2015

Autonomous Quadcopter Videographer

Quiquia Rey Coaguila
University of Central Florida

AUTONOMOUS QUADCOPTER VIDEOGRAPHER

by

REY R. COAGUILA
B.S. Universidad Peruana de Ciencias Aplicadas, 2009

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2015

Major Professor: Gita R. Sukthankar

ABSTRACT

In recent years, the interest in quadcopters as a robotics platform for autonomous photography has increased. This is due to their small size and mobility, which allow them to reach places that are difficult or even impossible for humans. This thesis focuses on the design of an autonomous quadcopter videographer, i.e. a quadcopter capable of capturing good footage of a specific subject. In order to obtain this footage, the system needs to choose appropriate vantage points and control the quadcopter. Skilled human videographers can easily spot good filming locations where the subject and its actions can be seen clearly in the resulting video footage, but translating this knowledge to a robot can be complex. We present an autonomous system implemented on a commercially available quadcopter that achieves this using only the monocular information and an accelerometer. Our system has two vantage point selection strategies: 1) a reactive approach, which moves the robot to a fixed location with respect to the human and 2) the combination of the reactive approach and a POMDP planner that considers the target's movement intentions. We compare the behavior of these two approaches under different target movement scenarios. The results show that the POMDP planner obtains more stable footage with less quadcopter motion.

A mis padres que me apoyan en todo. Gracias por sus enseñanzas y por su cariño. Gracias por todo, son los mejores.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Gita Sukthankar. Her guidance was crucial in all the aspects of this project, and beyond it. I would also like to thank Dr. Rahul Sukthankar, who guided me during the development of this work.

I would like to thank my friends at UCF and my lab mates at the Intelligent Agents Lab. The day to day life in Orlando was great thanks to all the wonderful people I met here. Special thanks to Astrid Jackson and Gonzalo Cucho, who were my test subjects for this project, and were always there to lend me an ear.

Last but not least, thanks to the Fulbright Program that sponsored my studies in the US and organized several events where I could met so many great people. In short, the Fulbright experience was simply amazing!

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

The "flying selfie bot" has emerged as an important commercial application of quadcopters, due to the voracious consumer demand for high quality photos and videos to share on social media platforms [1]. Although most efforts target outdoor and sports photography, quadcopters can be valuable for indoor photographers as well. For instance, Srikanth et al. [2] demonstrated the utility of quadcopters at providing rim illumination of a moving subject.

To make the best use out of its limited battery life, an autonomous system must solve the same optimization problem faced by a lazy wedding videographer who seeks to produce a steady stream of good footage, with minimal effort, commemorating the event. After the wedding, the videographer gets paid extra for every photo and video purchased by the guests. Photos of the bride, groom, and key members of the wedding party are likely to be highly requested, along with significant moments such as the couple's first kiss. A practiced human photographer is skilled at finding the best *vantage points* from which to capture the scene.

This thesis tackles a constrained version of the vantage point selection problem in which the aim is to produce a stream of high-quality photos of a single moving subject over a short time period with minimal quadcopter motion. We evaluate two approaches implemented on a Parrot AR.Drone:

- A reactive system that tracks the subject's head pose using PD control (Proportional / Derivative)
- Coupling the reactive system with a movement policy produced by a POMDP (Partially Observable Markov Decision Process) solver that considers the subject's movement intentions.

Head pose is estimated in real-time using a face detector system to calculate a bounding box, along

with the facial yaw. The distance between facial landmarks is then used to extract the 3D location of the face relative to the robot's camera. We demonstrate that adding a POMDP planner to the system generates more stable footage. In contrast, the PD controller must constantly respond to the subject's vacillations, resulting in unnecessary motion and less centered photos.

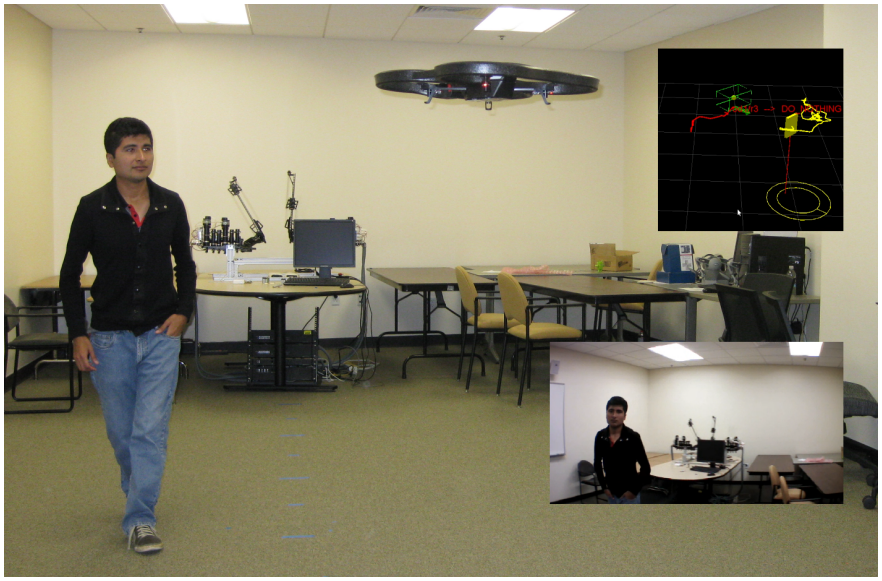


Figure 1.1: The system in action. The figure shows the quadcopter that is moving autonomously to keep track of the human. The sub windows show 1) the model of the world that the controller maintains to make its decisions and 2) the view from the quadcopter camera.

1.1 Related Work

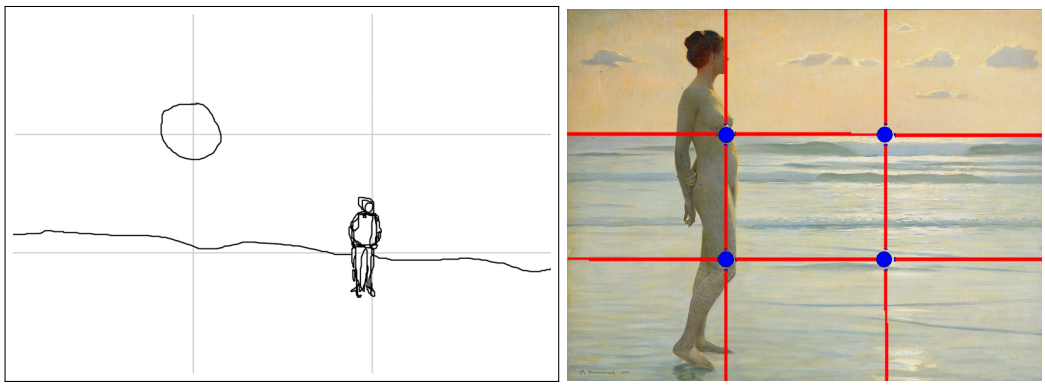
The vantage point selection problem is common to different areas of computer science. For example, in computer graphics, given a 3D mesh of an object, the *viewpoint selection* [3–5] problem focuses on finding a view or set of views that accomplish certain goal, such as covering the most surface possible, making the object visually appealing, or facilitating understanding. In 3D virtual world games and cinematography, this task is even more important because selecting a good view-

point increases the players/viewers immersion. Thus, emphasis has been placed on creating artistic results, maximizing scene understanding, and trying to convey emotions by choosing appropriate vantage points [6]. This has resulted in virtual cinematographers that use different techniques, such as finite state machines [7], genetic algorithms [8], or neural networks [9].

A common characteristic of the approaches mentioned above is that they require full information of the model and state of the world, i.e. the 3D model of the object and scene in computer graphics, or the position of characters and cinematography directives in 3D virtual world games. In our case, we have a stream of images coming from the quadcopter, and we would like to optimize their quality by changing the vantage point without further information. This is related to several composition rules that are used in photography to dictate the quality of images. Computer vision researchers have studied how to automate this process of *aesthetic quality assessment* [10–14]. A common approach in this area is to capture a set of features from images that were previously labeled as aesthetic / unaesthetic or had their quality ranked by humans, and then using machine learning techniques to either train a classifier (aesthetic / unaesthetic) or a predictor that uses regression to estimate quality scores. Various features are used, some of which are specific to the type of image (e.g. human portraits), but the most common ones are exposure of light and colorfulness, saturations and hue, size, aspect ratios, region composition, and the rule of thirds. The rule of thirds is one of the most important in photography composition: it specifies that the main object of a figure should be located at the intersection of (or along) the imaginary lines that cut the figure into three parts horizontally and vertically (Figure 1.2).

There has also been research in autonomous photography from the robotics perspective. For example, Byers et al. [15] implemented an autonomous event photographer that followed this procedure to take candid snapshots of conference attendees using an iRobot B2lr mobile robot. Subjects were identified using a color-based skin detection model, and vantage points were selected using an objective function that included a combination of photograph features and reachability consid-

erations. The event photographer was programmed using the same composition rules commonly taught to human photographers, including the rule of thirds mentioned above. In contrast, Campbell and Padmanabhan [16] created an autonomous photographer that did not use content-based heuristics for image composition. Instead of relying on color models, a depth clustering process was used to locate subjects based on motion parallax. One advantage of this system is that it could photograph non-human subjects, assuming that they posed for the camera.



(a) (b)

Figure 1.2: Rule of thirds. This composition rule specifies that the main interest point of a picture should be located on the intersection of the lines that divide the picture in three parts vertically or horizontally, as in (a), or along those lines, as in (b). (a) by CIngre via Wikimedia Commons/ CC BY-SA 3, (b) By Max Nonnenbruch [Public domain] via Wikimedia Commons

Finally, research on micro-aerial vehicles (MAVs), and especially on quadcopters, represents a related area that is worth mentioning. In recent years, interest in these vehicles has increased because of their ability to actuate in constrained and dangerous environments, and move freely in the 3D space. First work on this area focused on the development of adequate controllers [17–19] to aid the pilots with taking off, landing, and hovering. Autonomous navigation and simultaneous localization and mapping (SLAM) are also areas that have been researched extensively [20–23]

using different types of sensors like laser range finders, depth-RGB data or purely monocular information.

The *Joggobot* [24], a jogging companion drone, is an example of an autonomous quadcopter that performs human tracking. This one requires the subject to wear a special t-shirt for localization. Higuchi et al. [25] developed a system that allows the Parrot AR.Drone to act like a free camera that follows a person, inspired by the modern camera systems used in sport events. The person to be followed must wear a distinctive color suit, which is then tracked using a particle filter. Then, the system calculates the distance to the subject in order to follow or circle around him. The *FollowMe* system performs hands-free filming with a quadcopter; for this application, Naseer et al. [26] demonstrated autonomous human-following using an on-board depth camera. The FollowMe human tracker recovers full body pose from warped depth images, which are also used for gesture recognition. As mentioned before, Srikanth et al. [2] presented an autonomous photographer assistant system with a quadcopter that moves to maintain rim illumination of a moving subject.

1.2 Outline

The rest of this document is organized as follows:

- Chapter 2 describes the platform that was used for the project. This includes a general introduction to quadcopters and how they work, the details of the Parrot AR.Drone, and the Robot Operating System (ROS).
- Chapter 3 presents the computer vision algorithms that were used in order to process the images and obtain the human location. This includes face detection, face alignment and face pose estimation. Also, the method to transform the results of these algorithms to 3D world

measurements (location and distances) is explained in detail.

- Chapter 4 describes the theory of PD controllers and the details of our reactive controller.
- Chapter 5 describes Markov Decision Processes (MDPs) and Partially Observable MDPs (POMDPs), which were used in this project to model the intention of humans. Then, it explains in detail the POMDP model that we created for this task.
- Chapter 6 describes the implementation of our project and how all the pieces connect to create the whole system.
- Chapter 7 explain the experiments we performed to test our methods and the results that were obtained.
- Chapter 8 presents the conclusion of this thesis.

