).

5.1.2 Meta Architecture Design Choices

As described above, many aspects of DUNet's design, such as the use of top-down pathway, are motivated by our desire to improve performance on objects that occupy only a small portion of the image. Reliably finding such "small objects" is critical for robotics tasks, particularly when navigating a robot towards a semantic landmark that is farther away (e.g., "go to the fire alarm").

A straightforward approach towards this goal would have been to add more classification layers to SSD or consider more aspect ratios/scales. However, such an approach would come at significant computational cost. Instead, in DUNet, we are able to *reduce* the number of classification layers from six (SSD) to four, while achieving better performance on small object detection in streaming video.

Inspite of achieving real time object detection, SSD [73] still tends to get confused between classes of relatively small sizes. Attempts to solve this confusion by adding more aspect rations or scales for the anchor boxes may improve mAP slightly but results in more computational time which

significantly reduces the model speed. DUNet solves this confusion and increases the ability to detect small objects with fewer prediction layers in its meta-architecture. We use the same concept of SSD for the classification layers, but we were able to reduce the number of classification layers from six to four and achieve better performance at detecting small-sizes objects in real-time video streaming. We eliminate the use of two classification layers of SSD by a higher 80×80 resolution layer. SSD classification layers starts from 38×38 feature map, leading to reduced performance on small objects. If we want to increase one layer with the same SSD meta-architecture, we will lose the real-time detection speed. Reducing the number of classifications and using Dense blocks along with FPN enable us to reduces the number of classification from six to four with a significant increase in the detection accuracy of small objects.

The trade-off relationship between detection speed and accuracy limits the input size. For instance, in SSD [73], the authors demonstrate the difference between two implemented versions of SSD network, SSD300 with 300×300 input size resolution and SSD512 with 512×512. On PascalVOC2007 test, SSD300 has mAP=74.3 and 59 fps, whereas SSD512 has mAP=76.8 and 22 fps, so SSD512 is only 2.5% better accuracy than SSD300 sacrificing more than 62% of SSD300 speed. Based on this observation, in DUNet we chose 320×320 as the input size resolution. Our experiments show that this input resolution achieves a good balance between accuracy and speed, given our meta-architecture. As detailed below, DUNet without any pre-training and with random initialization outperforms the most recent state-of-the-art object detection models trained on large datasets like MSCOCO and ImageNet on object detection in streaming video.

5.2 Semi-Autonomous Visual Data Collection

As discussed in the introduction, object detection for indoor robotics imposes different challenges than those encountered in object detection for web imagery, such as the requirements for near real-

time processing of input video streams, the importance of reliably detecting small-sized objects and the ability to customize the detector for new object classes from limited labeled data. Fortunately, we can also benefit from several features of indoor environments, such as limited variability in terms of lighting, viewpoint, range and background conditions. Here, we present an approach for acquiring training data with minimal human labeling as well as a public dataset (DroSet) for evaluating object detectors on streaming video in such environments.

5.2.1 Background

Datasets for deep learning object detection networks require that each object in an image be labeled with a single ID (or a class) and localized via one of the bounding box coordinates standard assignment methods (minmax, corners, or centroids). Each image is then processed by a real-time augmentation process that generates synthetic variations using geometric and photometric transforms. The construction of DroSet is is described in this section.

Collecting labeled datasets for object detection (e.g., PASCAL VOC [34], COCO [70]) is significantly more onerous than labeling datasets for whole-image classification (e.g., ImageNet [30]). This is because each instance of a relevant object in the image must be localized using a bounding box, which can take several seconds per instance even for an expert annotator.

When the input images consist of a video stream, manually labeling each frame becomes impractical and it is important to consider semi-automated schemes for labeling that exploit temporal consistency.

5.2.2 *Interactive Data Collection*

We collect training and evaluation data using an indoor drone and a semi-autonomous user interface (SUI) [8]. The user interactively selects objects of interest and the tracking agent controls the drone to collect a stream of images capturing the object from multiple viewpoints, by tracking the object while flying in a variety of patterns. The system minimizes annotator effort by exploiting temporal consistency since the tracker automatically propagates the bounding box around the object from frame to frame. This data is then used to train DUNet and enables repeatable object detection experiments.

By initiating the semi-autonomous mode in our drone interface (SUI) [8], a tracking agent begins to track the object that has been requested by the user in the scene. The SUI collects images of the object from multiple viewpoints while its semi-autonomous navigation and records the bounding box annotations in the successive frames that can be used as ground truth information for training a deep CNN object detection model. As it flies around the scene, the drone attempts to collect images that center the target object in the area of interest designated during initialization. The resulting automatically-generated images are thus similar in composition to the centered images of objects in ImageNet [30]. The canvas tool used for object selection is shown in Figure 5.2 (a), and examples of the generated bounding boxes by the tracker agent with different visual effects are illustrated in Figure 5.2 (b-j).

5.2.3 *Live Data Augmentation*

Synthetic image augmentation [92] is performed on a captured image using a series of 2D geometric transforms (e.g., rotation, translations) and induced photometric variations (e.g., brightness, contrast and color shifts). The resulting set of images for each object instance are much richer than those that would be typically obtained from the internet since they include variations in appearance

induced by viewpoint changes as well as specular reflections from changing lighting (relative to camera).

Given that there is significant redundancy across consecutive images in the image stream, we choose a slightly different data augmentation strategy than is commonly employed on standard image datasets. Rather than applying all of the augmentation filters on each image, our system captures a fresh frame before applying each filter (to further introduce slight variations). Thus, successively captured frames are processed by each of the transformations. we acquire images at a high frame rate and only apply a single filter to any given image (to reduce the number of near duplicates). In each second, our system generates k frames where $k = f + g$. f is the desired capture rate, and g is the number of synthetic frames generated by the live data augmentation, based on the current number of active transformation filters.

We include all of the common geometric and photometric transformations, such as brightness, contrast, rotation, flipping, shadow, background, and color shift. The user can interactively add or remove filters as desired during the capture process, as well as selecting the rate at which each filter is applied (e.g., if more rotations vs. contrast changes are desired). Since each filter is applied to a freshly captured frame, the data generator generates fewer "near-duplicate" instances in its dataset than traditional data augmentation schemes that are forced to apply all filters on each original image. We specify 500 ms as a minimum threshold between consecutively captured images in order to allow sufficient time for the drone to change its position (gaining more variations in depth and angle view point).

The drone that we use in our experiments streams 720p resolution video at a rate of 30 fps. However, we need a higher input frame rate to achieve a real-time rebroadcasting frame rate after detection, so we used 640x360 pixel resolution broadcasting frame option that increase the transmission rate to $\sim$30.6 fps.

5.2.4 DroSet: A Dataset of Indoor Objects

Following the procedure described above, we captured footage of 10 object categories in indoor environments and organized it into training (75%), validation (15%) and test (10%) sets. This dataset, termed DroSet, has been released publicly[3] to enable other researchers to evaluate their object detection algorithms under our conditions.

DroSet consists of image streams for the following ten categories of indoor objects: christmas toy, coffee machine, potted plant, tissue box, robot, soccer ball, turtle bot, UAV, fire alarm, and tennis racket. By design, three of these categories (e.g., potted plant) overlap with categories in COCO, while the others are new. Some of the object categories exhibit little visual variation (e.g., fire alarm), while others (e.g., UAV) contain objects with very different appearances. Our choice of categories should enable researchers to better evaluate the extent to which transfer learning generalizes from standard datasets to our dataset for both the overlapping and new categories.

Table 5.1: mAP for SSD300 fine-tuned from a pre-trained VGG16 reduced model on ImageNet versus our DUNet320 trained from scratch with random normal distribution initialization on DroSet dataset.

Model	mAP	C-Toy	CofMach	PotPlant	Tissue-Box	Robot	SoccBall	Turtle-Bot	UAV	Fire-Alarm	T-Rckt
SSD300	0.83	0.641	0.807	0.922	0.63	0.977	0.567	0.937	0.979	0.935	0.907
DUNet	0.833	0.658	0.761	0.91	0.781	0.972	0.538	0.893	0.975	0.9	0.94

[3]The DroSet dataset is available at `https://goo.gl/xE6Jkr`

5.3 Experimental Results

For indoor robotics applications, it is important that proposed methods find a good balance between processing speed and detection quality. Thus, our primary experimental scenario (Sec. 5.3.1) evaluates methods on a 30fps input stream of frames. However, it is valuable to confirm that our proposed model is competitive on traditional object detection metrics, so we also include a direct comparison of DUNet against SSD on a standard dataset (Sec. 5.3.2).

DUNet is implemented using Keras with the TensorFlow [7] back-end. We use the TensorFlow Object Detection API [50] implementations for all of the baseline models, such as SSD-300. All of the experiments were conducted on a machine with an NVIDIA GeForce GTX Titan X graphics card.

5.3.1 Scenario I: Evaluation on Real-Time Robot Input Stream

In this scenario, we propose an evaluation procedure that can test several models on the same successive frames and communication environment. We use the Robotics Operating System (ROS) [94] to record frame streams captured by the quadcopter camera. This enables us to create repeatable playback environments for testing the different models under realistic robot conditions. For this scenario, we create bag test files for each of the ten DroSet categories (where exactly one instance of the given object appears in each frame) to enable computation of per-class results. These are available in the public DroSet release. We want to evaluate several detection models mimicking the live broadcasting of video streaming from the drone's frontal camera as a publisher topic message. Therefore, we have created bag files that record the video broadcasting as a publisher topic to be used by all the selected models as listener topics. Ten videos have been recorded as bag files for all DroSet objects individually. Each video consists of the required object. If the video includes an additional object occasionally, we didn't account for its existence in calculating TP, FN, and FP

scores. The experimental results of the tested models with DroSet as well as our DUNet model scores for TP in the selected 31 seconds evaluation period from each bag file are illustrated in Figure 5.7. DUNet frame rate achieves the closest to real-time in rebroadcasting (with 0.98sec to send 30 frames) and its TP rate in the evaluated period shows the highest response among all the other models. FN and FP rates of our DUNet are also in the lowest level as shown in Figure 5.5. We believe that DUNet is the most suitable model to be used for high-perspective detection cases like a drone, particularly with a small training dataset. This observation is made based on the average precision, recall, and accuracy scores in Figure 5.6.

Measuring different detection models on live video streaming seems to be complex as there is no unique metric measurement that we can rely on other than testing all the models on the same video samples and with the same system settings. We have experimented with two different scenarios using our DroSet dataset and DUNet model. All the tests have been performed on a Nvidia GeForce TITAN X which is the fastest graphics card that is possible to be deployed on a drone yet. Our platform is implemented using Keras with Tensorflow backend, and we also fine-tune real-time and near real-time models from the Tensorflow object detection API on DroSet to evaluate our DUNet model on the same small dataset. Our semi-autonomously created dataset, DroSet, contains ten classes: seven classes are totally new and do not exist in MSCOCO or PascalVOC datasets, and the other three are already existing classes in MSCOCO with different images. Having these three classes with new image data is necessary to evaluate the detection performance when performing fine-tuning on the other state-of-the-art object detection models against ours which is learned from random distribution. Some of the DroSet classes include objects with different shapes in the same category. Coffee machine, tennis rackets, and UAV classes contain different objects. For instance, the UAV class has images for a quadcopter and a helicopter.

