Particle Filter Localization for Unmanned Aerial Vehicles Using Augmented Reality Tags

Edward Francis Kelley V

Submitted to the
Department of Computer Science
in partial fulfillment of the requirements for
the degree of Bachelor of Arts
Princeton University

Advisor:
Professor Szymon Rusinkiewicz

May 2013

This thesis represents my own work in accordance with University regulations.

Edward Francis Kelley V

Abstract

This paper proposes a system for capturing 3D imagery of large objects using autonomous quadcopters. A major component of such a system is accurately localizing the position and orientation of the quadcopter in order to perform precise movements. This paper focuses on the design and implementation of a localization algorithm that uses a particle filter to combine internal sensor measurements and detected augmented reality tags in order to estimate the position and orientation of an AR.Drone quadcopter. This system is shown to perform significantly better than integrated velocity measurements alone.

Acknowledgements

Completing this thesis has been one of the most challenging, yet fulfilling experiences I have had in my time here at Princeton. I could never have completed this alone and I am indebted to a long list of mentors, friends, and family.

First and foremost, I would like to express my gratitude to my advisor, Szymon Rusinkiewicz, whose support and advice has been invaluable throughout this entire process. Professor Rusinkiewicz went above and beyond what could be expected of an undergraduate thesis advisor and I appreciate how much I have learned from him, ever since my days of struggling through "Death Graphics." I would also like to thank Professor Robert Stengel for his advice and support, as well as Professor Christopher Clark for introducing me to robotics and continuing to be a valuable source of advice during this project.

Furthermore, this project was funded by the School of Engineering and Applied Science, as well as the Morgan McKinzie ′93 Senior Thesis Prize Fund. I am grateful to attend a school that makes projects such as this possible.

A special thanks goes to my thesis partner Sarah Tang. Her friendship and cheery disposition made those long hours of watching generally-uncooperative quadcopters a fun and memorable experience.

I would also like to thank everyone who made my time at Princeton incredible. In particular: my fellow hashers in the Princeton Tower Club for keeping my spirits up and providing wonderful distractions from my computer screen; my quasi-roommates and dinner companions, Nick Adkins and Alice Fuller, for bringing me so much happiness and laughter; John Subosits for indulging my pretending to be an engineer; Rodrigo and the rest of the Menezes family for claiming me as one of their own; Jonathan Yergler, Coach Zoltan Dudas, and the rest of the Princeton Fencing Team for all of the wonderful memories, both on and off the strip.

Finally, I would like to thank my parents. It is only with their love and support that I have made it to this point. I am unable to fully convey my appreciation for everything they have done.

To my parents.

Contents

List of Figures

Chapter 1

Introduction

In recent years, research using small-scale Unmanned Aerial Vehicles (UAVs) has increased rapidly. Multi rotor aircraft, such as quadcopters (also known as quadrotors), have proven to be powerful platforms for applications in a variety of fields, from aerial photography to search and rescue [18, 17]. Once prohibitively expensive for most research projects, multi rotor aircraft have decreased substantially in price, ranging from a few hundred dollars to several thousand dollars. Additionally, on-board control systems have greatly added to the stability and ease of control of these aircraft, with many quadcopters using pre-programmed routines for difficult procedures such as takeoff and landing.

Although quadcopters have seen interest from the military and hobbyists for quite some time, these recent developments in price and stability have resulted in quadcopters being used in a wide array of applications. In 2010, a French company, Parrot, released the AR.Drone, a quadcopter intended for consumers. Unlike most other quadcopters, which were sold in kits targeted to experienced hobbyists or researchers, the AR.Drone was designed to be ready to fly right out of the box by an inexperienced pilot. Although affordable and easy to use, these quadcopters are far from just toys. Equipped with an array sensors and multiple cameras, the AR.Drone and other consumer-grade vehicles have been used by research groups to explore autonomous flight with a low barrier of entry, both in cost and complexity.

With the democratization of this technology, quadcopters can be used to be solved problems where cost and complexity for such a solution was once prohibitive.

1.1 Motivation

For applications ranging from developing video games to preserving archaeological artifacts, capturing 3D models of real-world objects has become an important task in a wide variety of fields.

While there are currently several different methods for capturing these models, each of these methods has associated limitations and drawbacks.