CHAPTER 2: PLATFORM

This chapter presents the platform that was used in this project. It starts with the basic theory behind quadcopter flight, as well as some details about the Parrot AR.Drone, which was selected as our hardware platform. Finally, we also describe the *Robot Operating System (ROS)* framework, which was employed for our project.

2.1 Quadcopter Flight

A quadcopter, also called quadrotor or quadrocopter, is a helicopter with four propellers. These propellers allow the drone to keep its position in 3D space and move in any desired direction. The position is maintained by rotating one set of opposing propellers clockwise, while rotating the other pair counter-clockwise (Figure 2.1(a)). This allows for the cancellation of the total torque in the vehicle and with enough thrust, the nullification of gravity.

To move in free space, the robot can modify the velocity of its propellers as follows:

- By increasing the velocity of the four propellers by the same amount and thus generating more overall thrust without creating torque, the copter is able to maneuver in a vertical line (along the z-axis) (Figure 2.1(b)).
- Horizontal movement is achieved by increasing the velocity of one propeller and by decreasing the velocity of the opposite propeller (Figure 2.1(c)). The effect of this action is a change in the roll and pitch angle which leans the robot and moves it horizontally. Depending on which propeller is accelerated, the robot moves in either the x or y direction.
- Finally, to change the direction of the robot (its yaw angle), two opposite propellers are ac-

celerated (Figure 2.1(d)). This creates a torque in the robot and thus an angular acceleration that changes the yaw angle.

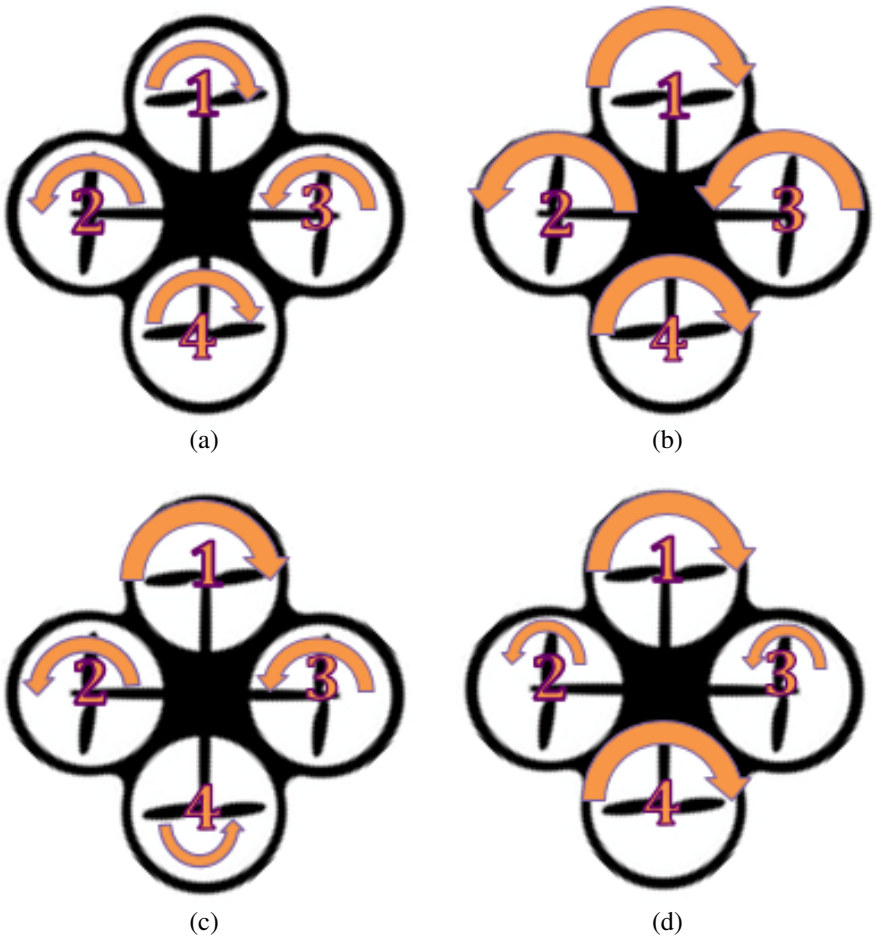


Figure 2.1: Basic Quadcopter Flight: The size of the arrows represents the amount of rotational speed (bigger arrows represent higher speeds). (a) Maintain position, (b) Move up, (c) Move forward, (d) Rotate clockwise.

2.2 Parrot AR.Drone Hardware and Software

The Parrot AR.Drone is a commercial quadcopter which was presented at the *International Consumer Electronics Show (CES)* in 2010. It is targeted for end-users interested in flying a quadcopter using a mobile application available for iOS and Android. While flying the robot, the users may choose to play two augmented reality games developed by Parrot. Also, the company released an API in order to encourage third-party developers to create their own applications. This makes the AR.Drone an interesting option for research in robotics and mobile vehicles [27].

AR.Drone 2.0, the next generation of the quadcopter, was released in 2012. This version includes some improvements in the hardware and is the one used in the experiments. The following section presents some specific details on the hardware of the AR.Drone 2.0 and its API.

2.2.1 Hardware of the AR.Drone 2.0

(a) Indoor hull (b) Outdoor hull

Figure 2.2: AR.Drone 2.0 and its different hull options

The Parrot AR.Drone comes equipped with 2 hulls, which are possible body protections as can be seen in Figure 2.2. The indoor hull protects the quadcopter from collisions with other objects, which can happen easily when flying in cluttered environments. The outdoor hull does not provide

protection for the propellers, but its smaller weight allows for a more stable flight and better maneuverability. The total weight of the quadcopter is 420 grams with the indoor hull and 380 grams when the outdoor hull is used. The size of the AR.Drone is 51.7 cm $\times$ 51.7 cm when using the indoor hull, and 45.1 cm $\times$ 45.1 cm with the outdoor one.

The AR.Drone is equipped with two cameras. A frontal camera, which produces an HD video stream (720p-30 fps) with wide angle lenses (92° diagonal), can be configured to stream at 360p (640 $\times$ 360) or 720p (1280 $\times$ 720). One problem that arises is the radial distortion that occurs due to the fish eye lens. Furthermore, the fast movement of the drone (especially when rotating), produces a strong motion blur and a linear distortion caused by the camera's rolling shutter [28].

The second camera is a QVGA (320 $\times$ 240) 60fps camera (64° lens), which points downwards. This camera is not severely affected by radial distortions, motion blur, or rolling shutter effects. Therefore, the AR.Drone's processor uses this camera to estimate the horizontal displacements and velocities.

In order to sense its location and movements, the AR.Drone is equipped with an ultrasound altimeter, a pressure sensor, a 3-axis accelerometer, a 2-axis gyroscope, and a 1-axis yaw-precision gyroscope. The altimeter and the pressure sensor are used to estimate the current altitude of the drone. The first one can only be used for altitudes of up to 6 meters, while the pressure sensor can be used at any height (although it's more noisy). The 2-axis gyroscope measures the roll and pitch angles, with an overall deviation of up to 0.5° which is not subject to drift over time. The yaw gyroscope measurements, on the other hand, can be noisy and accumulate drift of up to 60° per minute. The quadcopter propulsion is performed with four brushless engines, each controlled by a micro-controller. As a measure of security (to prevent repeated shocks), the engines are stopped inmediately if any of the controllers detect a propeller blockage. 1000mAh, 11.1V LiPo batteries, allow for approximately 10 minutes of continuous flight.

Finally, the AR.Drone executes its main computations with an 1GHz ARM Cortex A8 processor with 1Gbit DDR2 RAM. This processor runs an operating system based on GNU/Linux. The system also includes a USB port that can be used to connect a GPS module or to store videos when using the mobile applications. All communications between the drone and the outside world are performed via WiFi. For this, the AR.Drone creates a wireless LAN network as soon as the battery is connected.

2.2.2 Software of the AR.Drone 2.0

As mentioned previously, the AR.Drone comes with a set of mobile applications that allow end-users to fly the drone and play augmented reality games. The flight application can be seen in Figure 2.3.

Figure 2.3: AR.Flight application screenshot

A great part of the drone control is performed on-board (in the drone's processor). However, this software is not accessible by the user. Developers can perform some basic communications via telnet, but no documentation of the control software (which is not open-source) is provided and trying to access this information manually may damage the drone.

Therefore, the best way to communicate with the robot is via the communication channels set up by the drone's wireless network. These channels are:

- The navigation channel (UDP port 5554): The drone shares its basic navigational data, called navdata, via this channel. Some of the information that is accessible here is:
  - The drone's roll, pitch, and yaw (from the gyroscopes)
  - The horizontal velocity, estimated with the use of an on-board optical flow algorithm, utilizing the down-facing camera's video stream. It is important to note that the accuracy of this estimate depends heavily on the ground below the quadcopter. If the floor that the camera is observing is a textured surface the results are good. However, without textures in the surface, the results are poor, with up to 1 m/s of deviation from the true value. [4]
  - The drone height, obtained from the ultrasound sensor. The readings from this sensor depend on a flat surface since uneven or non-reflecting ground (as well as flying next to walls) can rapidly distort the results and cause the drone to oscillate in the vertical axis. If the ground is ideal, the ultrasound sensor achieves errors with a mean of only 8.5 cm at a height of 1 m.
  - Battery percentage
  - Internal timestamp, which is given in microseconds. This represents the moment when the data was sent from the quadcopter.
  - Internal state. A 32-bit bit field that indicates the robot's status. These states include, for example, LANDING, HOVERING, ERROR, TAKEOFF, and others.
  - The video channel (UDP port 5555): The drone continuously transmits a video-stream coming from the cameras. Two possible videos can be obtained; one for each camera (frontal and down-facing).

It's important to note that all videos are transferred with a lower frame rate and smaller resolution than the maximum permitted by the cameras because of the bandwidth limits imposed by the WiFi. Also, these videos are encoded using a proprietary format, which is explained in detail in the drone's documentation [29].

- The command channel (UDP port 5556): The drone listens to this channel waiting for the developer to send parameters used by the drone to fly. The available parameters are:
  - Desired roll and pitch angles, yaw rotational speed, and vertical (z-axis) speed. These values must be between -1 and 1.
  - A bit that represents the hover-mode. If set to true, the AR.Drone tries to keep its position ignoring all other control commands.
  - A bit that represents the emergency-mode. If set to true, the drone enters emergency mode and switchs off all engines.
  - A bit that represents the take-off and landing commands.
- An optional control port (TCP port 5559). This channel is reserved for changing specific parameters such as the desired video channel. The complete list of parameters is detailed in the quadcopter's documentation [29].

It is worth mentioning that the first three channels are UDP-type channels. This means that it is possible that packets can get lost or arrive out of order.

To facilitate the development of games using the AR.Drone, Parrot published an API that simplifies the communication with the robot. This API, provided in ANSI C, is documented in [3], and takes care of all the communication channels and the callbacks associated with the reception of video frames and navigational data. It also permits the encoding and sending of control commands.

The present project uses the *ardrone_autonomy* ROS package[1], which is built on top of the API and presents an interface for coding autonomous behavior in the ROS framework, abstracting the communication with the AR.Drone and the usage of their API. Specifically, this package allows us to receive callbacks with the navigation information from the quadcopter, as well as the images from the camera's view, all in standard ROS messages. Additionally, it also allow us to send simple *twist* commands to the drone, consisting of the linear velocity in 3D space (v_x, v_y, v_z) and the angular (yaw) velocity $\dot{\Psi}$.

2.3 Robot Operating System (ROS)

The Robot Operating System (ROS) is an open-source framework for robot software development. It provides the tools needed for hardware abstraction, low-level device control, useful common functionality, multiple processes and communication between them, and package management. This project is built using ROS as the backbone architecture for communication. This section presents a summary of the main concepts in ROS. For the full details of the framework, we refer the reader to their documentation [30].

2.3.1 ROS Filesystem Level

In terms of the files and folder structure of ROS resources, the central concepts to understand are the following:

- Packages: A package is the main unit for organizing software in ROS. It may contain ROS runtime processes, which are called nodes, libraries, dataset, configuration files, etc. It also

[1] `https://github.com/AutonomyLab/ardrone_autonomy`

includes the package manifest, which is a file that provides the metadata of the package, including its name, version, description, license information, and dependencies.