A comparison between DUNet against a comprehensive array of state-of-the-art models is illustrated in Figure 5.5. The benchmark is based on terms of detection quality (true positive, false

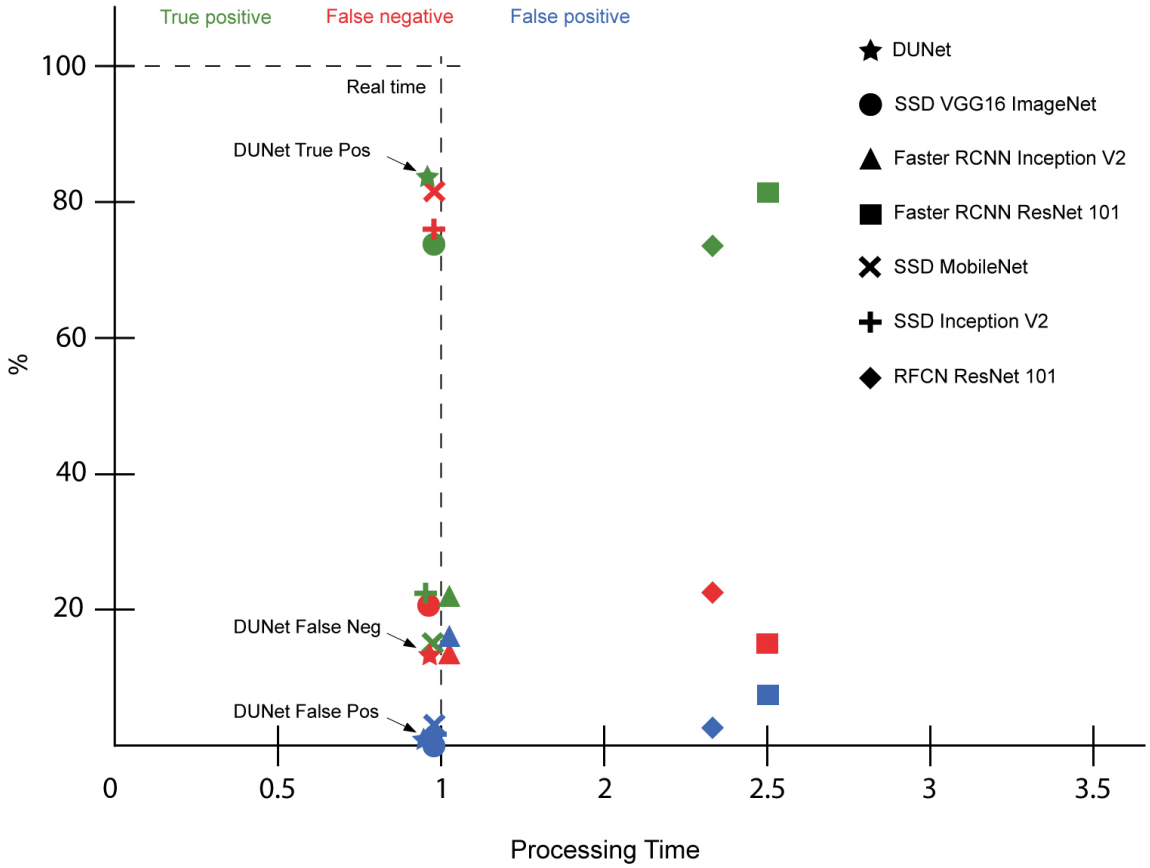


Figure 5.5: Scatterplot of detector quality (TP, FN, FP) on DroSet vs. processing time (normalized to real-time) for each model. DUNet clearly outperforms other models while processing input stream in real time.

negative and false positive rates) and processing time. We observe that DUNet clearly outperforms real-time baselines like SSD VGG16 ImageNet in terms of detection quality, and is 2.5× faster than state-of-the-art models like Faster R-CNN + ResNet 101, which are unable to keep up with the input stream. These results on DroSet are consistent with the speed/accuracy experiments reported on standard datasets [50]. The average precision, recall and accuracy on DroSet for all of the models are illustrated in Figure 5.6. Overall, DUNet is the clear winner.

A more detailed breakdown of each model (shown as a column) on the subset of sequences fea-

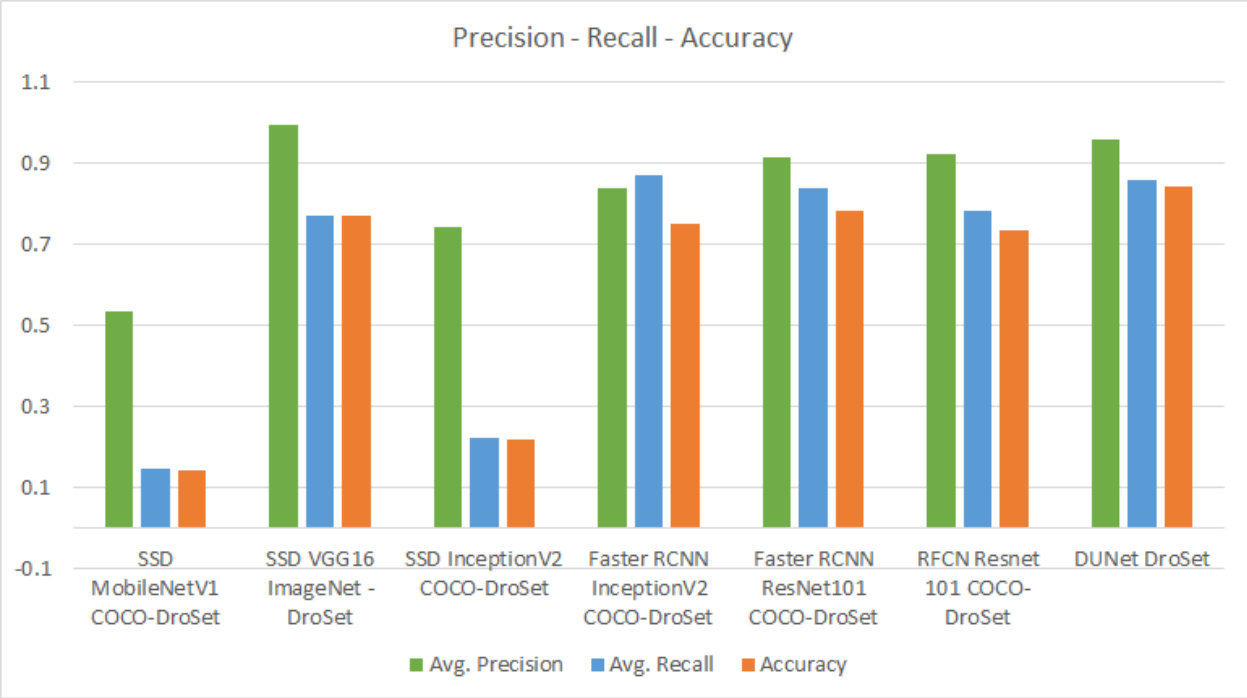


Figure 5.6: Average precision, recall, and accuracy on DroSet for the fine-tuned, pre-trained baseline models vs. DuNet (trained from scratch).

turing a given DroSet category is presented in Figure 5.7. The translucent bars correspond to the number of frames processed by each model in steady state. Most models either fail to process sufficient frames or exhibit low detection rate. We also see that fine-tuning standard pre-trained detectors on DroNet can vary widely: e.g., SSD MobileNet V1 pre-trained on COCO and fine-tuned on DroSet does well on tennis racket but terribly on christmas toy. Interestingly, there is not a clear correlation between the domain transfer performance for such baseline models and categories that overlap with COCO vs. new categories. DUNet (despite being trained from scratch) wins on both metrics on almost all of the classes.

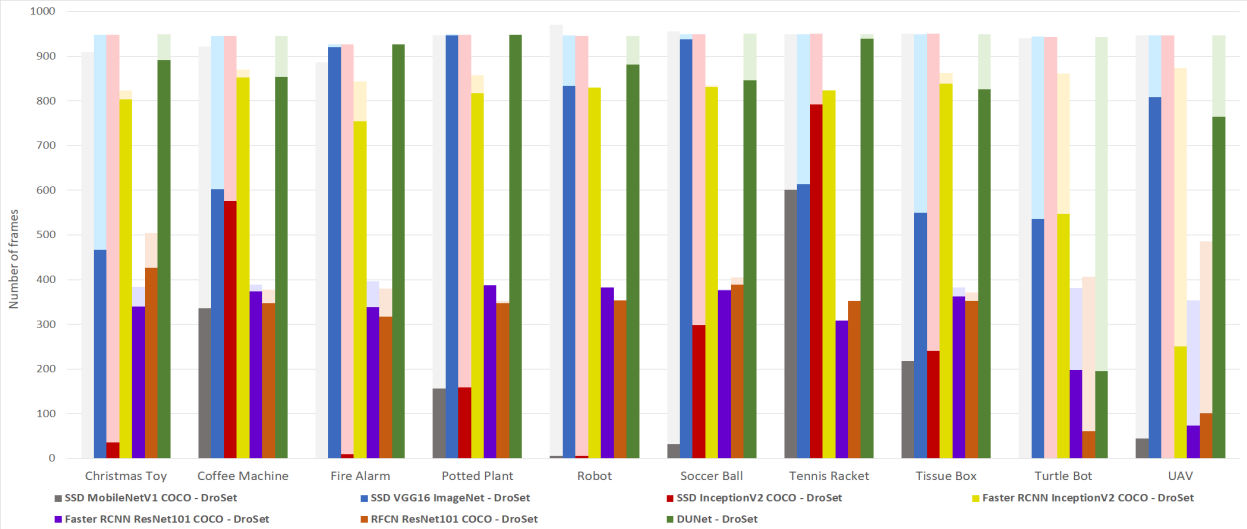


Figure 5.7: Number of processed frames and recall for each model, per DroSet category. Each column corresponds to a model, with translucent bars showing the number of processed frames and dark bars denoting the correct detections. DUNet and SSD-based models process more frames than others, and DUNet (far right column in each set) has the highest recall in almost every category. Note that many state-of-the-art models perform poorly on several categories.

Table 5.2: Direct comparison to SSD on PASCAL VOC (following protocol for Table 1 in [73] with union of PASCAL VOC 2007+2012 train/val) confirming that the proposed DUNet model is competitive on standard object detection benchmarks.

Model	aero	bike	bird	boat	botle	bus	car	cat	chair	cow	table	dog	horse	mbik	persn	plant	sheep	sofa	train	tv
SSD300	75.5	80.2	72.3	66.3	47.6	83.0	84.2	86.1	54.7	78.3	73.9	84.5	85.3	82.6	76.2	48.6	73.9	76.0	83.4	74.0
DUNet	**83.0**	**82.3**	69.5	62.5	37.7	**85.0**	**88.0**	84.2	**56.2**	76.1	73.6	80.4	**87.8**	82.5	**79.8**	46.1	**76.3**	75.0	**84.9**	**75.4**

5.3.2 Scenario II: Evaluating DUNet on Standard Benchmark

The second scenario evaluates DUNet under standard object detection conditions on traditional object detection benchmarks, against state-of-the-art models. This is primarily to confirm that our proposed meta-architecture is indeed competitive under such conditions and not overly specialized for our use case. In contrast, For better evaluating our DUNet model against other deep CNN object detection models, we need to learn DUNet on one of the generic datasets used for this kind of

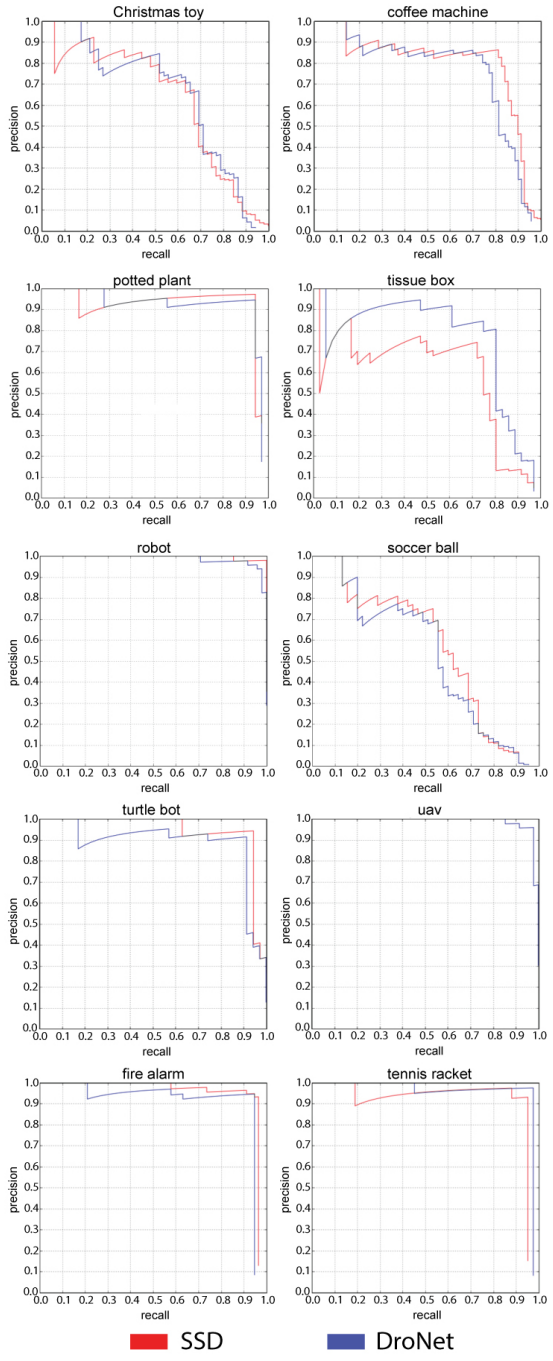


Figure 5.8: Precision - recall graphs for all DroSet objects

evaluation. In SSD [73], the authors have listed the validation results from different models in addition to two versions of SSD model for PascalVOC and MSCOCO datasets. We pick PascalVOC as a generic dataset to test it with our DUNet model. Pascal VOC07 and VOC12 training, validation, and test sets have been used in learning and evaluating DUNet. Table 4.3 show the evaluation results on PascalVOC07 test annotation file.

For this evaluation, we trained a DUNet model from scratch on PASCAL VOC [34], with the final layer replaced with PASCAL VOC object categories. We chose SSD300 as a strong baseline based on speed/accuracy results reported in Huang et al. [50] and replicated the experimental methodology described in the SSD paper [73]. We followed the standard evaluation metrics that is used for object detection models by scoring mAP and fps on the test annotation sets in order to evaluate our model against other deep CNN object detection models on a dataset other than DroSet and to use our DroSet with these models in this type of evaluation too.