1.2 Current Acquisition Methods

1.2.1 Manual Model Creation

The most common method of model acquisition is having a trained modeler produce a 3D model using reference imagery and measurements. Models are tediously crafted in CAD software in a process similar to sculpting. While this technique is used extensively in video games and films, it is typically too time consuming and expensive for many applications such as archeology.

1.2.2 Laser Scanners

Laser rangefinder technology is the "gold standard" of 3D model acquisition in terms of precision. Modern scanners can produce sub millimeter scans, which make them a great choice for detailed digitization of statues. Combined with high-resolution photograph texture-mapping, very few techniques can match the precision and quality

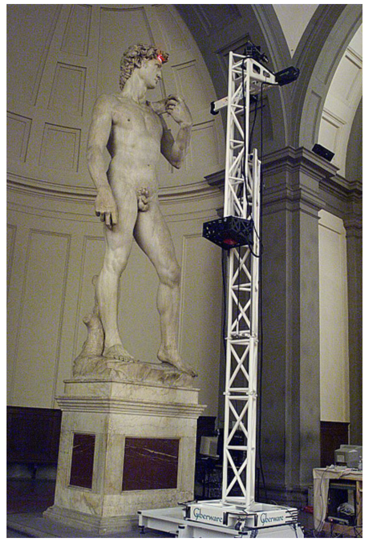


Figure 1.1: An example of a laser scanner setup used by the Digital Michelangelo Project [23].

of these scans. The Digital Michelangelo Project showed the power and precision of laser scanners by scanning several different statues, including Michelangelo's David, to 1/4mm accuracy [23].

However, laser scanners do have several drawbacks. The equipment is extremely expensive, bulky, and fragile. The Michelangelo Project had to transport over 4 tons of equipment to Italy in order to produce their scans. Additionally, laser scans involve immense setup and can take many hours. The model of David took over a thousand man-hours to scan and even more than that in post processing [23].

1.2.3 Multi-View Stereo

Multi-view stereo uses a collection of 2D images to reconstruct a 3D object model. By viewing a single object from hundreds of different camera positions, it is possible to generate a 3D model. Although this technique originally required precisely known camera coordinates, recent algorithms can produce a 3D model from an unordered collection of images with unknown camera positions, assuming that there is sufficient

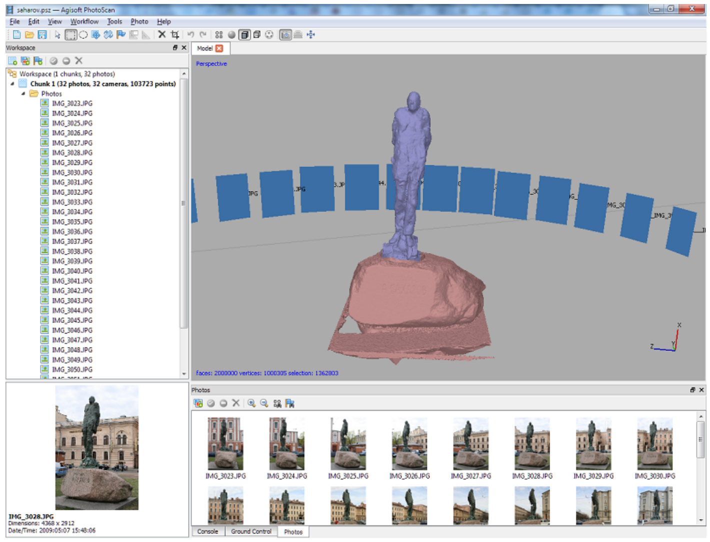


Figure 1.2: A 3D model of a statue generated by Agisoft Photoscan. Notice the derived camera planes encompassing the statue [1].

coverage. Existing software packages such as Bundler and Agisoft Photoscan can produce high-quality 3D reconstructions using these unordered image collections [6, 1].

The ability to use a collection of images without precise camera position information means that these 3D objects can be modeled substantially faster than with a laser scanner. With a smaller object, it is a relatively simple process to take pictures of the object from many different angles. However, for a larger object, such as a statue or building, the task of gathering imagery becomes substantially more difficult.