- Stacks: A collection of packages that provides aggregate functionality. Stacks are how ROS software is released and have associated version numbers. Whereas the goal of a package is to be a minimal unit for software that is easy to reuse, the goal of a stack is to simplify the process of sharing.

- Message (msg) types: These are files that describe a personalized message to be used in ROS. The definition includes the data structures that comprise the message.

- Service (srv) types: These are files that describe a service in ROS. Specifically, it defines the service request and response.

2.3.2 ROS Computation Graph Level

The Computation Graph is the network of ROS processes that work together to process data. The main concepts in this graph are:

- Nodes: A node is a process that performs computation. The ROS framework is designed to be modular and thus, the manipulation of a robot can involve hundreds of nodes that perform different tasks. For example, we may have nodes that perform specific tasks in computer vision, others that process the navigation information and localization, and so on.

- Master: This is a centralized entity that keeps track of all the Computation Graphs. For example, nodes use the Master to look up other nodes, send messages, etc.

- Parameter server: A part of the master that centralizes the parameters to be used by nodes or other entities.

- Messages: Nodes use messages to communicate between themselves. They can be simple messages that are built-in (e.g. booleans, integers, floats, strings) or can be defined by the developer as more complex structures including multiple types (and even arrays).

- Topics: When a node wants to communicate with other nodes, it publishes information to a topic, which is the title of the conversation. The nodes that want to listen to the information on that topic must subscribe to it. This architecture allows for multiple publishers and subscribers.

- Services: This is a second type of communication. While topics represent a many-to-many global communication, services are a solution for a simple request/reply model, where a node wants to ask for an operation and the server responds only to that node.

- Bags: This is a format to store any kind of ROS message data. It is a useful mechanism for performing tests and debugging with little effort, since entire runs with all their messages can be stored in bags and then be replayed several times.

As mentioned previously, we used the *ardrone_autonomy* package in ROS that allows for easy communication with the Parrot AR.Drone. Other packages that were used in our project are:

- APPL: Contains a node that connects the system to the APPL[2] implementation of the SARSOP algorithm to solve POMDPs. The POMDP file and its solution (the policy file) must have been trained previously with the APPL software and be loaded in the package. By doing this, the node can receive observations and output the actions according to the trained model.

- TF: This package facilitates the tracking of multiple coordinate frames over time. Since robots usually have different joints and are moving constantly, transforming from one coor-

[2] `http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/`

dinate to another at different times can be a challenging problem. This package solves this by maintaining a buffered tree-like structure that relates all the coordinate transforms.

- Rviz: This package provides easy 3D visualization tools for ROS. To be able to use this package, we can define basic primitives in our nodes such as where to place markers, and lines, and then use Rviz to visualize them in 3D.

CHAPTER 3: FACE LOCALIZATION

The principal component for localizing the quadcopter in our system is the location of the human subject. Specifically, we used the location of the person's face and its rotation (yaw) to plan the motions to execute. However, in order to get this information we rely only on the frontal camera of the quadcopter, thus using monocular images. This problem has been extensively researched in computer vision, leading to specific algorithms for tasks such as *face detection* and *face pose estimation*.

This chapter explains the details of the algorithms used to get the face information. First, we explain the face detection and alignment process, which analyzes images and finds the location of faces and the pixels that correspond to principal landmarks such as the nose and the eyes. Then, the pose estimation problem is described, which focuses on finding the position of a detected face (the yaw, pitch, and roll angles). Finally, we explain the localization process, which will estimate real world distances from the camera to the detected subjects, using the pixel location information and the pose angles.

3.1 Face Detection and Alignment

This section describes the face detection and face alignment problems, as well as the algorithms that were used in our project to solve these tasks. Specifically, to detect the subject's face in the feed obtained from the front-facing camera, we employ a fast detector based on the recent work on max-margin object detector [31], which uses Histograms of Oriented Gradient [32] (HOG) features in conjunction with structural SVMs, as made available in the Dlib open-source library [33].

Given the bounding box of a detected face, we use the "millisecond face alignment" method re-

cently proposed by Kazemi & Sullivan [34] to recover the locations of five key landmarks on the subject's face: the far corners of the eyes and mouth, and the tip of the nose. For this, we also used the implementation available in the Dlib library.

3.1.1 Object Detection and Histograms of Oriented Gradients

Object detection and tracking has attracted considerable attention in the computer vision community [35–38] due to its applicability to several different problems. In this task, there is a specific class of objects that we are interested in locating in an image or set of images. For example, pedestrian detection, which has been a principal focus in this area, deals with the location of people in open environments. This kind of detection is difficult because of the high appearance variability that humans can have in normal situations. Some of the factors that cause this variability are:

- Humans are very different one from another. They can have different height, weight, skin color and other physical characteristics that make the detection difficult. There is also change in the clothes and accessories a human can wear.
- Human can have different poses, caused by the joints in the body. These joints represent degrees of freedom that increase all the possible human poses, even for a single subject.
- The background in different photos can be different (e.g. streets, rooms) which represents noise for the detection. The illumination in different environments can affect the process too.
- The viewpoint affects how the human is seen. Different viewpoints such as a frontal view from the ground and a view from 2 meters above the ground to the right side of the human can have very different results.
- Occlusions may occur between different humans or between objects and humans, impeding the full view of the body.

Several algorithms have been developed that consider some or all of these aspects. One of the most popular methods is the Histogram of Oriented Gradients (HOG) [32], which was created initially for pedestrian detection. In our project, we are interested in face detection, and this method has been successfully extended to work with faces [38].

HOGs are a kind of *feature descriptor*. The goal of these descriptors is to capture the characteristics of an object, so that we can then generalize it and detect other objects of the same type. Given an image or patch of an image, its HOG descriptor is obtained by:

- Calculating the gradient at each pixel. The authors did experiments with different masks to calculate the derivatives and several smoothing scales, and the best results were found when no smoothing and a single 1-D centered mask [-1 0 1] was used. In the case of color images, the gradients are calculated for each channel and the one with the highest norm is selected at each pixel.

- Pixels are grouped into cells (with an usual size of 8 x 8 pixels) and a weighted histogram is calculated for each cell. Each histogram has 9 bins corresponding to the unsigned gradient directions (0°-180°). The weight of each vote corresponds to the gradient magnitude. To reduce aliasing, each vote is interpolated bilinearly between the corresponding bin and its neighbors.

- Cells are grouped into blocks (with an usual size of 2 x 2 cells), which are overlapped. Thus, in the case of 2 x 2 blocks, each cell appears in up to four different blocks. Each block, which is transformed in as single column vector, is normalized using:

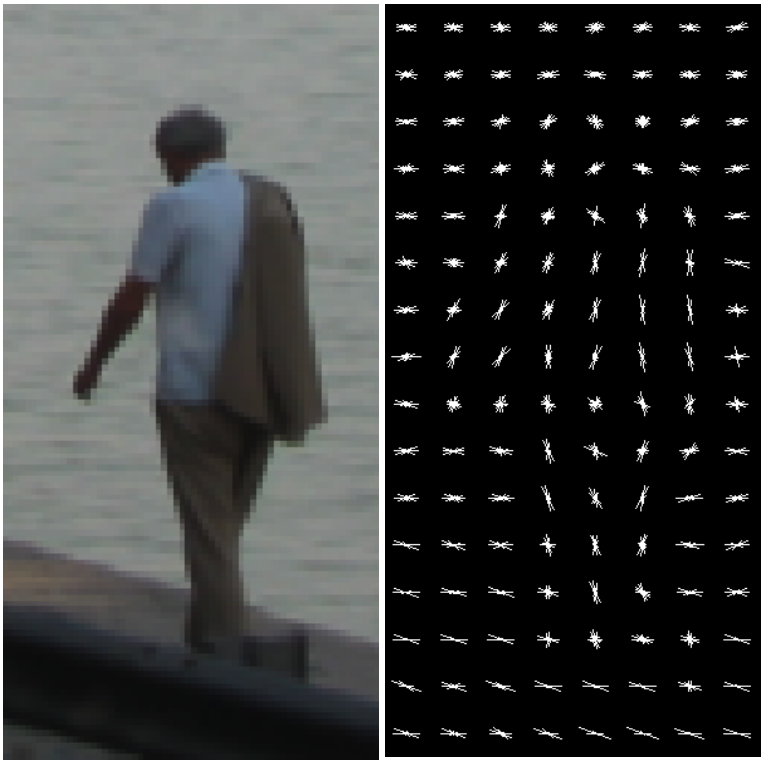
$$v' = \frac{v}{\sqrt{\|v\|_2^2 + \epsilon^2}} \tag{3.1}$$

  Where v is the unnormalized vector obtained after transforming the block and v' is the nor-

malized vector. ϵ is a small constant used for regularization; the authors found that the results are insensitive to the value of ϵ over a large range.

- The normalized vectors for all the blocks are concatenated in a single vector, which corresponds to the HOG descriptor.

Figure 3.1 shows an example of an HOG and the image it was obtained from. In this case, each block's histogram is shown separately.



(a) Example figure (b) HOG descriptor of the figure

Figure 3.1: Example of an HOG descriptor

To create a detector using this method, we first need a training set with positive (images of the object) and negative (images without the object) examples. The detector consists of a linear window

scoring function specified by a parameter vector w. Ideally, if $\phi(x, r)$ corresponds to the HOG feature of a sliding window r in image x, then the dot product $w \cdot \phi(x, r)$ is positive if the window corresponds to a human face, and negative otherwise.

In order to obtain w, this method uses a Support Vector Machine (SVM) that is trained with the positive and negative examples, previously preprocessing the images to obtain their HOG descriptors. The negative examples in the training set are usually obtained by sampling random patches from a set of pictures that do not contain the object, and searching for false positives (patches that were classified as the object by the detector after one initial training). These kinds of images represent "hard examples" that improve the quality of the training phase. Once enough of these hard negative examples are obtained, the training process is run again to obtain the final detector.

Finally, once w is obtained, the detection process is performed in each possible subwindow of a given testing image. For each subwindow, its HOG is obtained and multiplied by w, and if the result is positive the patch is classified as a detection. If several occurrences of the object are detected, a subsequent non-maximum suppression step is performed, which analyzes overlapping windows marked as positive and preserves only those that are maximum and do not overlap. To determine if two regions overlap with each other, it is common to consider the ratio of the intersection area over the total area (union of both regions). If the ratio is less than 0.5, then the regions do not overlap.

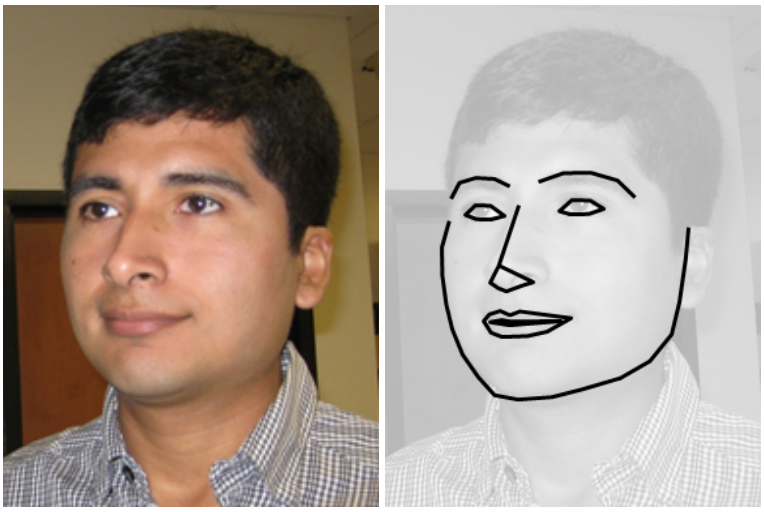
3.1.2 Max-Margin Object Detection

The HOG method described above is usually trained using a set of positive examples and negative examples, where the negative ones are sampled from a separate dataset of images not containing the object to detect. This is done because of computational reasons since the total number of subwindows in any picture can be very large, but at the same time the sampling may lead to a suboptimal detector [31]. Also, the fact that the negative images are sampled from a different set does

not permit analysis of the boundaries of the detected objects (subwindows partially overlapping an object) which are usually false alarms.