A direct comparison of SSD vs. DUNet is presented in Table 5.2. The dataset was the union of PASCAL VOC07 and VOC12, with results for SSD300 (first row) copied directly from the SSD paper [73]. We see that DUNet trained from scratch performs as well as SSD300, which includes a VGG16 feature extractor trained on ImageNet. DUNet mAP is higher than SSD300 taking in consideration the mentioned training versus fine-tuning configurations as shown in Table 5.1. The precision - recall for SSD300 and DUNet320 on DroSet objects are shown in Figure 5.8. Note that we did not optimize the DUNet performance on VOC for this scenario (e.g., through hyperparameter tuning) and we still get the same mean Average Precision (mAP = 74.3). Our experiments show the effectiveness of DUNet, both in its primary role as a strong meta-architecture for training customized real-time object detectors for indoor robots, as well as its competitiveness in standard conditions.

5.4 Summary

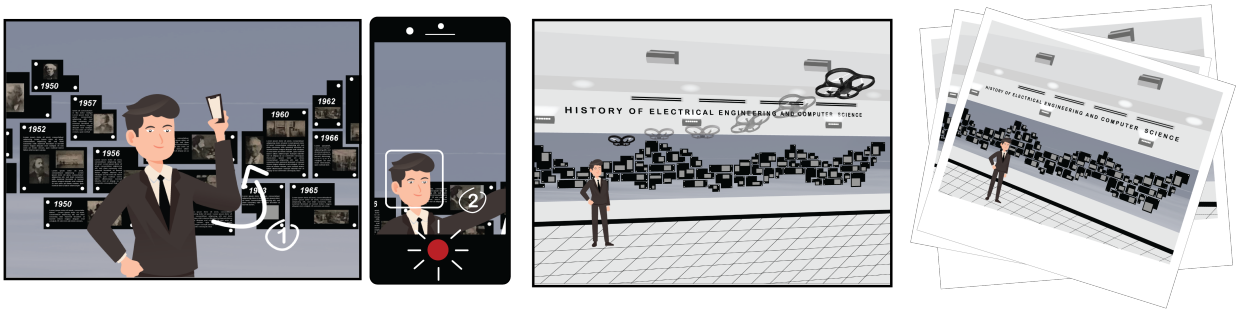
The chapter introduces DUNet, a novel meta-architecture for real-time object detection. Our design choices focus on reliable detection of small-sized objects through the use of dense blocks and top-down context, as well as customization of detectors for new object classes via training from scratch on limited datasets. We have made the data used for our evaluation publicly available—-DroSet, a dataset of indoor objects, collected semi-autonomously using a drone. This dataset consists of frame streams that can be played back in a repeatable manner so as to evaluate object detectors in robotics applications. Our experiments confirm that DUNet outperforms current state-of-the-art models on real-time object detection for indoor robotics. Additionally, even when trained from scratch, DUNet is competitive on standard object detection benchmarks. Our DUNet implementation and the DroSet dataset have been made publicly available to encourage further research in this area.

CHAPTER 6: LEVERAGING DEEP LEARNING MODELS TO CREATE A NATURAL INTERFACE FOR QUADCOPTER PHOTOGRAPHY

Although there has been prior work on the problem of improving quadcopter teleoperation [8, 65], and photography [26, 28], the premise behind most of these investigations has been that the user must learn the proposed interface paradigm. Our philosophy is to make the users learn as little as possible and the system learn as much as necessary. The *SelfieDroneStick* interface mimics the functionality of a selfie stick, enabling the user to control the quadcopter with only a mobile phone by a simple gesture and a click as shown in Figure 6.1.

The goal is to generate a well-framed selfie of the user against the desired background, as if it were taken using a virtual selfie stick extending from the user in the direction of the handheld smart mobile device (SMD). The user specifies the desired composition by taking an ordinary selfie using the SMD, where the relative orientation directly specifies the azimuth of the vantage point while the height, 3D space position, and desired distance is indirectly specified by the SMD elevation, position and size of the user's face in the captured frame respectively. The drone flies to the target viewpoint based on the vantage point specified by the SMD to capture the selfie using a learned controller. The drone mirrors the bearing of the SMD as measured by its onboard IMU and selects an appropriate distance such that the user's body visually occupies relatively the same area in the drone selfie as the user's face did in the SMD frame. The resulting photos frame the user against the entire background, just as if the user had used a very long selfie stick to compose the photograph.

Perception is accomplished with DUNet, our new meta-architecture for real-time object detection. The DUNet architecture consists of a sequence of dense blocks that process the input image at different scales, connected to a sequence of prediction layers, each of which independently generate



a | Taking a selfie. b | RL flies drone to optimal vantage point. c | Drone captures long distance selfie.

Figure 6.1: *SelfieDroneStick* enables long-range selfie photography by allowing the user (a) to easily designate a vantage point (b) from which a drone can capture well-framed photos of the user against the desired background.

detection results. Two strengths of DUNet are its processing speed and ability to reliably detect small objects; in prior work, we demonstrate its ability to learn customized object detection models for mobile robots [9]. This learning strategy eliminates the necessity of having extra convolutional layers to account for variations in appearance and orientation.

Instead of using deep reinforcement learning (RL) to learn a direct control policy based on the raw pixel data as was done in [79], our controller utilizes an abstract state space representation. First, DUNet is trained to detect the human face (which is prominent in the phone camera image) and also the human body (visible from the drone's viewpoint). We pre-train DUNet on PASCAL VOC [34] and WIDER FACE [121] datasets for human and face object categories. Deep Deterministic Policy Gradient (DDPG) [68] is then used to learn the flight policy in simulation using an abstract, continuous state space before being transferred to the real robot.

This work introduces a novel interface for automating UAV selfie-photography based on data received from any mobile device. Our system is able to capture selfies at the correct depth, background and orientation angle in the scene, with the user placed at the right position in the frame. Once trained, our RL controller autonomously flies the quadcopter to the user-selected vantage

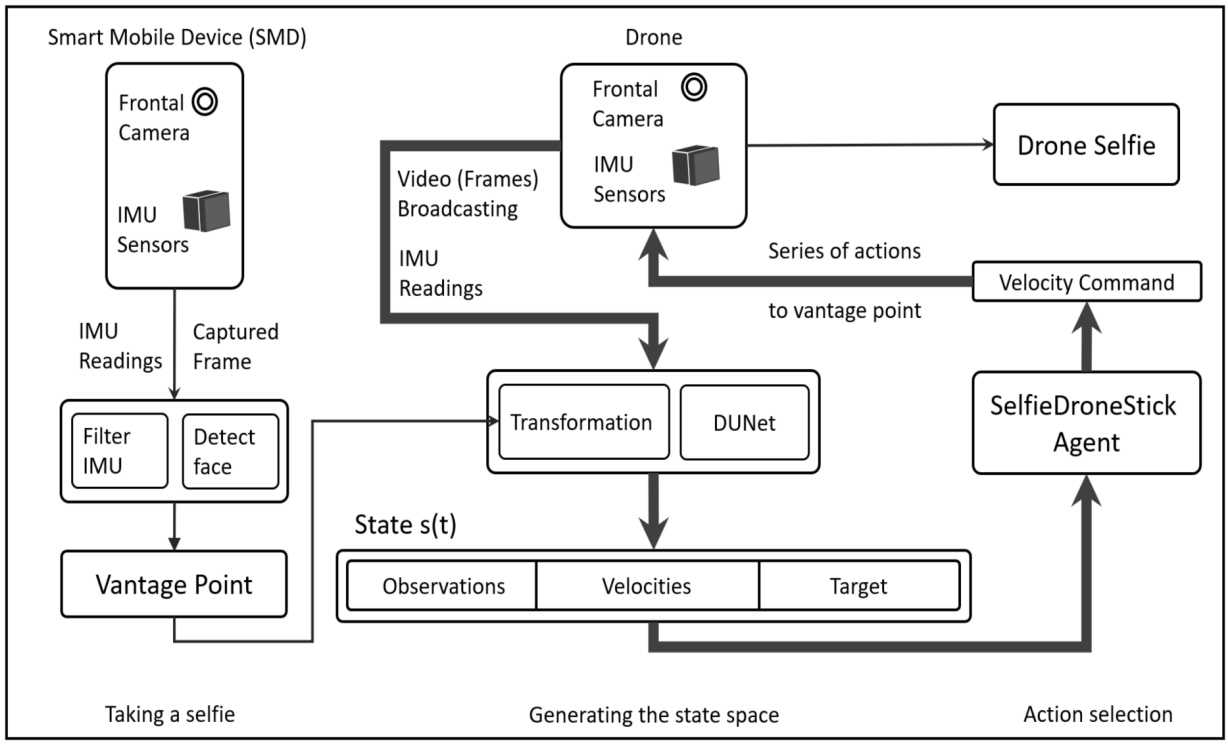


Figure 6.2: Using the device camera, the user snaps the shot as if taking a selfie while angling the phone to modify the face size and background in the captured frame. The camera frame and IMU sensor readings are then extracted to generate the desired vantage point. DUNet is employed to detect the human face and body in both the images captured by the mobile device and the drone. The transformation module creates the target point, and the SelfieDroneStick software agent autonomously navigates there using the learned control policy.

point. The system architecture for the Selfie Drone Stick is shown in Figure 6.1. The ROS configuration, simulated environment, and code are publicly available.[1]

Deep Deterministic Policy Gradient (DDPG) [68] is used to learn the flight policy in simulation using an abstract, continuous state space before being transferred to the real robot. To create a smooth and steady flight trajectory, we shape the reward to take into account both position and velocity.

[1] `https://github.com/cyberphantom/Selfie-Drone-Stick`

An overview of the *SelfieDroneStick* system is presented in Figure 6.2. First, the user specifies vantage points for the drone using an SMD, simply by clicking a series of selfies. For each vantage point, the system captures both a reference camera image as well as the SMD's corresponding orientation from its inertial measurement unit (IMU) sensor. By combining the orientation of the SMD with the framing of the user's face in the SMD camera image, we can extrapolate an ideal vantage point for the drone. This is transformed into a desired framing for the user in the drone's camera. Next, on the drone we combine information from its onboard IMU, its front-facing camera and the vantage point. Rather than tracking the user, we employ a fast object detector to localize the person in each frame (if visible) and form a state vector that is used by the SelfieDroneStick reinforcement learning (RL) agent to plan the drone trajectory to the next vantage point. Finally, the RL agent, which was trained in simulation, controls the drone via a series of velocity commands to guide it through the sequence of vantage points. When the drone is sufficiently near each vantage point, it captures a long-range selfie of the user. The next sections discuss each of these steps in greater detail.

6.1 Specifying Vantage Points using an SMD:

The user activates the SelfieDroneStick by taking a regular selfie using our web-based camera app. Then the shutter is pressed, the SMD's x-axis and y-axis orientation are recorded along with the camera image (Figure 6.3). Our approach takes the advantage of the IMU sensors and cameras that come with almost all SMD available today to gather the required information for building our novel state space. The IMU information partially specifies a bearing from the user along which the drone should seek to position itself in order to capture the desired shot. In addition to the bearing of the desired vantage point, we also need to specify the range. The key idea behind our interface is to enable the user to naturally specify the distance to the vantage point simply by varying the distance of the SMD from the user's face — moving the SMD further away should cause the

drone to capture photos from further away. Intuitively, extending the user's arm corresponds to a (magnified) extension of a selfie stick. This requires us to localize where the user is located within the selfie frame. We employ the DUNet architecture [9], a real-time object detection CNN model, for face detection and the same model is also used for person detection on the drone camera (see below). The position of the user's face (c_x, c_y), along with the ratio of the face bounding box to camera frame ω and the SMD orientation ϕ and ψ fully specify the drone vantage point; this is illustrated in Figure 6.3. In other word, from the SMD gyroscope and accelerometer we fuse and filter the IMU data to extract yaw and tilt angles and from the detected face bounding box (BBX) we obtain the location of the centroid and ratio of user's face to the SMD image. (v_{vpt}):

Engaging a professional object detector, DUNet, allows for changing the poses and orientation when taking the UAV selfie, and eases the transformation between the synthetic world and the real-world. In addition, employing a specialized object detection agent reduces the processing time as we don't want to deal with all the unnecessary pixel information when using RL to learn the controller. Moreover, when taking a selfie with a different background, we still able to get a drone-shot with the previously defined characteristics as the UAV autonomously navigate to follow the human subject. Multiple target (drone-shot vector) assignment for the UAV is handled by the TMA by taking selfies with the SMD. However, during training, we sample smd-vectors from the target generator as shown in Fig 6.2. The captured information is then passed through data fusion process to extract our smd-vector.

How do these measurements translate to a specific drone position? Capturing the desired selfie from a wider angle and higher perspectives requires scaling the vantage point data. The SMD captured frame is scaled to $[120, 150]$ width and height before it is input to our system. The quadcopter frame dimensions are configured to $[640, 360]$. The user's face to the image ratio falls in a different range to the ratio of the BBX on the user's body to the image captured by the drone. Visualizing the drone as if it were mounted on a virtual selfie stick extrapolated along the user's

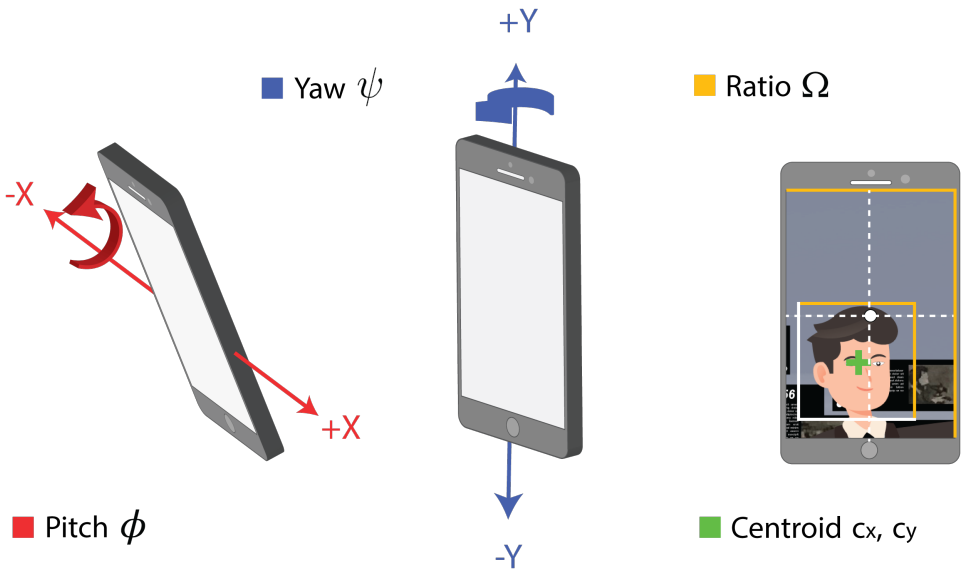


Figure 6.3: The drone vantage point is specified by the orientation of the smart mobile device, ϕ and ψ, as well as the position and size of the user in the camera frame, (c_x, c_y) and ω. Moving the SMD away from the user corresponds to extending a "virtual selfie stick", guiding the drone to capture a photo from further away.

arm, past the SMD, shows that some of the coordinates map directly from the SMD to the drone: for instance, the azimuth to the drone is simply the yaw angle of the SMD, ψ. Others require an explicit transform: we empirically observe that the ratio of the user's face to the SMD image size falls in the range of $[0.1, 0.2]$ while desirable drone shots have a ratio of user's body to drone image size of $[0.03, 0.4]$. Thus, we linearly map the ranges of the former to the latter to obtain suitable range distances for the drone. Finally, the height of the drone is derived from a combination of SMD tilt ϕ and position (c_x, c_y) and size ratio ω of the user's face in the frame using straightforward geometry.

6.2 Perception System:

There have been recent successes of Deep Reinforcement Learning that train end-to-end directly from pixel inputs in the context of automated game playing. However, for real-world robotics, it is still generally more practical to build reinforcement learning (RL) planners on top of stable perception systems.

We developed an object detector DUNet [9] (see Fig. 5.3) that is specialized for detecting small objects and optimized for inference on compute-constrained platforms. DUNet (Densely Upscaled Network) is a convolutional neural net (CNN) architecture that extends the meta-architecture of the popular SSD object detector with ideas from DenseNet, Feature Pyramid Network and Top-Down Modulation.

At a high level, DUNet consists of a sequence of dense blocks that process the input image at different scales, connected to a sequence of prediction layers, each of which independently generate detection results. The two sequences are connected both laterally (at appropriate scales) and vertically (through max pooling and upscaling). To the best of our knowledge, DUNet is the first architecture to exploit both DenseNet-style concatenation (via dense blocks) in the bottom-up pathway and ResNet-style summation (via the upscaling) in the top-down pathway.

Since RL planners are trained in simulation, we employ an instance of DUNet pre-trained on data collected in the simulated world during training and replace it with a DUNet model pre-trained on PASCAL VOC [34] for real-world deployment. This enables the RL policy learned during simulation to transfer to the real-world without requiring explicit domain transfer. Since DUNet runs at real-time (Fig. 5.5) and achieves 80% mAP on person detection (Table 5.2), we elect to perform per-frame detection rather than tracking the user through the stream of images. DUNet localizes detected objects with a bounding box and for the purposes of the RL planner, only the size and location of bounding boxes containing the user are relevant.

We believe that pixel-feature representations and the robot sensory readings are both needed to build an adequate observation state for our agent to be able to generate smooth and stable trajectories that lead to the requested goal state. Therefore, instead of learning from pixels only or relying on the low-dimensional noisy observations coming from the UAV IMU sensors, we design and carefully-select our system input state space. The drone's state space s_t composed of: observations – odometry derived from on-board sensors (gyroscope, accelerometer and pressure readings) and localization of the user in the drone's front-facing camera (generated by DUNet), (b) linear and angular velocities, (c) target – vantage point specification.

When designing a state space, it is worthwhile to focus only on the relevant features since RL scales poorly with state space dimensionality. For this reason, we condense (a) to a 5-D tuple that specifies the drone's pose relative to the user: $\mathbf{a} = [\psi^d, \Upsilon^d, c_x^d, c_y^d, \omega^d]$, corresponding to the current azimuth to the user, drone height, and observed bounding box location and size ratio, respectively; the superscript ^{d} denotes that these are all measurements on the drone rather than similar parameters measured on the SMD.

The velocity state (b) is straightforward, $\mathbf{b} = [\dot{x}, \dot{y}, \dot{z}, \dot{Z}$ consisting of three linear velocity components and the angular velocity around the vertical axis (drone yaw rate of change).

The final aspect of the state (c) is the location of the next vantage point, specified using the same coordinates as the drone pose: $\mathbf{c} = [\psi^v, \Upsilon^v, c_x^v, c_y^v, \omega^v]$, as above with superscript ^{v} denoting that this specifies the vantage point.

The target human BBX ratio can be achieved by our *SelfieDroneStick* agent through moving the quadcopter away from or toward the detected human body. A simple online object detection algorithm can be adopted to detect the human face in the SMD as it occurs only once per vantage point assignment and there is no need for a fast model.

From the detected face, we obtain the centroid point of its BBX which helps us locate the human

body in the quadcopter frame when taking the selfie with respect to the background. Both of the face and human BBXs centroid points are normalized with respect to the width and height of the SMD frame and the drone frontal camera respectively.

The drone target height is obtained by following the same procedure for the BBX ratio translation with a simple slope function after assigning the min and max thresholds for the rotation angle around the mobile device x-axis (tilting) and for the quadcopter height as well.

The complete state vector is a concatenation of these three tuples, $\mathbf{s}_t = [\mathbf{a}, \mathbf{b}, \mathbf{c}]$, resulting in a 14-dimensional state space which includes the drone's sensors and odometry readings at each time step t, in addition to the target point which is translated from the SMD vantage point. The observations consist of the human BBX centroid point and ratio that comes from DUNet which runs on the live broadcasting frames captured by the drone camera. The sensory readings are obtained by the published data from the drone gyroscope, accelerometer, and pressure sensors to provide the current height, yaw angle.

6.3 Reward Shaping:

At a high level, the goal of the *SelfieStickDrone* agent is to pilot the drone quickly and smoothly to each vantage point, without overshooting or oscillating. A naive formulation of such a goal in reinforcement learning (RL) would be to place a single sparse reward at the vantage point and to train the RL agent from scratch on a real drone. Such an approach is doomed to fail and would result in unstable drone flights that endanger both the robot and the user. In addition to these issues with safe exploration, training an RL system requires high sample complexity, and this is exacerbated by sparse reward functions.

Practical RL for robotics relies heavily on training RL policies in simulation and then transferring the learned models to the real world. It also benefits significantly from *reward shaping* and cur-

riculum learning through *staging* of rewards. We describe these in the context of our application.

Reward shaping for RL requires a careful balance between terms that are so punishing that they drive agents to absorption states (e.g., penalizing the agent at each time step in order to incentivize efficient flights could encourage the agent to end its episode quickly by crashing) and reward functions that are too rewarding near the goal that the agent chooses to dawdle near the goal state, accumulating a long sequence of partial rewards without achieving its objective.

Our reward function consists of two main terms: (1) a basin of attraction surrounding the specified vantage point to incentivize policies that fly the drone to the goal, and (2) a term to punish high-speed flight near the vantage point to encourage a smooth, non-oscillating approach before taking the selfie. The reward function also considers the fact that the drone flies through a sequence of vantage points and must smoothly transition from one to the next.

While creating a good reward function required considerable experimentation, we can explain its construction intuitively as follows. A natural expression for distance from the current drone pose to the next vantage point (goal) is given by the Euclidean distance between the corresponding pose vectors: $||\mathbf{a}-\mathbf{c}||_2 = ||(\psi^v, \Upsilon^v, c_x^v, c_y^v, \omega^v) - (\psi^d, \Upsilon^d, c_x^d, c_y^d, \omega^d)||_2$. We want the reward to decay with distance from the goal, and we also want to penalize speed when near the goal. From these, we propose the following formulation for the reward:

$$R = \text{CLIP}_{[0,1]}\left(\cos(\gamma e^{-\alpha||\mathbf{a}-\mathbf{c}||_2})e^{-\beta||\mathbf{b}||_1}\right), \tag{6.1}$$

where $\text{CLIP}_{[0,1]}(.) = \max(\min(.,1),0)$, $\{\mathbf{a},\mathbf{b},\mathbf{c}\}$ are the three components of the RL state vector (described above) and $\{\alpha = 1.3, \beta = 0.35, \gamma = 11.3\}$ are empirically tuned hyperparameters. The $\cos(.)$ term bounds the basin around the vantage point where the agent can collect partial rewards and high speeds are not rewarded near the goal. We include the cos function to reduce the conflict that may occur between the adjacent targets locations as our *SelfieStickAgent* operates on multiple

targets and has the ability to switch to any target from any position upon requested by the user. α is used to attenuate the gradient of the reward as the drone approaches the goal. β is used to adjust the velocity effect on the reward function. With this scenario, we can control the agent greediness in being closer to the target location and orientation by choosing the best path with the highest possible speed without loosing the human object before starting to give more attention about reducing its speed and stays as closer as possible to the goal state to end up the episode with a successful terminal condition. The rewarding balance between the distance Δdis and the drone velocities $\overline{vel}$ is shown in Figure 6.4.

Each episode runs until one of the following termination conditions is triggered:

- If the agent achieves a reward of > 0.85 in a given timestep, the episode is terminated as an early success, with a reward $+1$.
- If the drone flies outside the safe zone (exceeds height or ratio limits), the episode is terminated as an early failure, with reward -0.8.
- If the object detector fails to find the subject (e.g., user is out of view), the episode is terminated early, with reward -0.8.
- If the step counter reaches the max. episode length (41 in our experiments), the episode terminates with the current reward, R.

At each time-step, the agent receives a reward R except for two cases, where the reward is explicitly shaped:

1. An exploration reward of $+1$ is given whenever the agent achieves $R > 0.75$; this is to encourage it to explore nearby states to achieve early success.

2. When the drone moves so that the detected person falls very close to the edge of the image (within 10 pixels of frame), the reward is set to -0.8, but the episode is not terminated early; unless the drone acts quickly, the person will go outside the frame and trigger the early condition discussed above.

The reward shaping incentivizes the agent to keep the user in its field of view and to stay within the safe zone. Once it can achieve these basic objectives, the agent can learn to fly the drone towards the vantage point. Our *SelfieStickDrone* agent is required to pilot the drone to the vantage point smoothly and with minimum flying time. Optimal flying behavior can't be achieved without encoding some information about the task especially when having multiple goals as sparse reward function leads to flying the drone in unstable trajectory and with a lot of fluctuating when reaching to the goal state. Shaping the reward function reduces the training time because the agent is getting information about how well it is performing during environment exploration. In addition, by staging and shaping rewards, multiple variables in error calculations are engaged in a way that achieves a smooth continuous gradient which creates a balance between the greedy behavior in reaching the target and the speed consideration to reach our narrow success criteria. The agent learns that the velocity starts to matter when get closer to the goal and stabilizes the trajectory by gradually slowing down and preparing to stop ahead before reaching the target.

A few other subtleties are worth mentioning: we provide the maximum reward $+1$ whenever the drone is within a radius of $1m$ of the vantage point and within $\pm 10^{\circ}$ in orientation. This acknowledges that the user's specification of the vantage point is intrinsically approximate and encourages the drone to fly quickly to the vantage point and take selfies while hovering rather than making unnecessary minor adjustments in an attempt to hit the exact position, which would have no meaningful impact on the qualities of photos acquired. The zero reward that the agent receives when it is far away from vantage points encourages the drone to fly quickly through those regions and the penalty for failing to detect the user is an incentive to keep the user in view (when possible).