The Max-Margin Object Detection (MMOD) method [31] considers these limitations by optimizing over all sub-windows instead of using sampling. This is equivalent to improving the accuracy of a detector that works with complete images, instead of first training a binary detector that focuses on patches and then using it to obtain an overall detector. In order to make this computationally feasible, the method uses structured SVMs and a convex optimization algorithm, which results in faster training times in comparison to other state-of-the-art methods. Also, in terms of performance in the face detection task, MMODs get better results that other recent methods such as the deformable part models [39].

A face detector based on this method and trained with the LFW dataset [40] is available in the Dlib open-source library [33]. This is the implementation we used for our project.



(a) (b)

Figure 3.2: Face alignment: (a) Original figure where a face has been detected (b) Landmarks identified by the face alignment process.

3.1.3 Face Alignment

Face alignment is the process of identifying key landmarks in a detected face. As example of its results can be seen in Figure 3.2, where the edges in the second picture are defined by the pixel locations of the landmarks.

For our project, we used the method by Kazemi and Sullivan [34], which uses an ensemble of regression trees to estimate the shape from an initial configuration. Specifically, they used a cascade of regressors, where each one predicts an update vector from the image and the current shape estimate. So, given the current shape $\hat{\mathrm{S}} = (\mathrm{x}_1^T, \mathrm{x}_2^T, ..., \mathrm{x}_p^T)^T \in \mathbb{R}^{2p}$ represented as a vector containing the 2D pixel locations x_i of each one of the ith landmarks in the image, the update process is performed by:

$$\hat{\mathrm{S}}^{(t+1)} = \hat{\mathrm{S}}^{(t)} + r_t(I, \hat{\mathrm{S}}^{(t)}) \tag{3.2}$$

where r_t is the tth regressor in the cascade. This update improves the estimate, and the process is repeated for each regressor in the cascade. In order to train the regressors, their method uses a gradient tree boosting algorithm [41] with a sum of square error loss. Labelled data is required to perform this process.

An important factor in the algorithm is that the regressors make their predictions based on features, such as intensity values computed from the image I, that are not indexed in the global coordinate of the image, but relative to the current shape estimate $\hat{\mathrm{S}}$. This helps diminish the high variability in the features caused by different illuminations and shape deformations.

This method is available in the Dlib library, where it was pretrained with the 300-W faces in-the-wild dataset [42]. The resulting shape detector provides 68 landmarks that include points in the eyes, eyebrows, nose and mouth. Figure 3.2(b) was obtained using this particular implementation on a sample image.

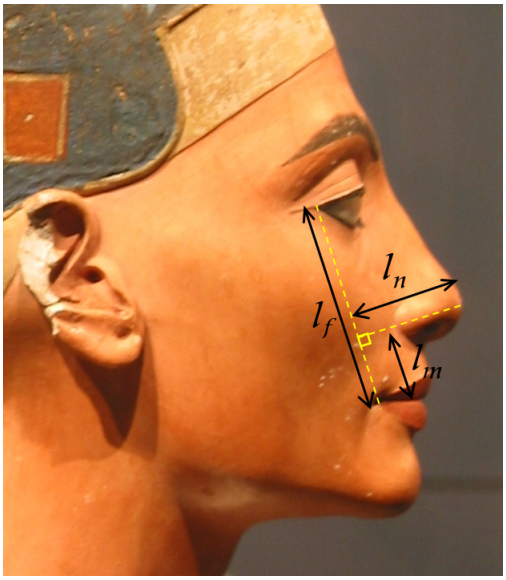
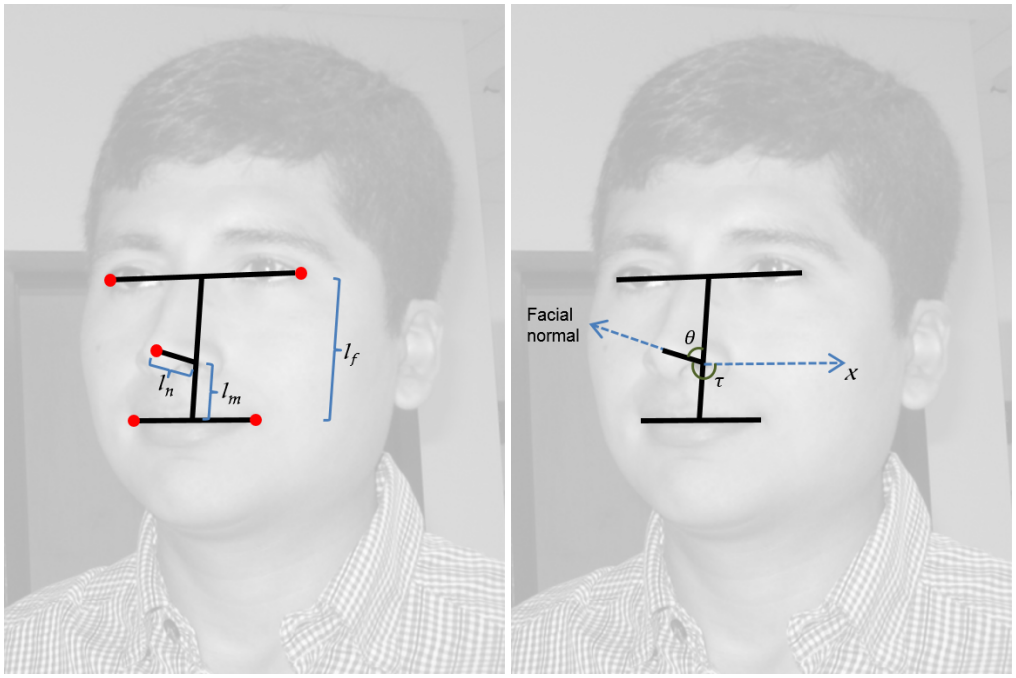


Figure 3.3: Distances used in the 3D facial model: the distances l_f, l_n, and l_m are the projection of the world distances L_f, L_n, L_m. Image adapted from "Nefertiti Bust" by Magnus Manske via Wikimedia Commons / CC BY-SA 3

3.2 Face Pose Estimation

The head pose estimation problem focuses on obtaining the orientation of a human head from images (which can be monocular or contain depth information). It has been tackled extensively in the computer vision community and different approaches currently exist. The survey by Murphy et al. [43] collects some of the recent work in the area and divides it into different categories depending on the main approach used. For our project, we used the work by Gee and Cipolla [44] that is categorized as a geometric approach since it uses the location of features in the 2D image to calculate the pose. Their work presents two different methods to calculate the pose; the first one is a 3D method that uses weak perspective geometry and the position of 5 landmarks that can be seen in Figure 3.4(a), and the second one exploits the skew-symmetry of the facial plane. Next, we explain the first method, which was the one used in our system.



(a) (b)

Figure 3.4: Face pose estimation: (a) The landmarks used in the 3D method are shown as red dots, as well as the distances used in the calculation of the pose (b) Angles used in the method. The facial normal is determined by the nose tip, and our method calculates the slant angle σ to obtain the normal.

The 3D facial model is based on the ratios of three real-world distances L_f, L_n, and L_m, and their projections in the image l_f, l_n, and l_m (Figure 3.3). Specifically, we need the value of the ratios $R_m = L_m/L_f$ and $R_n = L_n/L_f$. In the original paper, the authors used $R_m = 0.4$ and $R_n = 0.6$, and in our experiments we found that these values work well for different faces. This is due to the fact that ratios tend to be the same for different people, even though the sizes may vary.

From the image, we calculate the values of l_f, l_n, and l_m by considering the obtained landmarks, as shown in Figure 3.4(a). The distance l_f is calculated in the symmetry axis that passes through the midpoint between the far corners of the eyes and the midpoint of the mouth. l_m is calculated

by using the ratio R_m and the obtained l_f. l_n is calculated as the distance from the tip of the nose to its base (extreme of l_m). The method additionally requires the angles θ and τ shown in Figure 3.4(b).

With this information, we calculate the slant angle σ, i.e. the angle between the optical axis and the facial normal in 3D space, by using:

$$m_1 = \left(\frac{l_n}{l_f}\right)^2 \tag{3.3}$$

$$m_2 = \cos^2(\theta) \tag{3.4}$$

$$\hat{d_z^2} = \frac{R_n^2 - m_1 - 2m_2R_n^2 + \sqrt{(m_1 - R_n^2)^2 + 4m_1m_2R_n^2}}{2(1-m_2)R_n^2} \tag{3.5}$$

$$\sigma = \cos^{-1}|\hat{d}_z| \tag{3.6}$$

Then, we can calculate the facial normal:

$$\hat{\mathrm{n}} = [\sin\sigma\cos\tau, \sin\sigma\sin\tau, -\cos\sigma] \tag{3.7}$$

from which we obtain the yaw and pitch angles of the subject's face.

3.3 Face Location Estimation

The previously described methods allow us to detect images in a figure and their corresponding yaw estimates, but they don't give us any information of the pose in the 3D real world. In order to get this information, we used basic perspective geometry and some assumptions on the size of the face. This section explains this process.

First, we assume that the focal length f of the camera (in pixels) can be obtained through calibra-

tion. We also assume that the distance D between the two far corners of the eyes is known; we find $D = 9$ cm to be a good estimate for adult subjects. Next, we obtain the pixel location (x, y) of the midpoint between the subject's eyes as well as the measured distance d_m (in pixels) of the projection of D in the image.

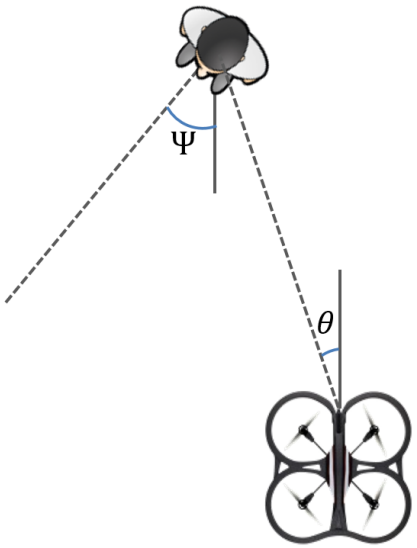


Figure 3.5: Face Location Estimation: Angles used in the calculation

The depth, Z, can be estimated from the measured distance between the subject's eyes in the image, after compensating for the foreshortening caused by the subject's head yaw (Ψ) and the azimuth (θ) corresponding to the subject's horizontal displacement, as illustrated in Figure 3.5.

$$Z = \frac{fD}{d_m} \cdot \frac{\cos^2(\Psi)}{\cos(\theta)}, \tag{3.8}$$

where Ψ is obtained from the face alignment and the azimuth θ given by:

$$\theta = \tan^{-1}\left(\frac{X}{Z}\right) = \tan^{-1}\left(\frac{x}{f}\right) \tag{3.9}$$

Once Z has been estimated, obtaining X and Y is straightforward: under perspective projection, a point in 3D space (X, Y, Z) is related to its corresponding 2D location in the image (x, y) by:

$$X = x \cdot \frac{Z}{f} \tag{3.10}$$

$$Y = y \cdot \frac{Z}{f} \tag{3.11}$$

Finally, we perform a correction on the yaw since faces on the extremes (left or right) of the figure that are looking to the front seem to be looking to the sides from the quadcopter's view. Also, we need to normalize the yaw angle to the quadcopter's frame of reference. To account for all this, we define a corrected yaw:

$$\Psi^* = \pi - (\Psi + \theta) \tag{3.12}$$

which is used as the final estimate of the yaw of the subject.