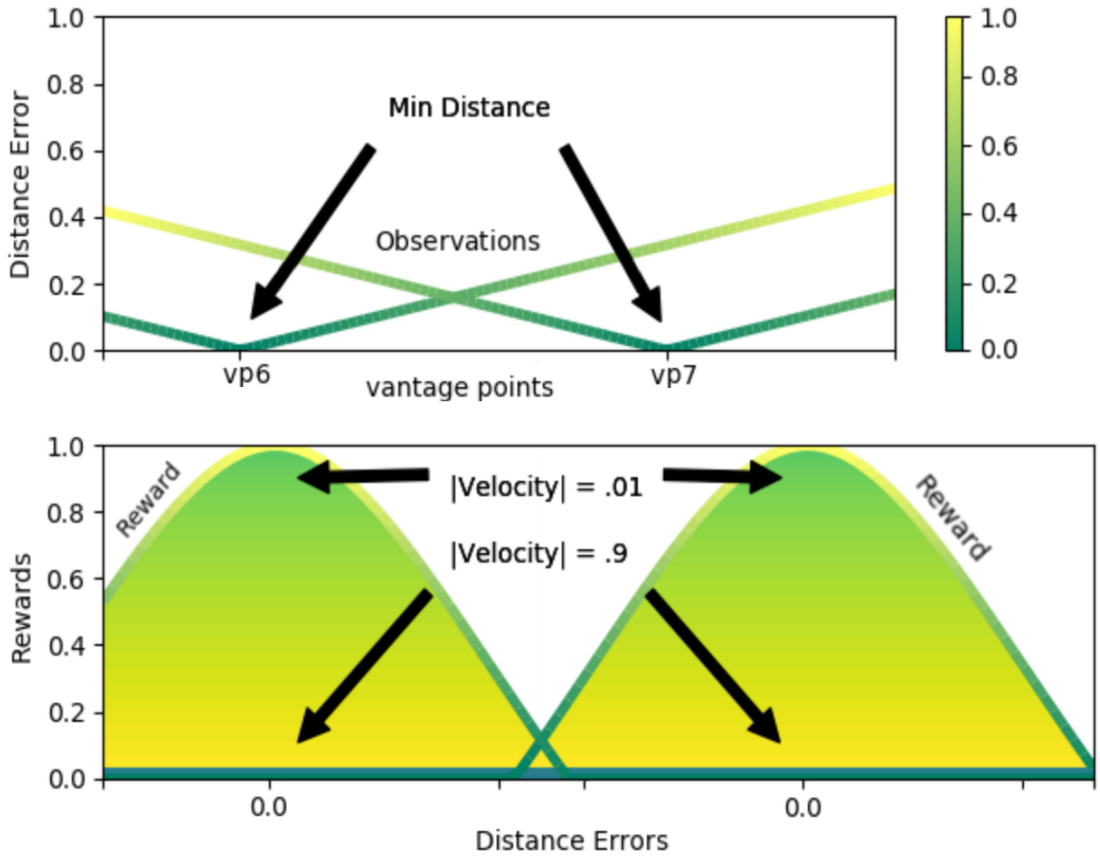


Figure 6.4: Shaping the reward function for distance vs. velocity to achieve a stable trajectory and a good quality long-range selfie by the drone.

The latter is a form of curriculum learning: the drone first learns to explore while keeping the user in view and then learns to head to the vantage point – without losing the user.

6.3.1 Training Deep RL in Simulation and Transfer to Real-World

As discussed above, it is almost impossible to train a deep reinforcement agent to fly a real-world drone from scratch on this task. Therefore, we trained our system in a 3D emulator with a custom-built environment and a physical simulation of our quadcopter. The emulator creates a series of

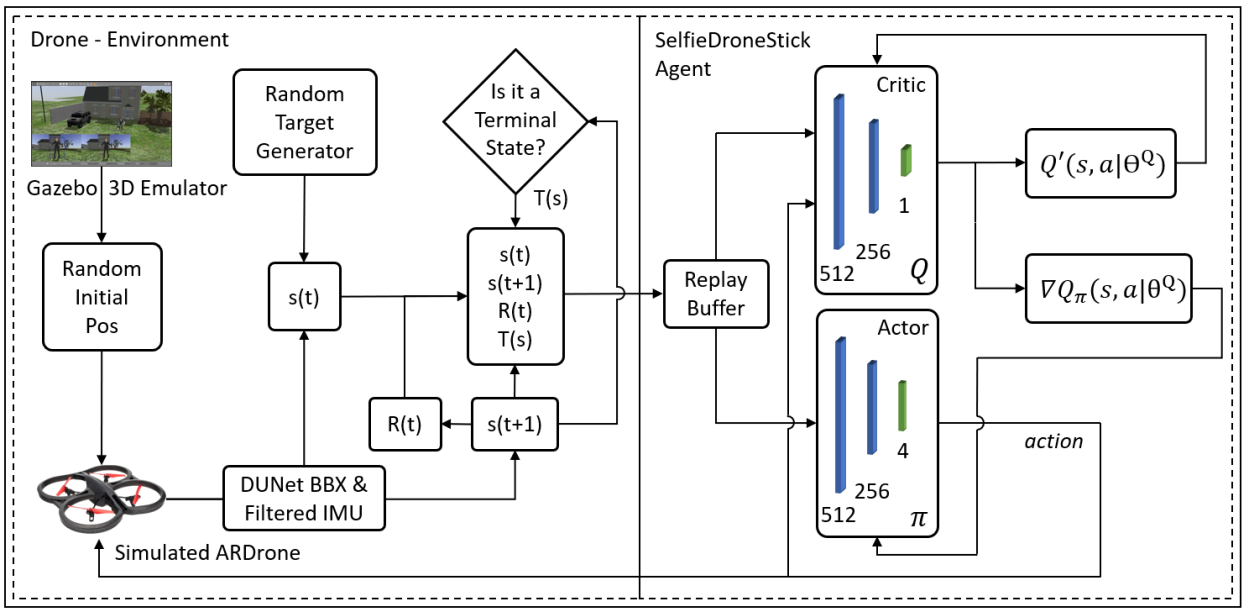


Figure 6.5: Overview of the Deep RL agent training pipeline (target networks not shown). During each episode, the simulated drone is initialized in a random pose and assigned a random vantage point as target. The target critic network predicts the Q-value (Q') and the critic network provides the action gradients (∇Q_π).

training simulations for the SelfieDroneStick agent with different initialization poses and vantage points. The agent tries actions, observes a new state and collects rewards at each episodic time-step t. We explored training the agent with a variety of Deep RL algorithms (described below) but at a high level, the goal is to learn a policy $\pi(\mathbf{s})$ by updating a value function $Q_\pi(\mathbf{s}, \mathbf{a})$, where $\mathbf{s}$ and $\mathbf{a}$ denote the agent's current state and its selected action, respectively.

In spite the fact that discretizing the action space is the most common procedure that is followed by researchers to train recent RL algorithms such as DQN and the policy gradient method. However, this leads to having an infinite number of actions if we consider multiple velocities which compromises our neural network convergence if done at a fine granularity but creates oscillations when done coarsely to reduce the action space size. Each trajectory contains a series of actions where each action $a \in A$ at each time-step t per state in our continuous state space $s \in S$ for a speci-

fied number of time-steps T. Our action space is 4D and continuous: 3 linear velocities $(\dot{x}, \dot{y}, \dot{z})$ and a yaw rate $\dot{Z}$. Typical RL agents in the literature select an action from a discrete set (e.g., a game playing RL agent that selects which button to press on a game controller) and learning in a multi-dimensional continuous action space, as is common in robotics, is more challenging.

Learning a deep NN on continuous action space becomes a challenging problem when using four different velocities as a single action at each time-step. DQN cannot be trained on continuous action space. Policy Gradient (PG) is a state-of-the-art RL algorithm that can be used in either a discrete or continuous action space. Since finding the optimal policy and value function is the key success for solving RL problems, PG uses actor-critic to decouple learning the policy from the value function. During our evaluation of the stochastic Gaussian policy gradient, we experienced convergence problems, accompanied by noisy gradients and high variance. Thus, for our system we adopted Deep Deterministic Policy Gradient (DDPG) [68], which is based on the prior DPG [107] work.

DDPG is an off-policy, model-free learning algorithm that yields good environment exploration through the use of a stochastic behavioral policy. The intuition is that it is easier to learn the optimal Q value by employing greedy deterministic policy learning through following the time difference (TD) bootstrapping error. The actor takes the state and predicts the action using its policy network, and the critic provides the Q value (expected return) based on the state and the actor's predicted action. Optimizing the Q value of the critic network is done by minimizing the loss between the prediction of the critic target network and the expected return found using the Bellman equation at each time step:

$$y_i^{\text{target}} = r_i + \gamma Q^{'}(s_{i+1}, \pi^{'}(s_{i+1}|\mu^{'})|\varphi^{'}), \tag{6.2}$$

$$\text{loss} = \frac{1}{N}\sum_{i}(y_i^{\text{target}} - Q(s_i, a_i|\varphi)^2), \tag{6.3}$$

where π is the actor policy network, $\pi^{'}$ is the actor policy target network, Q critic value function network, $Q^{'}$ critic target value function network, μ actor network hyper parameters, φ critic network hyper parameters. y^{target} can be estimated from the target networks for both actor and critic normalized by a batch (size $N = 256$) sampled from the experience replay. A replay buffer is used to eliminate correlation between successive samples, reduce the variance, and avoid falling into local maxima. Instead of using the actor and critic networks directly, target networks are used to learn TD error targets in order to add stability and regularization to the learning process. The loss function is computed by finding the normalized difference between the estimated Q value from the target critic network and the expected return for the current state.

Gradients obtained from the loss function are used to update the critic network parameters through value iteration. The actor network is optimized through the deterministic policy toward the direction of the gradient coming from the critic network for the action-value function.

$$\nabla_{\mu}\pi \simeq E[\nabla_{\mu}\log\pi^{\mu}(s|\mu).\nabla_{a=\pi(s)}Q_{\pi}(s, a|\varphi)] \tag{6.4}$$

$\nabla_{\mu}\log\pi^{\mu}(s|\mu)$ is the policy gradient with respect to the actor network parameters μ. $\nabla_{a=\pi(s)}Q_{\pi}(s, a|\varphi)$ is the action-value gradient of the critic network with respect to the actor's action.

To improve the exploration of the environment, random noise is added using an Ornstein-Uhlenbeck process for each dimension to the action prediction of the actor network for the first 512 episodes. Each episode starts from a randomly initialized position and orientation to add more exploration variance. The training phase is illustrated in Figure 6.5. When the Q_{max} value function stabilizes to the highest return, the training is stopped (Figure 6.7). After the optimal policy is learned, the controller is transferred to the real-world with a physical quadcopter considering real-world data

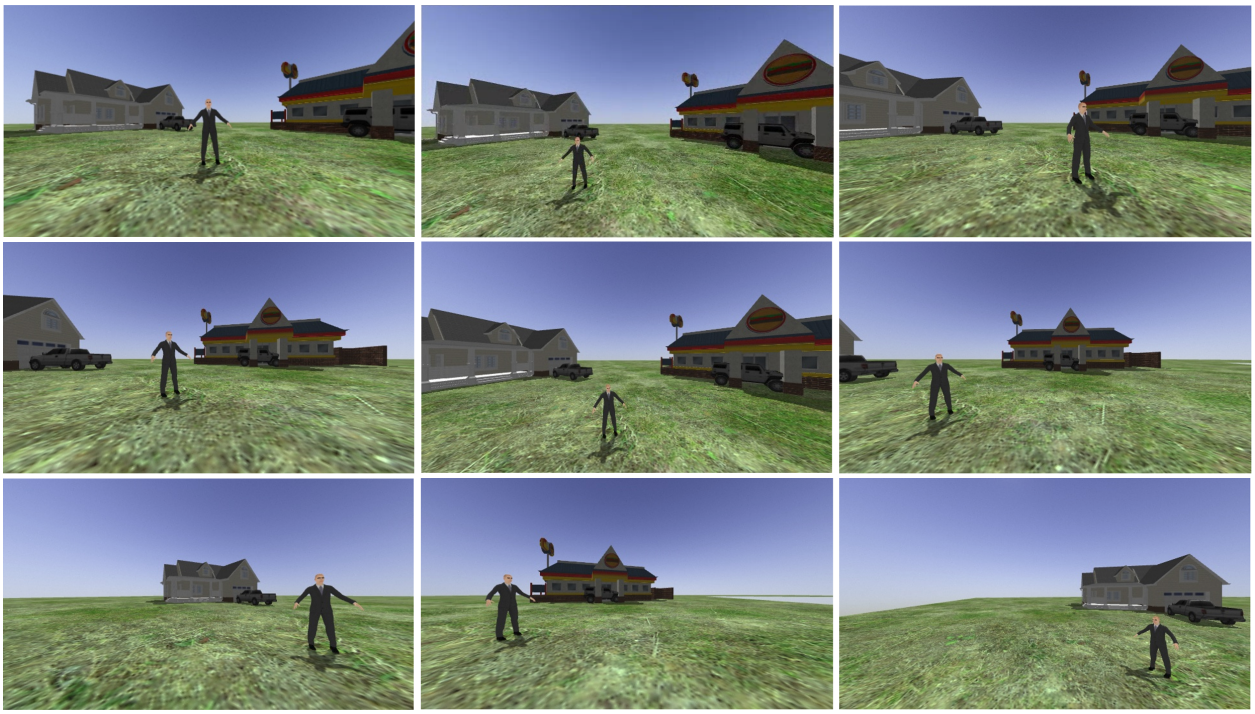


Figure 6.6: Samples drawn from the simulated training environment in which the Deep RL agent is trained to fly against varying backgrounds from different vantage points.

with DUNet object detector.