CHAPTER 4: CONTROL

Control theory characterizes dynamical systems with inputs and how to modify their behaviors with feedback which is managed by a controller. The main goal of this area is to control a system in such a way that its output signal, the *measured output*, matches a *reference* signal. The controller performs this task by sending commands (feedback) to the system that depends on the measured error, i.e. the difference between the system's output and the reference. Figure 4.1 shows an schematic representation of this process.

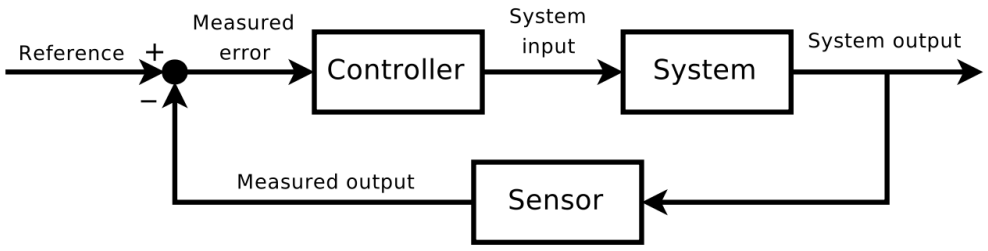


Figure 4.1: Feedback loop in control theory, via Wikimedia Commons / CC BY-SA 3

This chapter presents PID controllers (Proportional-Integral-Derivative Controllers), which are a basic tool to calculate commands. Also, we describe the specific details of the controller we implemented for our system.

4.1 PID controllers

PID controllers are a mechanism to calculate the commands to give to a system in the feedback loop. For this, they use the formula:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t) \tag{4.1}$$

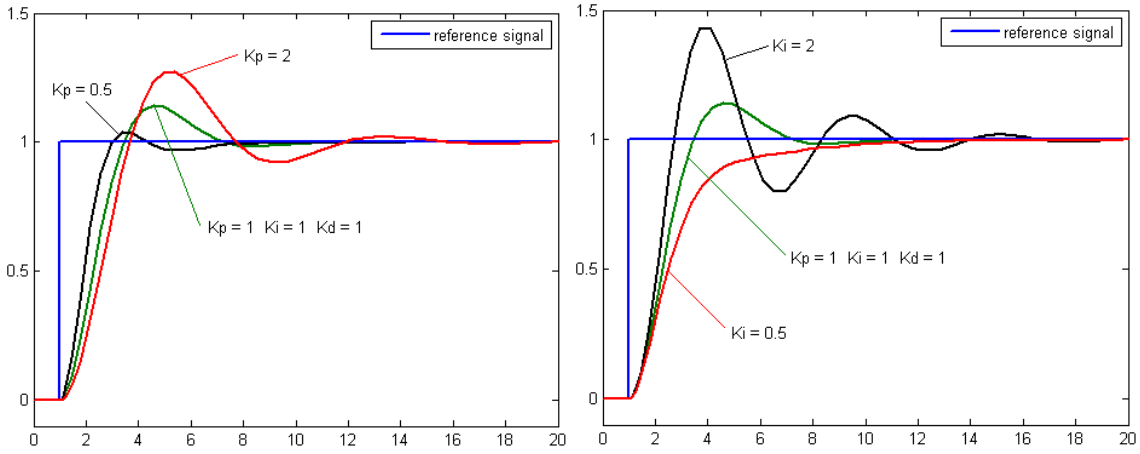
This method uses three terms:

- Proportional term: Given by $K_p e(t)$, this term is proportional to the current error $e(t)$, which is the difference between the output measurement of the system and the reference. It is the main term in the controller that reacts immediately to changes in the error. The magnitude of K_p affects the velocity of convergence, as can be seen in 4.2(a). Higher values result in faster changes in the output, but this may lead to oscillations (and divergent behaviors in the worst case). Smaller values result in less sensible controllers that may take too long to reach the desired reference.

- Integral term: Given by $K_i \int_0^t e(\tau)d\tau$, this term is proportional to both the magnitude and the duration of the error. It is useful to take into account residual steady-state errors that may exist in the system (e.g. compensating against the wind for outdoor quadcopters). However, since its value considers all the past error, it may cause overshooting, so careful tuning is required. In practice, the integral term is usually changed to a summation when the measurements are discrete.

- Derivative term: Given by $K_d \frac{d}{dt} e(t)$, this term is proportional to the change in error (its derivative). It is used to decrease the oscillations in the system by decreasing the magnitude of the commands when the output signal is close the reference and the error is already going

down. An example of this is a mobile robot that is already moving towards its goal; if it is close to it, the system should not accelerate anymore (it may even start deaccelerating).



(a) Effect of K_p

(b) Effect of K_i

(c) Effect of K_d

Figure 4.2: Effect of the terms of a PID Controller. Each figure shows how the system output changes with different values for the controller parameters. In each figure, one parameter is changed while the others are kept constant. Via Wikimedia Commons / CC BY-SA 3

It is common to use only some of the terms in this type of controller depending on the behavior of the system. Common variations are P (only the proportional term), PI (proportional and integral),

and PD (proportional and derivative) controllers. For our project, we are using a PD controller since steady errors were negligible in our experiments. This is explained in the next section.

4.2 Reactive Behavior for Quadcopter Autonomous Navigation

Assuming that the pose (3D coordinates and yaw) of the subject's face is known in some coordinate system, we can use a PD controller to perform a reactive behavior that moves the quadcopter to a certain goal. In our project, the localization is handled by the face detection subsystem, as explained in Chapter 6, so this section focuses on the details of the PD controller and how the reactive behavior works.

The goal location, which the reactive controller will try to maintain even when the subject is moving, is defined as a point with coordinates (x_g, y_g, z_g) centered in front of the subject's face. This is a desirable vantage point from which the quadcopter can obtain frontal photos. This point is determined by two variables:

- d_{front}: the frontal distance between the goal location and the subject face.
- d_{height}: the altitude of the robot with respect to the eyes of the detected face. A value of $d_{\text{height}} = 0$ will maintain the robot at the same level as the eyes of the subject.

Specifically, if the robot location is given by (x_r, y_r, z_r) with a yaw Ψ_r and the human location is (x_f, y_f, z_f) with yaw Ψ_f, the goal for the robot is calculated as:

$$x_g = x_f + \cos(\Psi_f) \times d_{\text{front}} \tag{4.2}$$

$$y_g = y_f + \sin(\Psi_f) \times d_{\text{front}} \tag{4.3}$$

$$z_g = x_f + d_{\text{height}} \tag{4.4}$$

and the error e for each component is:

$$e_x = (x_g - x_r) \times \cos(\Psi_r) + (y_g - y_r) \times \sin(\Psi_r) \tag{4.5}$$

$$e_y = (x_g - x_r) \times -\sin(\Psi_r) + (y_g - y_r) \times \cos(\Psi_r) \tag{4.6}$$

$$e_z = z_g - z_r \tag{4.7}$$

which is the vector between the goal and the robot as seen from the robot's frame of reference.

The reactive behavior tries to maintain the robot in the goal location, facing towards the human. In order to do this, a PD controller is used for the X and Y components (frontal and side movements of the robot in an horizontal plane). This controller uses the error for X and Y mentioned above for the proportional term. The derivative term uses the velocity of the drone obtained from the navigation information (also in the robot's frame of reference). The Z component is managed by a simpler P controller, which uses only the proportional term (fed with e_z).

Finally, the yaw component is calculated differently. Although our intention is that the robot faces the human subject once it is in the goal location, considering only the final yaw may cause the robot to lose the sight of the person. For this reason, we instead try to maintain the robot always pointing towards the human subject. For example, consider the case in Figure 4.3, the command for the robot will be to rotate towards its left so that it keeps seeing the human. If this logic is maintained through the quadcopter flight motion, it will be facing the subject at its destination.

The simplicity of the reactive behavior allows for fast reaction times when the person starts moving, e.g., walking around a room or turning, but the responsiveness comes at a price. Not only is the resulting footage jerkier due to excessive quadcopter motion but it is also possible for the quadcopter to be frequently out of position when the subject turns away and then back, because it responds immediately (with large motions) to small changes in head orientation. In order to

overcome these issues, we designed a POMDP model that considers the human intentions, and it is explained in the next chapter.

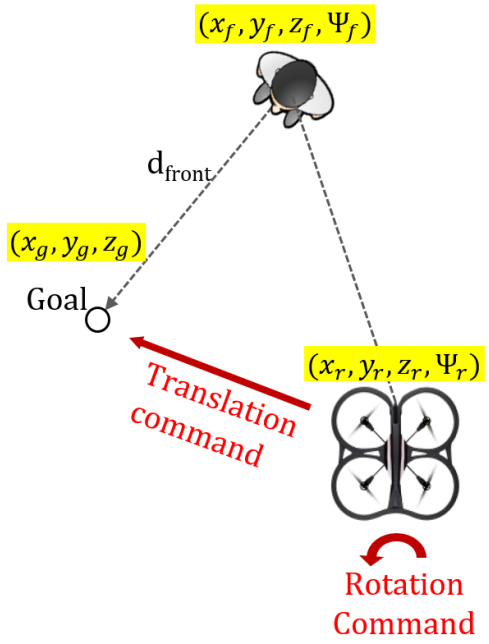


Figure 4.3: Reactive Behavior: The goal location is calculated as d_{front} meters in front of the person. The height of the goal is d_{height} meters above the location of the eyes.

CHAPTER 5: PLANNING USING POMDPS

One of the goals of this project is to create a controller capable of considering the intentions of the subject when planning the motions of the quadcopter. For example, if the person in front of the drone is carrying out short movements but is mostly maintaining its center position (i.e. its intention is to maintain it), the controller should not send commands to the robot. This will minimize the motion of the camera and produce more stable footage. In the other hand, if the intention of the human is to move to a far location, or make a large face rotation, the quadcopter should correct its position to avoid losing sight of the person.

The PD controller described in the previous chapter always reacts to the subject's motion without considering his intention. In order to improve this controller, we used a Partially Observable Markov Decision Process model (POMDP), which maintains a belief of the subject's intention and generates a macro-action. This model works as an extra-layer in the system, so the macro-action will then be processed by the PD-controller if a motion is required.

This chapter explains the theory behind Markov Decision Processes (MDPs) and POMDPs, and how to solve them. Then, it also details the model used in this project, which we named the *Intention Aware Videographer Problem.*

5.1 Markov Decision Processes

Planning under uncertainty is an important task in robotics, given that we can only take measurements up to some precision, and the results of our actions are not deterministic. In this sense, MDPs are a framework that allows us to partially solve this problem, assuming that the current state is known (completely observed) but the transitions generated by our actions are stochastic.

Formally, a Markov Decision Process is defined as a 5-tuple:

$$(S, A, T, R, \gamma)$$

where:

- S is a finite set of states
- A is a finite set of actions
- T is a transition model that defines the probability that executing action a in state s at time t will lead to state s' at time $t + 1$:

$$T(s, a, s') = P(s'|s, a) \tag{5.1}$$

- $R(s, a)$ is the reward function, which depends on the current state s and action a.
- $\gamma \in [0, 1]$ is the discount factor that weights the importance of present rewards versus future rewards.

At each time step t, the agent (the controller) is in one state s_t, which is known, and must choose an action to execute a_t. This will change the current state of the agent according to the transition model, and yield a reward r_t. The main problem of a MDP is to find an optimal *policy* π that indicates the actions to execute in specific states such that the total expected reward:

$$\mathrm{E}\left(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right) \tag{5.2}$$

is maximized.

A policy $\pi : S \rightarrow A$ is then a mapping from the states in S to actions A. The process of solving an MDP consists of obtaining an optimal π, i.e. a policy that produces the highest expected reward.

5.2 Partially Observable MDPs

An important assumption in MDPs is that the states are completely observed. However, in practice this is not the case for several applications, especially in robotics where we are limited by the precision of different sensors. Partially Observable Markov Decision Processes relax this assumption by considering that the agent cannot observe the state directly, but instead it receives stochastic *observations* that depend on its sensors. Specifically, this model has the same components as an MDP, but adds a set of observations O and an observation model:

$$Z(s, a, o) = P(o|s, a) \tag{5.3}$$

which details the probability of getting observation o, if the agent is currently in state s and has executed action a. Figure 5.1 shows an standard representation of POMDP models and how the different components depend stochastically on the others.