6.4 Results

With our state space design, moving the learned model from the synthetic environment to the real-world is straightforward. The Gazebo simulator and ROS were configured to work with the OpenAI Gym toolkit to observe a new synthetic state every 160ms. Training was performed on a single NVidia Titan X GPU. To expedite the training, we set threshold values for the human object ratio and drone height to generate a safe zone for the drone movement. The vertical space is set to be in the range of $[0.5, 3]$ meters and the yaw angle to be between $[-75°, 75°]$. To prevent the quadcopter from being too close to the detected human or moving too far and losing references,

the human size ratio is only valid in the range $[0.5, 0.03]$. The 10 pixels from each side of the quadcopter frame which are $[0-10, 630-640]$ horizontally and $[0-10, 350-360]$ vertically are set as thresholds to avoid losing the human subject during trajectory. Network hyper parameters and structure are set to meet the wide range of states in the continuous state space and to compensate for the multiple target goal assignments. Both the actor and critic networks have two hidden layers with sizes $[512, 256]$. The Adam optimizer [59] is used for training with the learning rates set to 10^{-5} and 10^{-4} respectively. Elu activation function and the batch-normalization layers are eliminated as their effect is very minimal with such low capacity network. The output action selected by the actor is scaled to be in the range of $[-0.8, 0.8]$ as we are using a commodity quadcopter that exhibits shaky motion when flying faster than 0.8 ($\simeq 4m/s$) in any direction; this camera shake results in significantly degraded image quality and detection performance. The Deep RL agent is trained for 18K epochs in our simulated environment and then deployed on the drone (see Figure 6.7).

To evaluate our system performance we ran multiple tests. In the first scenario, we assigned multiple targets in the simulated world and the agent performance is tested when it is trained with a linear reward function based on the distance measurement vs. our shaped reward function. A summary on the performance of our approach is made by measuring the mean distance for the steps in each of the four initial starting points and the variance in the last ten steps of the trajectory of the roll, pitch, vertical incline, and yaw velocities as shown in Table 6.1. The trajectory in achieving the vantage point from the single forward density actor network is illustrated in Figure 6.8. The reward shaping importance is clearly shown with respect to the trajectory with sparse rewarding. In addition, a PID controller is used as another baseline to benchmark our approach. With PID, it is noticed that the risk of loosing the reference which is the detected human body is very high if the initial point is far from the vantage point. However, if it is close, the PID is very stable in reaching the vantage point and hover in place until the trajectory cycle finished. A solution for

that is to reduce the speed, but that increases the time required to reach the target which is not a recommended solution with the low battery life of the drone.

During training, our environment consists of a 3D domain containing a single human and single UAV (drone) model against a simple background. Take-off is done once at the process initialization with making sure that the drone is facing the human subject. Then, random initial starting point is generated for each episode. Also, the random target generator assign a target (Which mimics the vantage point-target transformation process) for that episode to be attached with the state before being included in the replay buffer. Some examples of the target generator output is shown in Figure 6.6. We test the system in several realistic simulated environments, such as the one shown in Figure 6.9. This allows us to conduct end-to-end experiments under repeatable conditions with known ground-truth, with the same perception system (DUNet) as we employ for real-world testing, as shown in Figure 6.10.

Both the simulated and real-world scenarios follow a consistent script:

1. The drone is initialized facing the human subject at a safe distance. In simulation, take-off and landing are straightforward; for the real drone, the user initiates take-off by holding the SMD flat and initiates a landing sequence by tilting the SMD 90° around the x-axis.

2. Prior to activating the SelfieDroneStick, the drone hovers in front of the user, centering the user in the middle of the image with $\Omega_{obs} \simeq 24\%$.

3. Once the SelfieDroneStick agent has been activated by the user taking a selfie using the SMD, the SelfieDroneStick agent flies the drone to the specified vantage point using the learned Deep RL controller.

4. Once the drone arrives at the bearing and range consistent with the specified vantage point, it takes a long-range selfie of the subject.

Table 6.1: Contrast between standard and reshaped reward functions. The mean distance is for calculated after episode completion. The variance are calculated for the last 10 steps in each episode to measure the stability of the drone when reaching the vantage point (Each episode is assigned to have 41 steps).

Init. Start	Standard Reward f.			Reshaped Reward f.			PID		
	Dist	Var. Vel	Avg. Rwd	Dist	Var. Vel	Avg. Rwd	Dist	Var. Vel	Avg. Rwd
P1	0.133	0.168	0.153	0.144	0.141	**0.405**	0.275	0.003	0.018
P2	0.235	0.0707	0.053	0.148	0.148	**0.334**	0.257	0.004	0.017
P3	0.089	0.155	0.348	0.10	0.147	**0.495**	0.202	0.007	0.031
P4	0.124	0.170	0.06	0.085	0.144	**0.55**	0.110	0.003	0.112

The obtained reward after learning from the simulated data is shown in Figure 6.7. The learned controller is then evaluated in realistic simulated scenarios (Figure 6.9).

SelfieDroneStick generalizes the experience learned in the emulator to a new environment outside its domain considering the real world observation as a game state. Therefore, learning is performed on a plain 3D emulated domain with a simple synthetic environment with a human and UAV models. We test our platform on a several designated emulated domains before testing in real-world. Learning is performed with a single NVidia Titan X GPU on an external machine. The GPU is used to support training on a larger batch of samples and to run the DNN of our human body object detector. The *SelfieDroneStick* interface operating in a real-world setting for the three scenarios shown in the previous experiment (easy scenario shown in center) is illustrated in Figure 6.10. These experiments employed an iPhone SMD in conjunction with an AR.DRONE 2.0 UAV with a 30fps frame rate. The interface enables the user to take multiple selfies with different backgrounds as the user moves in the environment. Videos of the system can be viewed at `http://ial.eecs.ucf.edu/SelfieDroneStick/`.

It is worth considering a closed loop feedback controller as a non-learning method to solve this problem. A traditional PID control loop is used as the baseline system that performs visual servoing

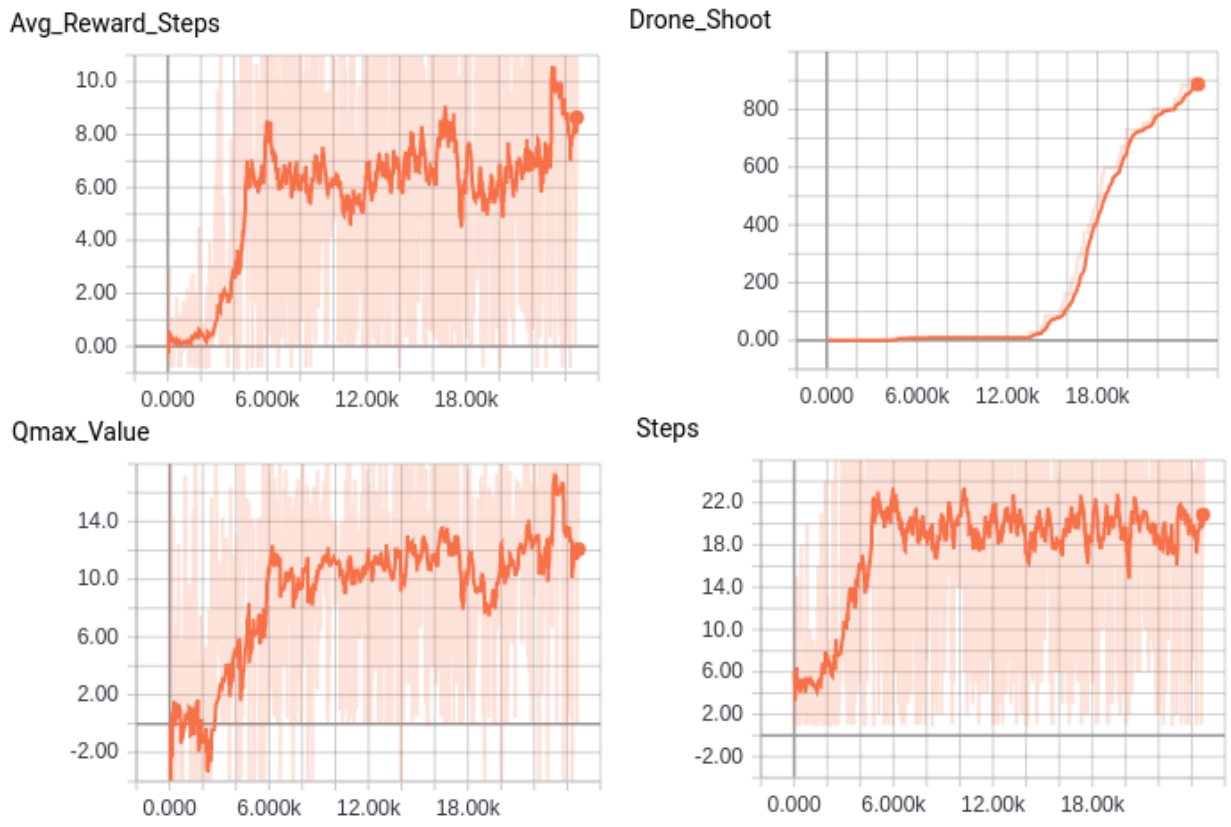


Figure 6.7: Episodic rewards, drone shots counter, Q-value, and # steps per episode are monitored and recorded during our training. As can be noticed from the # steps development, our reward function incentivizes the RL agent to first focus on the user. Then it learns to fly the drone to the specified vantage point from a variety of random initialized positions and orientation angles. Training converges in 19K epochs and the agent starts achieving the target position and orientation for the drone-shoot, after which we transfer the agent from simulation to the physical drone.

to bring the drone to the vantage point. Architecturally, it is identical to the Deep RL version where both are fed with continuous state space information and predict continuous actions. The reference of the input state is based on the occurrence of the target human body. With PID if the agent loses the detected human, it is hard to continue with the trajectory. Whereas with DRL, the agent is trained to predict the action ahead of time for the future state, and this helps the quadcopter avoid losing the user. The trained RL agent has already seen such cases and received punishment, so in the testing phase the agent attempts to avoid it. However, the PID controller has good behavior in

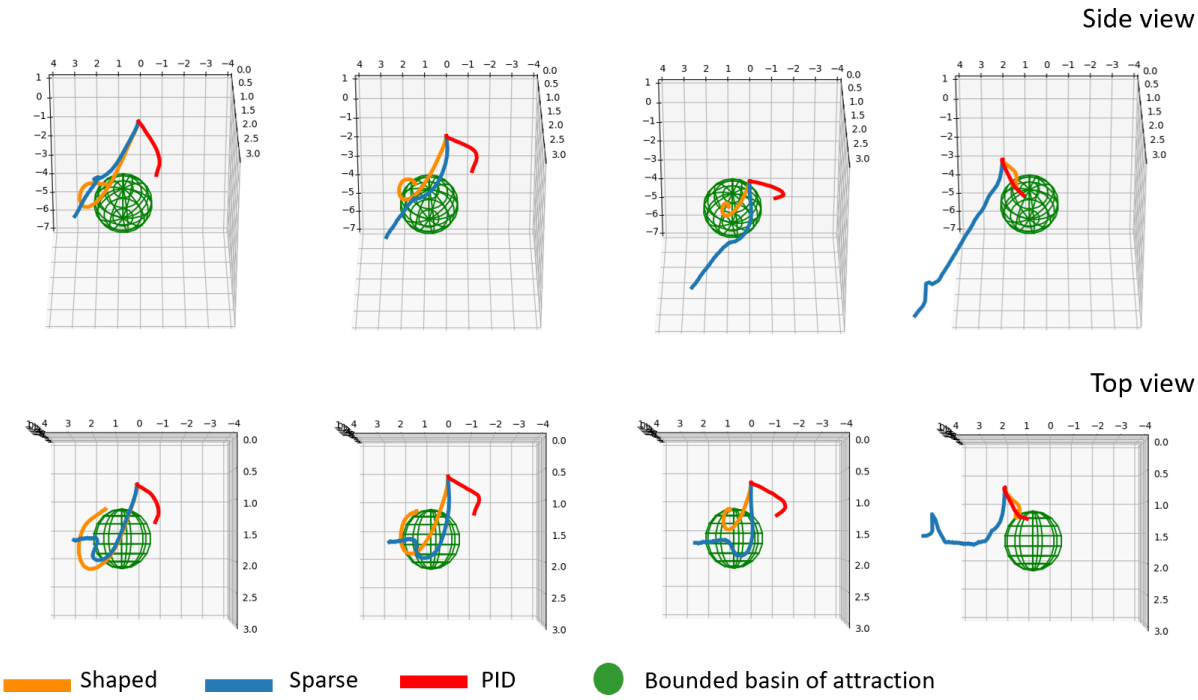


Figure 6.8: Importance of reward shaping: with sparse rewards, the drone is forced to explore randomly while the shaped reward guides the drone to the vantage point. The green sphere represents the target position with error threshold. A PID trajectory is plotted as an additional baseline.

keeping the drone in a steady state if it is initialized near the vantage point.

6.5 Summary

This chapter describes the most significant lessons learned while designing and training the deep learning modules for our *SelfieDroneStick* system. Deep learning can help make the user experience more natural; rather than having the photographer learn how to use the system, the system should learn the flight controller that best expresses the user's wishes. Although our current research utilizes reward shaping to express human preferences in advancing the trajectory stability until reaching the target without any oscillations and through making the appropriate decisions in

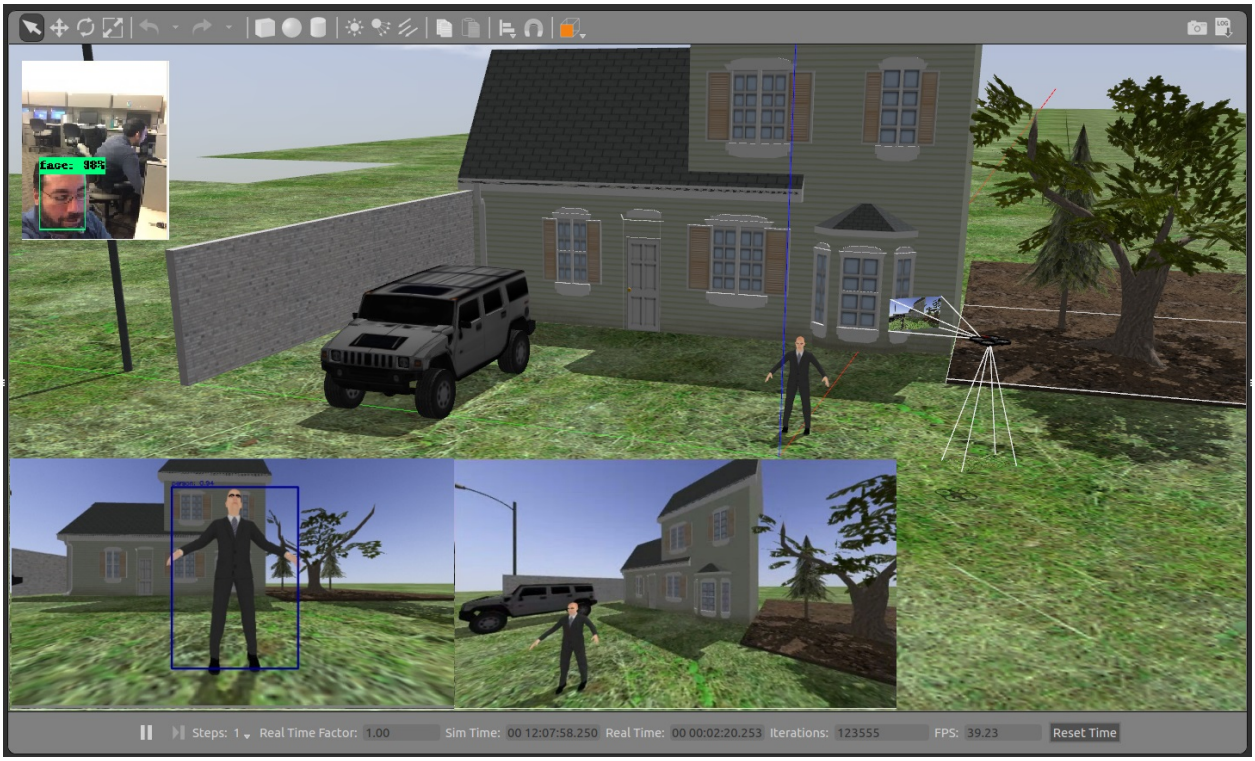


Figure 6.9: Evaluating *SelfieDroneStick* in a realistic simulation environment. Bottom left: initial view of subject from drone. Top left: user specifies a target vantage point using SMD. Bottom: Deep RL controller navigates drone to vantage point and captures long-range selfie.

selecting actions for the successive states that lead to a smooth stopping.

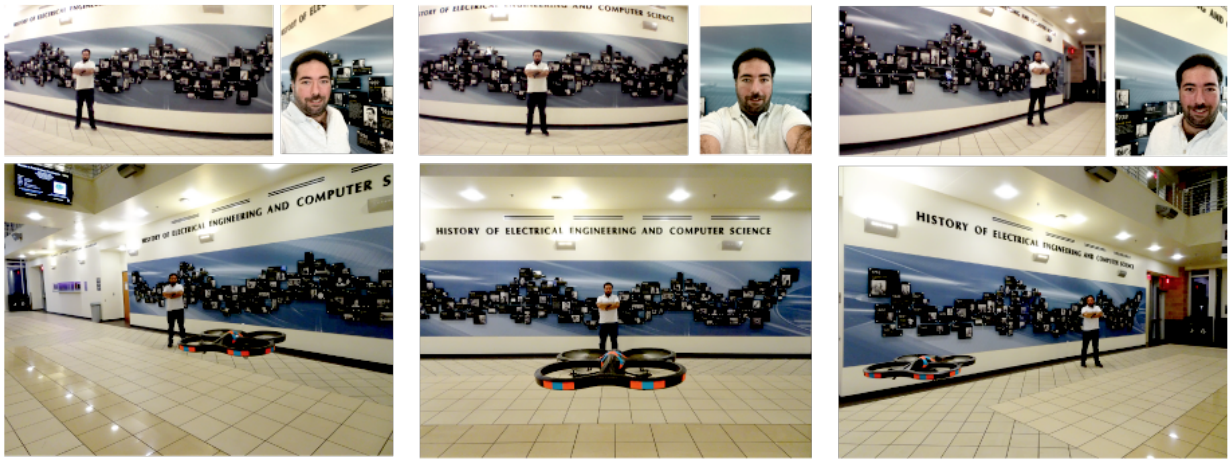


Figure 6.10: Testing with Parrot ARDrone 2.0 in indoor real-world environment though assigning three different vantage points using a SMD and transforming them to correspondents from the drone perspectives for taking the long-range selfie.

CHAPTER 7: CONCLUSION AND FUTURE WORK

7.1 Conclusion

This dissertation presents three major contributions to the problem of guided autonomy for quadcopter photography. First we introduce a sketch-based user interface (SUI) which operates in two modes: 1) direct velocity command transformation from the gestural strokes sketched by the user; 2) semi-autonomous interaction through engaging a smart agent as a co-pilot to assist in capturing photographs with the frontal camera. The SUI facilitates drone control and sends more precise velocity commands than other means of robot interaction. SUI relieves the navigation load from the user in order to perform custom photography for any object in real-time. The imagery samples with the corresponding annotations are used to construct the visual dataset used for training a CNN-based object detection model which requires a relatively rich dataset with labeled and annotated objects in images. Results from several experimental scenarios demonstrate that our interface outperforms standard commercial and official solutions, such as joystick and AR.Free Flight. The participants were able to robustly execute navigation patterns and collect visual datasets without crashing and expressed satisfaction with the user experience.

DUNet, the second contribution in this research, is a novel meta-architecture for real-time object detection specialized to work with relatively small datasets without the need for previous knowledge or transferred representation from other trained models. Our design choices optimize the reliable detection of small-sized objects through the use of dense blocks, top-down context with its detection meta-architecture, as well as customization of detectors for new object classes via training from scratch on limited data. DroSet, an imagery dataset of eleven labeled and bounding boxes annotated objects collected in an indoor environment, was made publicly available as part of this dissertation to encourage further research in this area. This dataset consists of frame streams

that can be played back in a repeatable manner so as to evaluate object detectors in robotics applications. Our experiments confirm that DUNet outperforms current state-of-the-art models on real-time object detection for indoor robotics with with high recognition efficiency on the human body. Additionally, even when trained from scratch, DUNet is competitive on standard object detection benchmarks.

The third contribution in this dissertation is designing and training a guided autonomy drone flying and photography agent, *SelfieDroneStick*. This DRL agent learns to execute trajectories that reach the vantage point requested by the user. Training is performed on observations that are collected when flying a simulated quadcopter in a synthetic environment. Staging and shaping the reward function facilitates learning a stable trajectory by ensuring the balance between speed vs. distance which leads to reducing the oscillation during flight. With this platform, the flight experience becomes more natural when taking a selfie with a Smart Mobile Device (SMD).

7.2 Future Work

Possible future directions that target the advancement of Human Robot Interaction (HRI) and visual servoing with aerial robots are highlighted below:

- Developing the Smart User Interface (SUI) to include extra levels of automation and visual augmentation to create more variations in the samples of the generated imagery dataset that may engender higher performance in object detection.
- Augmenting DUNet architecture to include components from newer deep learning architectures such as Capsule Networks (CapsNets).
- Using a learning from demonstration approach to reduce the DRL training time which is required with a large number of vantage points. *SelfieDroneStick* is trained in an off-policy

manner; during its training, it was noticed that the model is highly dependent on the collected experience.

- Another possible trend is improving the autonomous behavior during the trajectory to achieve the vantage point by providing analytical factors from the surroundings.
- The photography task can be advanced to achieve more complex videography for the object from different viewpoints and heights at the same time through engaging aerial swarm robots with a cooperative multi-robot system.

APPENDIX : PERMISSION TO REUSE PUBLISHED MATERIAL

LIST OF REFERENCES

[1] Ar.drone simulator. http://wiki.ros.org/tum_simulator. Accessed: 2019-03-23.

[2] Cosimir simulator. http://www.festo.com. Accessed: 2019-04-04.

[3] makehuman software. http://http://www.makehumancommunity.org/. Accessed: 2019-04-04.

[4] opensim simulator. http://www.opensimulator.sourceforge.net. Accessed: 2019-04-04.

[5] Robot web tools. http://robotwebtools.org. Accessed: 2019-03-23.

[6] webots simulator. http://www.cyberbotics.com. Accessed: 2019-04-04.

[7] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[8] S. Alabachi and G. Sukthankar. Intelligently assisting human-guided quadcopter photography. In *Proceedings of Florida Artificial Intelligence Research Society*, Melbourne, FL, May 2018.

[9] S. Alabachi, G. Sukthankar, and R. Sukthankar. Customizing object detectors for indoor robots. *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[10] B. Alexander, K. Hsiao, C. Jenkins, B. Suay, and R. Toris. Robot web tools [ROS Topics]. *IEEE Robotics & Automation Magazine*, 19(4):20–23, 2012.

[11] P. Benavidez, J. Lambert, A. Jaimes, and M. Jamshidi. Landing of an ardrone 2.0 quadcopter on a mobile base using fuzzy logic. In *2014 World Automation Congress (WAC)*, pages 803–812. IEEE, 2014.

[12] S. Bernardini, M. Fox, and D. Long. Planning the behaviour of low-cost quadcopters for surveillance missions. In *ICAPS*, 2014.

[13] R. B. Bloom and B. Foreword By-Behlendorf. *Apache Server 2.0: The Complete Reference*. McGraw-Hill, Inc., 2002.

[14] M. Blösch, S. Weiss, D. Scaramuzza, and R. Siegwart. Vision based mav navigation in unknown and unstructured environments. In *2010 IEEE International Conference on Robotics and Automation*, pages 21–28. IEEE, 2010.

[15] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. Visual object tracking using adaptive correlation filters. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2544–2550. IEEE, 2010.

[16] H. Bou-Ammar, H. Voos, and W. Ertel. Controller design for quadrotor uavs using reinforcement learning. In *2010 IEEE International Conference on Control Applications*, pages 2130–2135. IEEE, 2010.

[17] S. Bouabdallah and R. Siegwart. Backstepping and sliding-mode techniques applied to an indoor micro quadrotor. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 2247–2252. IEEE, 2005.

[18] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4243–4250. IEEE, 2018.