Solving a POMDP model will yield optimal action policies, but these are different than in MDP case. This is due to the fact that we can no longer map the current state, which is unknown, to actions. Instead, POMDP policies maintain belief states $b(s)$, which are probability distributions over the state space S. Specifically, a belief b can be represented by an $|S|$ dimensional vector where each entry indicates the probability that the agent is an specific state. The policies then are mappings from the belief space (all such possible vectors) to actions ($\pi : B \rightarrow A$).

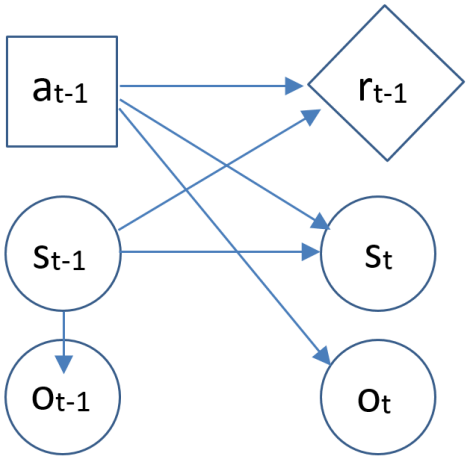


Figure 5.1: Standard representation of a POMDP. Nodes represent variables: states(s), observations(o), actions (a), and rewards (r). The edges represent the conditional probabilities, e.g. the state s_t is conditionally dependent on the previous state s_{t-1} and the action executed a_{t-1}.

Due to the fact that the size of the belief space B is exponentially bigger than the state space S, computing an exact solution is infeasible for any large POMDP. However, many approximation algorithms have been proposed based on the idea of sampling from the belief state, instead of visiting all possible beliefs. These are called *point-based algorithms*, and their main characteristic is that they represent the belief space with a representative sample of points. Then, these algorithms use *value iteration* [45] to improve over an initial policy, exploring the belief space and executing *backup operations* over the sample points. This process is repeated until convergence or until it runs out of time. In order to be more efficient, these algorithms sample from the reachable space $R(b_0)$ of a given initial belief b_0, and not from all the belief space.

HSVI2 [46] and SARSOP [47] are two of the state-of-the-art methods to solve POMDPs, and both use a point-based approach. The first uses heuristics to focus the search in areas where the gap between the lower and upper bound of the optimal value function is smaller. SARSOP, on the other hand, explicitly attempts to sample from the optimally reachable space $R^*(b_0)$ (the belief

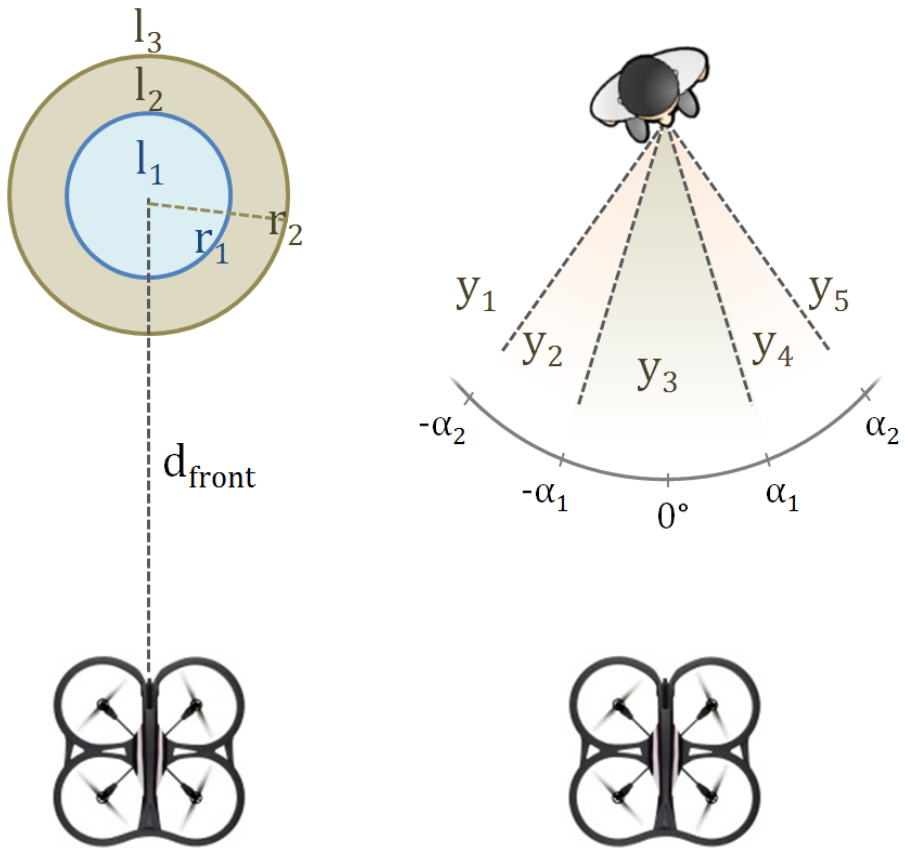
states reachable from b_0 under *optimal* sequences of actions). For this, the algorithm performs an heuristic exploration of $R(b_0)$ and improves its sampling through on-line learning. It also uses bounding techniques to avoid regions that are not likely to be optimal, thus focusing on $R^*(b_0)$.

5.3 Intention Aware Quadcopter Videographer Problem

This section explains the POMDP model we developed for our system. The goal of this POMDP is to make the agent (the controller) more judicious in its selection of vantage points by avoiding unnecessary quadcopter motion. A key observation is that a subject may make many small motions near a location punctuated by occasional larger motions between locations. To model this, we predict the *intention* of the subject (either maintaining or changing pose) and learn a "lazier" controller that executes significant quadcopter motion only when merited. We modeled the person's intention as a partially observed variable since we know that it is not observed in the world, but the controller does observe the subject motion and can estimate its intention from it.

In our model, the principal agent has a state consisting of the following partially observable variables:

- loc: This represents the location of the subject, modeled as the distance from the goal, and discretized into three variables: l_1, l_2 and l_3 (see Figure 5.2(a)). At each planning step, the goal location is calculated as d_{front} meters in front of the quadcopter. If the distance between this goal (considering only the x and y directions and not the height) and the subject is smaller than r_1, then the value for loc should be l_1. If it is between r_2 and r_3, its value should be l_2. If it is larger than r_3, it should be l_3. This discretization from the center of the goal location, as opposed to a 2D grid, allows for a simpler and smaller POMDP with fewer number of states.



(a) Location Model (b) Yaw Model

Figure 5.2: POMDP quadcopter world: the location is calculated by discretizing the distance from the goal, and the yaw is calculated by discretizing the yaw between -90°and 90°.

- yaw: The yaw is defined as the angle of the subject's face with respect to the quadcopter's front-facing camera. An angle of zero would mean that the human is facing parallel to the camera (see Figure 5.2(b)). These readings are discretized into 5 variables $y_1, \ldots, y_5$ defined by the limiting angles $-\alpha_2$, $-\alpha_1$, α_1 and α_2 ($0 < \alpha_1 < \alpha_2$).

- lint: Represents the "location intention" of the human to move away from its location or not. It can have the values *move* or *keep*.

- yint: Represents the intention of the human to rotate its main focus point (the yaw direction). It can have the values *front*, *rotate_left*, and *rotate_right*.

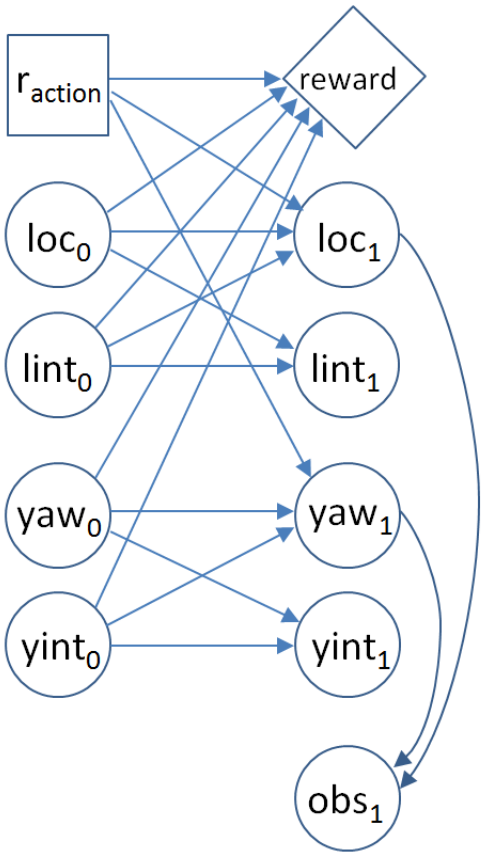


Figure 5.3: POMDP for the Intention Aware Quadcopter Videographer Problem

In each time step, the agent receives a noisy reading of the location and yaw of the subject. This observation is used to update the belief about the current state including the intentions that are not directly observed. For this, we model the transition probabilities as shown in Figure 5.3. In our model, the subject maintains its location and yaw direction around the safe zones (mostly in l_1 and y_3, but occasionally moving and returning from l_2, y_2 and y_4) when the intentions are *lint* = *keep* and *yint* = *front*. These intentions may change over time causing the human to move away (states

l_3, y_1, and y_5).

At each planning step, the agent may execute two different actions: *do_nothing* or *correct*. From the controller perspective, the first action avoids sending any command to the quadcopter, and second one calculates a new goal using the same steps as in the reactive behavior and then executes motion commands to achieve it. In our POMDP model, *do_nothing* does not change any state variable, while *correct* has the effect of moving the human closer to the goal (towards l_1 and y_3). The reason for this is to make the POMDP model account for the actions of the controller.

We modeled the rewards in the POMDP problem such that the *do_nothing* action is preferred when the subjects' intention is to maintain its pose and its yaw direction. On the other hand, the *correct* action is preferred when the robot location and yaw are in the extremes (l_3, y_1 and y_5) or when the intention has changed, such as when the robot remains in the middle zone too long (l_2, y_2 or y_4).

This model was solved offline using the SARSOP algorithm and an optimal policy was obtained. During the quadcopter's flight, we use this policy and the estimation of the subject's location to get a general action (*do_nothing* or *correct*) that will be later processed by the system. This process will be explained in detail in the following chapter.

CHAPTER 6: IMPLEMENTATION

In the previous chapters, we have described the platform and the theory behind the individual components of our system. This chapter ties everything together and explains our implementation, paying attention to how the components interact with each other. Also, we mention some implementation considerations that were useful for obtaining a good overall performance.

6.1 System Architecture

Figure 6.1 shows our system architecture. As mentioned before, we used *ROS* to handle the communication in our system, which basically consists of a *ROS package* with two *nodes*:

- Localizer Node: Tracks the current state of the quadcopter along with the subject's location and relative orientation. This information is then broadcasted so that other nodes, such as the controller, can use it.
- Controller Node: Selects vantage points to best capture frontal images of the subject as it moves and generates the appropriate motion commands. Internally, it may use the reactive behavior or the POMDP planner to calculate the vantage points. It is important to notice that a small head turn from the subject can require the quadcopter to sweep in a wide arc in order to photograph the subject from a frontal vantage point. The controller must take this into account as it moves.