[19] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[20] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[21] Z. Byers, M. Dixon, K. Goodier, C. M. Grimm, and W. D. Smart. An autonomous robot photographer. pages 2636–2641, 2003.

[22] Z. Byers, M. Dixon, W. D. Smart, and C. M. Grimm. Say cheese! experiences with a robot photographer. *AI magazine*, 25(3):37–37, 2004.

[23] J. Campbell and P. Pillai. Leveraging limited autonomous mobility to frame attractive group photos. pages 3396–3401, 2005.

[24] T. G. Carreira. Quadcopter automatic landing on a docking station. *Instituto Superior Técnico*, 2013.

[25] P. Castillo, A. Dzul, and R. Lozano. Real-time stabilization and tracking of a four-rotor mini rotorcraft. *IEEE Transactions on control systems technology*, 12(4):510–516, 2004.

[26] E. Cheng. *Aerial photography and videography using drones*. Peachpit Press, 2015.

[27] F. Chollet et al. Keras, 2015.

[28] R. Coaguila, G. Sukthankar, and R. Sukthankar. Selecting vantage points for an autonomous quadcopter videographer. In *Proceedings of Florida Artificial Intelligence Research Society*, pages 386–391, Key Largo, FL, May 2016.

[29] D. Cummings, S. Fymat, and T. Hammond. Sketch-based interface for interaction with unmanned air vehicles. In *Extended Abstracts on Human Factors in Computing Systems*, pages 1511–1516. ACM, 2012.

[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE CVPR*, 2009.

[31] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[32] J. Engel, J. Sturm, and D. Cremers. Accurate figure flying with a quadrocopter using onboard visual and inertial sensing. *Imu*, 320(240), 2012.

[33] J. Engel, J. Sturm, and D. Cremers. Camera-based navigation of a low-cost quadrocopter. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2815–2821. IEEE, 2012.

[34] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, Jan. 2015.

[35] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[36] R. A. S. Fernández, J. L. Sanchez-Lopez, C. Sampedro, H. Bavle, M. Molina, and P. Campoy. Natural user interfaces for human-drone multi-modal interaction. In *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, pages 1013–1022. IEEE, 2016.

[37] C. Forster, S. Lynen, L. Kneip, and D. Scaramuzza. Collaborative monocular slam with multiple micro aerial vehicles. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3962–3970. IEEE, 2013.

[38] C. Forster, M. Pizzoli, and D. Scaramuzza. Air-ground localization and map augmentation using monocular dense reconstruction. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3971–3978. IEEE, 2013.

[39] D. Gandhi, L. Pinto, and A. Gupta. Learning to fly by crashing. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 3948–3955. IEEE, 2017.

[40] S. García, M. E. López, R. Barea, L. M. Bergasa, A. Gómez, and E. J. Molinos. Indoor slam for micro aerial vehicles control using monocular camera and sensor fusion. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 205–210. IEEE, 2016.

[41] C. Gebhardt, B. Hepp, T. Nägeli, S. Stevšić, and O. Hilliges. Airways: Optimization-based planning of quadrotor trajectories according to high-level user goals. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 2508–2519. ACM, 2016.

[42] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[43] R. Girshick. Fast R-CNN. *arXiv preprint arXiv:1504.08083*, 2015.

[44] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[45] B. Gromov, L. Gambardella, and A. Giusti. Video: Landing a drone with pointing gestures. In *HRI '18 Companion: 2018 ACM/IEEE International Conference on Human-Robot Interaction Companion, March 5–8, 2018, Chicago, IL, USA*. ACM, 2018.

[46] J. P. Hanna and P. Stone. Grounded action transformation for robot learning in simulation. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[47] K. Higuchi, Y. Ishiguro, and J. Rekimoto. Flying eyes: free-space content creation using autonomous aerial vehicles. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*, pages 561–570. ACM, 2011.

[48] G.-S. J. Hsu and J.-W. Huang. A photographer robot with multiview face detector. In *2016 IEEE International Conference on Industrial Technology (ICIT)*, pages 2152–2156. IEEE, 2016.

[49] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *IEEE CVPR*, 2017.

[50] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR*, 2017.

[51] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[52] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[53] N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *European Conference on Artificial Life*, pages 704–720. Springer, 1995.

[54] S. Jegou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio. The one hundred layers tiramisu: Fully convolutional DenseNets for semantic segmentation. *arXiv preprint arXiv:1611.09326*, 2017.

[55] N. Joubert, M. Roberts, A. Truong, F. Berthouzoz, and P. Hanrahan. An interactive tool for designing quadrotor camera shots. *ACM Transactions on Graphics (TOG)*, 34(6):238, 2015.

[56] V. Kazemi and J. Sullivan. One millisecond face alignment with an ensemble of regression trees. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1867–1874, 2014.

[57] A. G. Kendall, N. N. Salvapantula, and K. A. Stol. On-board object tracking control of a quadcopter with monocular vision. In *2014 international conference on unmanned aircraft systems (ICUAS)*, pages 404–411. IEEE, 2014.

[58] J. Kim and D. H. Shim. A vision-based target tracking control system of a quadrotor by using a tablet computer. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1165–1172, 2013.

[59] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[60] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 225–234, 2007.

[61] L. Kneip, M. Chli, and R. Y. Siegwart. Robust real-time visual odometry with a single camera and an imu. In *Proceedings of the British Machine Vision Conference 2011*. British Machine Vision Association, 2011.

[62] N. P. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IROS*, volume 4, pages 2149–2154. Citeseer, 2004.

[63] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[64] K. Lan and K. Sekiyama. Autonomous robot photographer with kl divergence optimization of image composition and human facial direction. *Robotics and Autonomous Systems*, 111:132–144, 2019.

[65] Z. Lan, M. Shridhar, D. Hsu, and S. Zhao. Xpose: Reinventing user interaction with flying cameras. In *Robotics: Science and Systems*, 2017.

[66] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4), 1989.

[67] C. Leger. *Darwin2K: An evolutionary approach to automated design for robotics*, volume 574. Springer Science & Business Media, 2012.

[68] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[69] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature pyramid networks for object detection. In *IEEE CVPR*, 2017.

[70] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*, pages 740–755. Springer, 2014.

[71] Q. Lindsey, D. Mellinger, and V. Kumar. Construction of cubic structures with quadrotor teams. *Proc. Robotics: Science & Systems VII*, 2011.

[72] K. Liu, D. Sakamoto, M. Inami, and T. Igarashi. Roboshop: multi-layered sketching interface for robot housework assignment and management. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 647–656. ACM, 2011.

[73] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single shot multibox detector. In *European Conference on Computer Vision (ECCV)*, pages 21–37. Springer, 2016.

[74] M. R. Loghmani, B. Caputo, and M. Vincze. Recognizing objects in-the-wild: Where do we stand? In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[75] S. Lupashin, A. Schöllig, M. Sherback, and R. D'Andrea. A simple learning strategy for high-speed quadrocopter multi-flips. In *2010 IEEE international conference on robotics and automation*, pages 1642–1648. IEEE, 2010.

[76] L. Ma and L. L. Cheng. Studies of ar drone on gesture control. In *2016 3rd International Conference on Materials Engineering, Manufacturing Technology and Control*. Atlantis Press, 2016.

[77] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525. IEEE, 2011.

[78] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[79] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[80] F. Mueller and M. Muirhead. Jogging with a quadcopter. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 2023–2032. ACM, 2015.

[81] M. Müller, S. Lupashin, and R. D'Andrea. Quadrocopter ball juggling. In *2011 IEEE/RSJ international conference on Intelligent Robots and Systems*, pages 5113–5120. IEEE, 2011.

[82] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[83] T. Nägeli, L. Meier, A. Domahidi, J. Alonso-Mora, and O. Hilliges. Real-time planning for automated multi-view drone cinematography. *ACM Transactions on Graphics (TOG)*, 36(4):132, 2017.

[84] M. Obaid, F. Kistler, G. Kasparavičiūtė, A. E. Yantaç, and M. Fjeld. How would you gesture navigate a drone?: a user-centered approach to control a drone. In *Proceedings of the 20th International Academic Mindtrek Conference*, pages 113–121. ACM, 2016.

[85] Parrot. Drone, 2019.

[86] N. Passalis and A. Tefas. Deep reinforcement learning for frontal view person shooting using drones. In *2018 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS)*, pages 1–8, 2018.

[87] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[88] H. X. Pham, H. M. La, D. Feil-Seifer, and L. V. Nguyen. Autonomous uav navigation using reinforcement learning. *arXiv preprint arXiv:1801.05086*, 2018.

[89] S. Pillai and J. Leonard. Monocular slam supported object recognition. *arXiv preprint arXiv:1506.01732*, 2015.

[90] S. Piskorski, N. Brulez, P. Eline, and F. D'Haeyer. AR Drone Developer Guide, 2012.

[91] R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, and A. Cangelosi. Autonomous quadrotor landing using deep reinforcement learning. *arXiv preprint arXiv:1709.03339*, 2017.

[92] D. A. Pomerleau. *Neural Network Perception for Mobile Robot Guidance*. Kluwer Academic Press, 1992.

[93] V. L. Popov, K. B. Shiev, A. V. Topalov, N. G. Shakev, and S. A. Ahmed. Control of the flight of a small quadrotor using gestural interface. In *Intelligent Systems (IS), 2016 IEEE 8th International Conference on*, pages 622–628. IEEE, 2016.

[94] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[95] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016.

[96] J. Redmon and A. Farhadi. YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[97] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[98] D. Richards, T. Patten, R. Fitch, D. Ball, and S. Sukkarieh. User interface and coverage planner for agricultural robotics. In *Proceedings of ARAA Australasian Conference on Robotics and Automation (ACRA)*, 2015.

[99] P. M. Roth and M. Winter. Survey of appearance-based methods for object recognition. *Inst. for Computer Graphics and Vision, Graz University of Technology, Austria, Technical Report ICGTR0108 (ICG-TR-01/08)*, 2008.

[100] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. LabelMe: a database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1):157–173, 2008.

[101] I. Sa and P. Corke. *Vertical Infrastructure Inspection Using a Quadcopter and Shared Autonomy Control*, pages 219–232. Springer Berlin Heidelberg, 2014.

[102] D. Sakamoto, K. Honda, M. Inami, and T. Igarashi. Sketch and Run: a stroke-based interface for home robots. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 197–200, 2009.

[103] D. Scaramuzza, M. C. Achtelik, L. Doitsidis, F. Friedrich, E. Kosmatopoulos, A. Martinelli, M. W. Achtelik, M. Chli, S. Chatzichristofis, L. Kneip, et al. Vision-controlled micro flying robots: from system design to autonomous navigation and mapping in gps-denied environments. *IEEE Robotics & Automation Magazine*, 21(3):26–40, 2014.

[104] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[105] Z. Shen, Z. Liu, J. Li, Y.-G. Jiang, Y. Chen, and X. Xue. DSOD: Learning deeply supervised object detectors from scratch. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[106] A. Shrivastava, R. Sukthankar, J. Malik, and A. Gupta. Beyond skip connections: Top-down modulation for object detection. *arXiv preprint arXiv:1612.06851*, 2016.

[107] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, 2014.

[108] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[109] W. D. Smart, C. M. Grimm, M. Dixon, and Z. Byers. (not) interacting with a robot photographer. In *Proceedings of the AAAI Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments*, 2003.

[110] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE CVPR*, 2015.

[111] C. Szegedy, S. Reed, D. Erhan, D. Anguelov, and S. Ioffe. Scalable, high-quality object detection. *arXiv preprint arXiv:1412.1441*, 2014.

[112] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[113] A. Tayebi and S. McGilvray. Attitude stabilization of a vtol quadrotor aircraft. *IEEE Transactions on control systems technology*, 14(3):562–571, 2006.

[114] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[115] R. G. Valenti, Y.-D. Jian, K. Ni, and J. Xiao. An autonomous flyer photographer. In *2016 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 273–278. IEEE, 2016.

[116] P. Viola, M. Jones, et al. Rapid object detection using a boosted cascade of simple features. *CVPR (1)*, 1(511-518):3, 2001.

[117] H. Voos. Nonlinear state-dependent riccati equation control of a quadrotor uav. In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, pages 2547–2552. IEEE, 2006.

[118] H. Voos. Nonlinear and neural network-based control of a small four-rotor aerial robot. In *2007 IEEE/ASME international conference on advanced intelligent mechatronics*, pages 1–6. IEEE, 2007.

[119] O. Walker, F. Vanegas, F. Gonzalez, and S. Koenig. A deep reinforcement learning framework for uav navigation in indoor environments. In *2019 IEEE Aerospace Conference*, pages 1–14. IEEE, 2019.

[120] S. Wang, R. Clark, H. Wen, and N. Trigoni. DeepVO: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[121] S. Yang, P. Luo, C.-C. Loy, and X. Tang. Wider face: A face detection benchmark. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5525–5533, 2016.