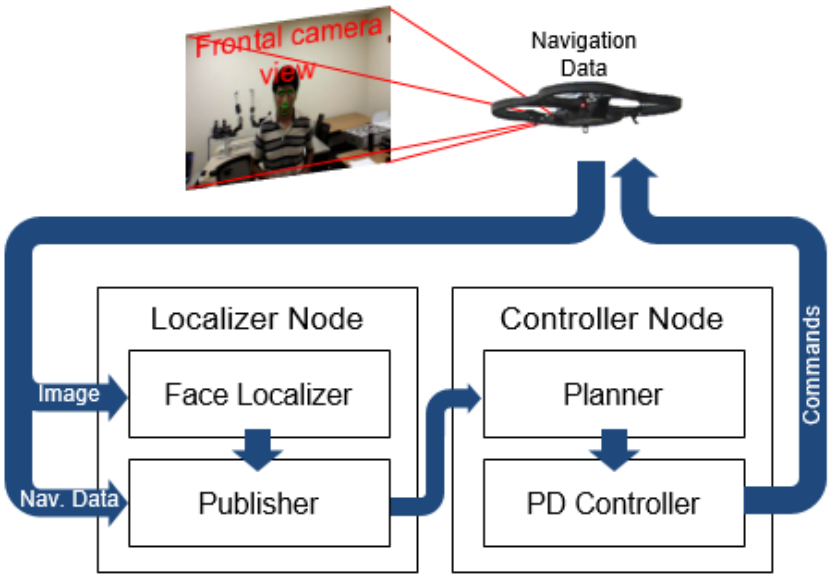


Figure 6.1: System Architecture

6.2 Face Localizer Node

The goal of this node is to process the image stream from the quadcopter's front-facing camera to detect faces and their poses (yaw). This information is used to locate the quadcopter and to create a model of the world (coordinates of the quadcopter and the subject) that will be broadcasted to other nodes. This process is explained in detail in this section.

Every time a new image is received (from the quadcopter's frontal camera), the face detection, alignment, and pose estimation processes are executed as detailed in Chapter 3. These tasks are performed in the face localizer sub module (Figure 6.1). A practical consideration is that the frontal camera in the AR.Drone exhibits a high degree of radial distortion, which if not corrected significantly degrades the localization estimates. The straightforward solution of warping each image to undo the distortion is computationally unattractive. Fortunately, we observe that the

detection and face alignment procedures are robust to image distortions, which allows us to avoid the expensive warp by performing those operations on the raw images. Then, we transform only the detected locations of the five facial landmarks to correct for radial distortion. The transformed locations of the five landmarks are fed into the pose estimation process to obtain the location estimates. This consideration enables us to obtain realtime processing without compromising the quality of our results.

The detected location and pose obtained by the face localizer are calculated with respect to the camera's frame of reference. However, this needs to be transformed to the robot's frame of reference before computing the control command. Also, since the quadcopter is tilted every time it moves horizontally, the location with respect to the camera may not be accurate. To account for this, we correct the calculated location by the roll and pitch angles that are provided by the quadcopter's gyroscope. While straightforward in concept, this process is complicated by the fact that the image stream and the navigation data are received at different rates and with different delays. We address this by executing an additional step to match the face location estimate to the most accurate quadcopter location available, inspired by the approach used in [23]. To do this, we maintain a queue of the estimated quadcopter locations over the last second, obtained from the navigation data, and when the face localizer returns an estimate of the face with respect to the camera, we use its timestamp to match the best available estimate of where the robot was when the picture was taken.

We also maintain an estimate of the drone's position with respect to its initial location. This is performed by integrating the (noisy) navigation information, which can result in an inaccurate global estimate of the quadcopter's location. To mitigate the effects of this noise on planning for vantage point selection, the planner operates in the current rather than the global reference frame. The complete state, consisting of location and 2D orientation (yaw) information for the robot and

subject (face) is summarized as:

$$(x_r, y_r, z_r, \Psi_r, x_f, y_f, z_f, \Psi_f).$$

6.3 Controller Node

The goal of the controller node is to obtain and send the commands to move the quadcopter, based on the state that is shared by the localizer node. Our system has two modes of operation:

- Reactive: The robot continuously calculates goal as a fixed point with respect to the face location. This goal is directly passed to the PD controller that will calculate commands to move towards it. The details of this process can be seen in Chapter 4.
- POMDP based: The robot will try to estimate the moving behavior of the human, as explained in Chapter 5. This will generate a macro action with two options: *do_nothing* and *correct*. If the action is *correct*, the PD controller is used to calculate the commands, otherwise no other calculations needed.

In our system, the controller node is composed of two sub modules: the planner and the PD controller. The planner is responsible for the high-level decision (the mode of operation) so if the user selects the POMDP based behavior, this module will obtain the macro-action from the model first and then decide if a movement is needed. If this is the case, or if the user selected the reactive behavior instead, the planner asks the PD controller to calculate the commands (using the same methodology in both cases). Then, the PD controller sends the commands to the drone once they are calculated. This is done by broadcasting the commands to the appropriate channels that are heard by the *ardrone_controller* node (part of the *ardrone_autonomy* package).

In order to work with our POMDP model, we used the Approximate POMDP Planning Toolkit (APPL).[1] We trained the model offline and then used the APPL ROS package[2] which allows us to execute the model (send observations and get actions). The planner sub-module uses this package to obtain the POMDP macro-actions.

An additional consideration is the frequency of update of the POMDP planner (how often do we send observations and get actions). Since this is a high-level planner, we don't need to update with the same frequency as the PD-controller. Also, the resulting actions should be given enough time to make some changes in the quadcopter's location, i.e. a full correction instead of partial movements. For these reasons, the planner, when using the POMDP behavior, updates the macro-action every 0.5 seconds, and the resulting macro-action is executed for the following 3 seconds.

[1] `http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/index.php?n=Main.HomePage`

[2] `http://wiki.ros.org/appl`

CHAPTER 7: EXPERIMENTAL RESULTS

This chapter presents the experiments performed in order to test our method, the metrics that were used, and the results obtained for the reactive controller and the POMDP-based one. A video showing the quadcopter being controlled by the system can be viewed at the following link: http://youtu.be/s6WJ6SaLZ1k.

7.1 Test Environment

Our physical testing environment was a room 7.5 by 4.0 meters long, with a height of 2.7 meters. We executed several runs, which consisted of five minutes of autonomous control of the quadcopter with a single subject in the room (who moves during the experiment). For each run, we first started the quadcopter (taking-off) manually, while the human was located in front of the drone at a distance between 1.5 and 2.0 meters. This served as an initialization so that the robot is positioned when the autonomous controller starts.

We consider three types of runs, characterized by the speed of the subject's movement:

- Scenario Type 1 - Slow Motion: In this case, the subject maintains his position for periods of 15 to 20 seconds, executing short and slow translations (less than 1 meter) and rotations in between.
- Scenario Type 2 - Medium Motion: The subject maintains his position for short periods of time, but now may execute small movements and rotations while doing so. The translations in between are longer (2 to 3 meters) and faster.
- Scenario Type 3 - Fast Motion: The human does not, in general, maintain position. He

moves around and rotates his face constantly with higher speed. Occasionally the subject will remain in certain locations for a longer period of time.

The goal of the quadcopter is to be positioned 1.6 meters in front of the human, and at the same height as the eyes. For the POMDP planner case, the values for r_1, r_2, α_1, α_2 are 0.4 meters, 0.6 meters, 15°, and 40°respectively.

7.2 Metrics

Two categories of metrics were obtained from each run. The first includes command-related metrics that measure the magnitude of the movements the quadcopter executed over time:

- Total Commands x/y/z: the sum of the absolute values of the linear velocity commands in three dimensions given to the quadcopter during a single run.
- Total Commands Yaw: the sum of the absolute values of the rotation commands (yaw) given to the quadcopter during a single run.

The second includes the face-quality metrics, obtained by analyzing the video recorded by the quadcopter:

- Face Size Error: This metric measures the difference in the size of the face in different frames, with respect to the desired size. The desired size is calculated as the mode of the detected face rectangles in the runs with slow motion. This area represents the size we would expect to see when the robot is precisely at the goal distance. Then, for each frame, we detect the face and calculate the ratio of its area vs. the desired area. If the ratio is smaller than one,

we use its inverse. This ratio then represents how different the area is with respect to the desired one. The total error for a run is the average ratio over all frames.

- Face Location Error: This corresponds to the average absolute value of the distance between the face (i.e., the center of the detection rectangle) and the frame center in pixels.
- Face yaw: Average absolute value of the yaw, where zero represents a face that is looking straight at the camera.

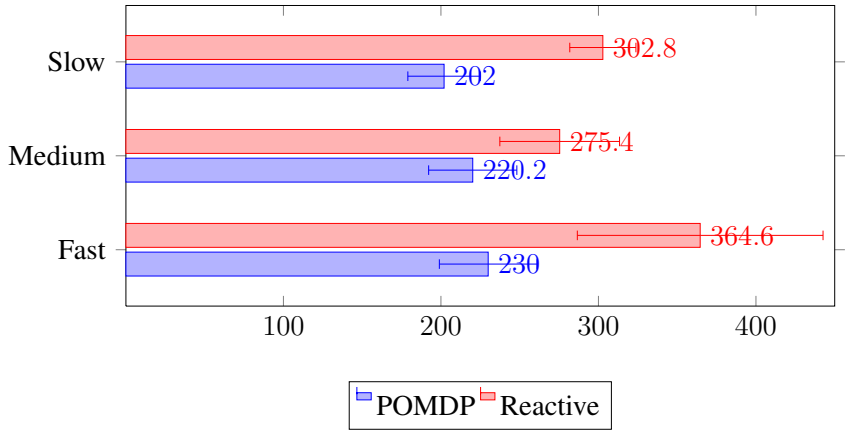
7.3 Results

As shown in Figure 7.1, the POMDP controller generally commands less quadcopter motion in comparison to the reactive behavior. As a result, the videos obtained when using the POMDP planner are more stable and do not shift as much. However, the tradeoff is that the subject may be closer or farther than the desired distance for short periods of time. Figure 7.2(a) shows that the face size error is slightly greater with the POMDP controller. This occurs because the POMDP waits longer while determining the subject's intention before executing actions.

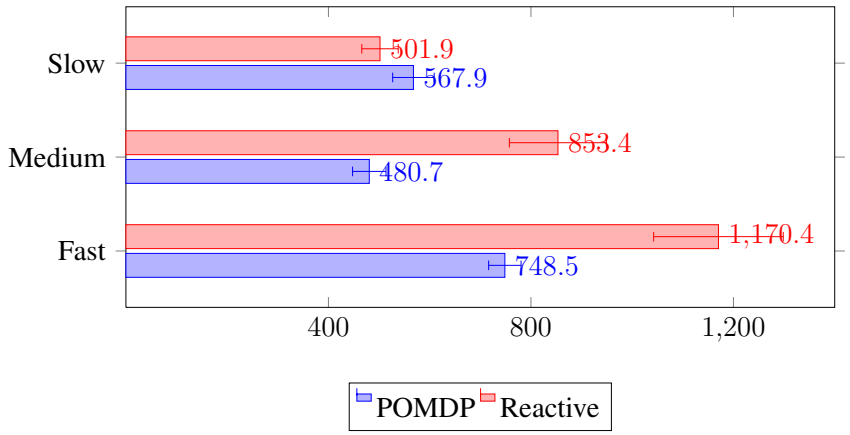
Figure 7.2(b) shows the error in location of the face, measured in pixels from the center of the image. The POMDP controller exhibits a smaller error for the medium and fast scenarios. This occurs because the subject may be farther than the goal, hence any horizontal movement corresponds to a smaller movement in the center of the detected face. Thus, even though we are allowing the subject to move further before reacting, the face will in average be closer to the center of the frame in comparison to the reactive behavior.

Finally, Figure 7.2(c) shows the error in the yaw, measured as the difference in the angle of the observed face vs. a frontal face (looking directly at the quadcopter). The POMDP controller achieves better results in the medium and fast scenarios. By oscillating constantly while trying to maintain

itself in front of the subject, the reactive behavior creates situations where the subject's head pose is worse. On the other hand, the POMDP filters small changes in the yaw so the average error is smaller.

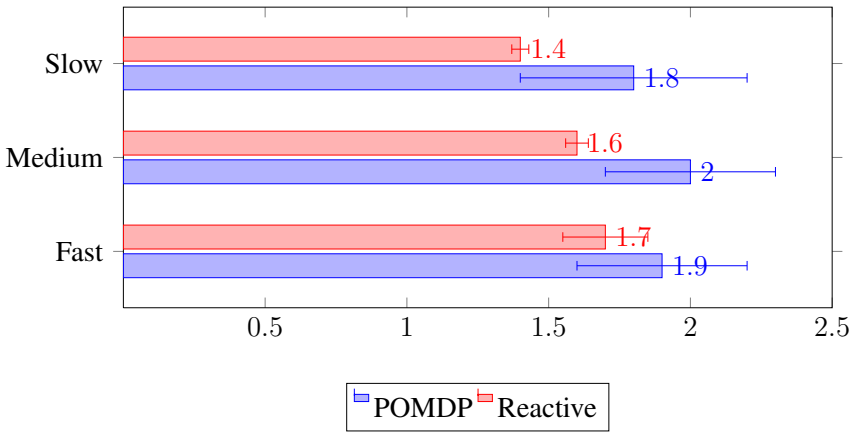


(a) Average Total Commands (x/y/z)

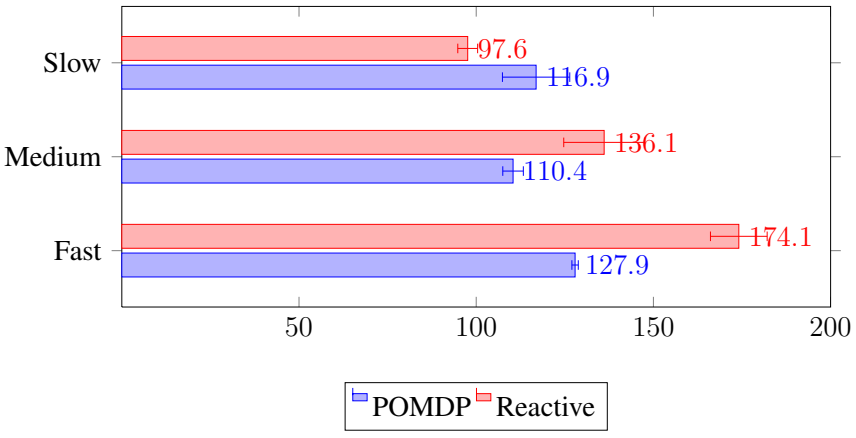


(b) Average Total Commands (yaw)

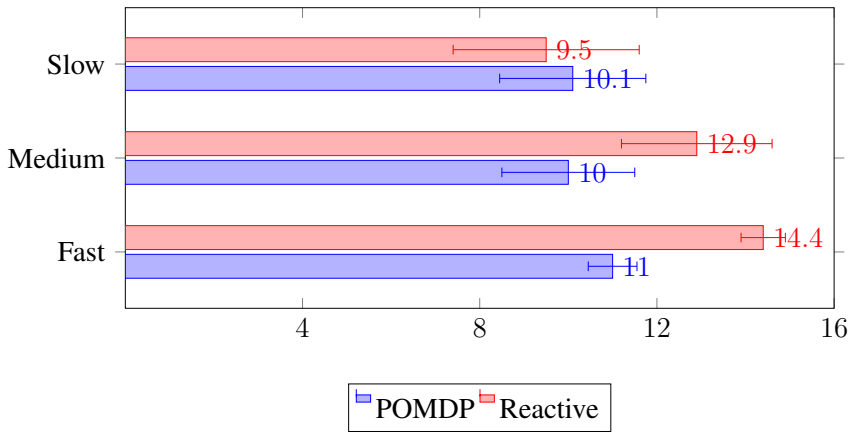
Figure 7.1: Total commands executed by the controllers under different scenarios. Smaller values indicate that the quadcopter moved less, resulting in more stable movies.



(a) Face Size Error



(b) Face Location Error



(c) Face Yaw Error

Figure 7.2: Face quality metrics measured as the average difference vs. the ideal size, location (the center of the image), and a frontal looking face.

CHAPTER 8: CONCLUSION AND FUTURE WORK

This thesis describes an autonomous quadcopter videographer that detects and tracks a human subject's face. Our solution primarily employs monocular information, which is obtained from the frontal camera and processed to estimate the subject's pose. We evaluate the performance of two vantage point selection strategies: 1) a PD controller that tracks the subject's head pose and 2) combining the reactive system with a POMDP planner that considers the subject's movement intentions. The POMDP is able to filter short motions and reacts only when the human moves farther or rotates more. As a result, this controller executes less motion, thus obtaining more stable video sequences than the PD controller alone.

In future work, we plan to tackle multi-person scenarios and create more complex image composition policies to shoot group videos. By introducing multi-frame evaluation metrics that consider events rather than static scenes, we can potentially improve the narrative structure of the video in addition to the visual aesthetics.

LIST OF REFERENCES

[1] D. Schneider, "Flying selfie bots," *IEEE Spectrum*, vol. 52, no. 1, pp. 49–51, 2015.

[2] M. Srikanth, K. Bala, and F. Durand, "Computational rim illumination with aerial robots," in *Proceedings of the Workshop on Computational Aesthetics*. ACM, 2014, pp. 57–66.

[3] P.-P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich, "Viewpoint selection using viewpoint entropy," in *Proceedings of the Vision Modeling and Visualization Conference (VMV)*, vol. 1, 2001, pp. 273–280.

[4] C. H. Lee, A. Varshney, and D. W. Jacobs, "Mesh saliency," in *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3. ACM, 2005, pp. 659–666.

[5] M. Feixas, M. Sbert, and F. González, "A unified information-theoretic framework for viewpoint selection and mesh saliency," *ACM Transactions on Applied Perception (TAP)*, vol. 6, no. 1, p. 1, 2009.

[6] A. Jhala and R. M. Young, "A discourse planning approach to cinematic camera control for narratives in virtual environments," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, vol. 5, 2005, pp. 307–312.

[7] D. B. Christianson, S. E. Anderson, L.-w. He, D. H. Salesin, D. S. Weld, and M. F. Cohen, "Declarative camera control for automatic cinematography," in *Proceedings of the National Conference on Artificial Intelligence (AAAI), Vol. 1*, 1996, pp. 148–155.

[8] N. Halper and P. Olivier, "Camplan: A camera planning agent," in *Smart Graphics 2000 AAAI Spring Symposium*, 2000, pp. 92–100.

[9] A. Hornung, G. Lakemeyer, and G. Trogemann, "An autonomous real-time camera agent for interactive narratives and games," in *Intelligent Virtual Agents*. Springer, 2003, pp. 236–243.

[10] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Studying aesthetics in photographic images using a computational approach," in *Proceedings of the European Conference on Computer Vision*. Springer, 2006, pp. 288–301.

[11] Y. Ke, X. Tang, and F. Jing, "The design of high-level features for photo quality assessment," in *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1. IEEE, 2006, pp. 419–426.

[12] L.-K. Wong and K.-L. Low, "Saliency-enhanced image aesthetics class prediction," in *Proceedings of the International Conference on Image Processing (ICIP)*. IEEE, 2009, pp. 997–1000.

[13] C. Li, A. Gallagher, A. C. Loui, and T. Chen, "Aesthetic quality assessment of consumer photos with faces," in *Proceedings of the International Conference on Image Processing (ICIP)*. IEEE, 2010, pp. 3221–3224.

[14] S. S. Khan and D. Vogel, "Evaluating visual aesthetics in photographic portraiture," in *Proceedings of the Eighth Annual Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging*. Eurographics Association, 2012, pp. 55–62.

[15] Z. Byers, M. Dixon, K. Goodier, C. M. Grimm, and W. D. Smart, "An autonomous robot photographer," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, 2003, pp. 2636–2641.

[16] J. Campbell and P. Pillai, "Leveraging limited autonomous mobility to frame attractive group photos," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005, pp. 3396–3401.

[17] S. Bouabdallah and R. Siegwart, "Full control of a quadrotor," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*. IEEE, 2007, pp. 153–158.

[18] G. P. Tournier, M. Valenti, J. P. How, and E. Feron, "Estimation and control of a quadrotor vehicle using monocular vision and moire patterns," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2006, pp. 21–24.

[19] J. Roberts, T. Stirling, J.-C. Zufferey, and D. Floreano, "Quadrotor using minimal sensing for autonomous indoor flight," in *Proceedings of the European Micro Air Vehicle Conference and Flight Competition*, 2007.

[20] S. Grzonka, G. Grisetti, and W. Burgard, "A fully autonomous indoor quadrotor," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 90–100, 2012.

[21] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an rgb-d camera," in *International Symposium on Robotics Research (ISRR)*, 2011, pp. 1–16.

[22] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy, "Stereo vision and laser odometry for autonomous helicopters in gps-denied indoor environments," in *SPIE Defense, Security, and Sensing*. International Society for Optics and Photonics, 2009, pp. 733 219–733 219.

[23] J. Engel, J. Sturm, and D. Cremers, "Scale-aware navigation of a low-cost quadrocopter with a monocular camera," *Robotics and Autonomous Systems*, 2014.

[24] E. Graether and F. Mueller, "Joggobot: a flying robot as jogging companion," in *CHI'12 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2012, pp. 1063–1066.

[25] 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*. ACM, 2011, pp. 561–570.

[26] T. Naseer, J. Sturm, and D. Cremers, "Followme: Person following and gesture recognition with a quadrocopter," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*. IEEE, 2013, pp. 624–630.

[27] T. Krajník, V. Vonásek, D. Fišer, and J. Faigl, "Ar-drone as a platform for robotic research and education," in *Research and Education in Robotics-EUROBOT 2011*. Springer, 2011, pp. 172–186.

[28] J. Engel, "Autonomous camera-based navigation of a quadrocopter," Master's thesis, Technical University Munich, 2011.

[29] S. Piskorski, N. Brulez, P. Eline, and F. DHaeyer, "Ar. drone developer guide," *Parrot, sdk*, 2012.

[30] ROS.org Documentation, "Ros introduction — ros.org documentation," 2014, accessed 15-March-2015. [Online]. Available: http://wiki.ros.org/ROS/Introduction

[31] D. E. King, "Max-margin object detection," *arXiv preprint arXiv:1502.00046*, 2015.

[32] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1. IEEE, 2005, pp. 886–893.

[33] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[34] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," in *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2014, pp. 1867–1874.

[35] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *Acm computing surveys (CSUR)*, vol. 38, no. 4, p. 13, 2006.

[36] M. Enzweiler and D. M. Gavrila, "Monocular pedestrian detection: Survey and experiments," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 31, no. 12, pp. 2179–2195, 2009.

[37] E. Hjelmås and B. K. Low, "Face detection: A survey," *Computer vision and image understanding*, vol. 83, no. 3, pp. 236–274, 2001.

[38] C. Zhang and Z. Zhang, "A survey of recent advances in face detection," Tech. rep., Microsoft Research, Tech. Rep., 2010.

[39] X. Zhu and D. Ramanan, "Face detection, pose estimation, and landmark localization in the wild," in *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012, pp. 2879–2886.

[40] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," University of Massachusetts, Amherst, Tech. Rep. 07-49, October 2007.

[41] T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, J. Friedman, and R. Tibshirani, *The elements of statistical learning*. Springer, 2009, vol. 2, no. 1.

[42] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou, and M. Pantic, "300 faces in-the-wild challenge: The first facial landmark localization challenge," in *ICCV Workshops*, 2013, pp. 397–403.

[43] E. Murphy-Chutorian and M. M. Trivedi, "Head pose estimation in computer vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 31, no. 4, pp. 607–626, 2009.

[44] A. Gee and R. Cipolla, "Determining the gaze of faces in images," *Image and Vision Computing*, vol. 12, no. 10, pp. 639–647, 1994.

[45] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1994.

[46] T. Smith and R. Simmons, "Point-based pomdp algorithms: Improved analysis and implementation," *arXiv preprint arXiv:1207.1412*, 2012.

[47] H. Kurniawati, D. Hsu, and W. S. Lee, "Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces." in *Robotics: Science and Systems*, vol. 2008. Zurich, Switzerland, 2008.