1.2.4 Microsoft Kinect

Several groups have researched using the Microsoft Kinect for model acquisition. As the Kinect produces an RGB image with depth information, either the Kinect or the model most be moved in order to produce a complete scan. While this has been found to be useful in scanning small to medium size objects, such as people, the Kinect has several limitations. First of all, the depth range is rather short, on the order of a few meters. Additionally, because the depth is partially determined using an infrared pattern, the Kinect does not work in locations with a large amount of background infrared light, such as outdoors.

1.3 Problem Definition

The goal of this system to capture imagery of large 3D objects for use in multi-view stereo software. This system has several requirements:

1. **Low Cost**

   The system should be substantially cheaper than laser scanning.

2. **Easy to Use**

   This system should be able to be deployed by users with minimal training and setup. Additionally, the hardware should be off-the-shelf and easily accessible.

3. **Complete Coverage**

   The system must be able to capture images from a wide variety of positions, completely covering every part of the target object.

4. **High Quality Imagery**

   The system must produce sufficiently high resolution, non-blurry images for use in multi-view stereo software.

1.4 Proposed Solution

We propose using low-cost autonomous quadcopters to gather the imagery needed for use in multi-view stereo software. By flying a quadcopter with a mounted camera around the target object, imagery can quickly be gathered from a wide variety of positions around the object. Using quadcopters has many advantages.

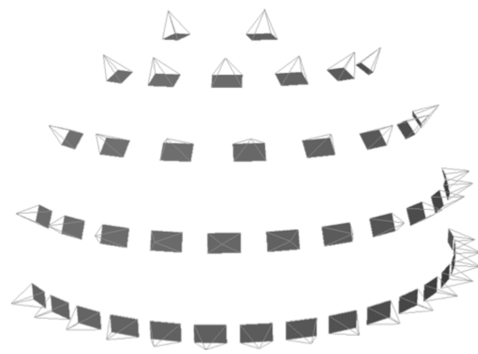


Figure 1.3: An example of camera positions for use in multi-view stereo. Such viewpoints could be attained by using quadcopters [18].

1. Quadcopters can capture images from positions unreachable by ground-based cameras.

2. By methodically flying around the target object at different altitudes, complete coverage of the target object can be achieved.

3. The imagery can be captured very quickly, on the order of a few minutes.

4. Quadcopters are small, portable, and easily deployable.

1.5 Solution Components

In order to implement such a solution, there are two main components which must be created. First, a localization algorithm is needed to produce position and orientation estimates of the quadcopter in the global frame. Then, a controller is needed to produce the control inputs for the quadcopter to move in the desired path. This thesis focuses on the design, implementation, and testing of the localization component, while the thesis of Sarah Tang in the Mechanical and Aerospace Engineering Department addresses the controller component of this system.

Chapter 2

Related Work

2.1 3D Model Acquisition Using Quadcopters

The past decade has seen a huge increase in the use of quadcopters for a variety of applications. With the improvement of stabilization software, quadcopters have seen a rise in popularity as a stable, cheap, and highly maneuverable aerial platform.

Although a relatively new field, several research groups have studied the use of quadcopters in 3D model construction. Irschara et al. created a system to generate 3D models using images taken from UAVs. While a quadcopter was used for gathering imagery, the quadcopter was manually controlled and the main focus of their work was photogrammetry-based model creation [18]. Steffen et al. studied surface reconstruction using aerial photography captured by UAVs [29].

2.2 GPS-denied Navigation of Quadcopters

There has been substantial work done on autonomous navigation of quadcopters without the use of GPS. As GPS signal cannot typically penetrate the walls or ceilings, any autonomous flights taking place indoors must use methods other than GPS for localization.

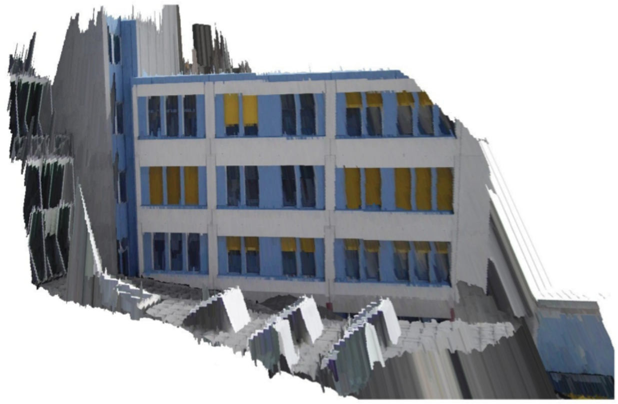
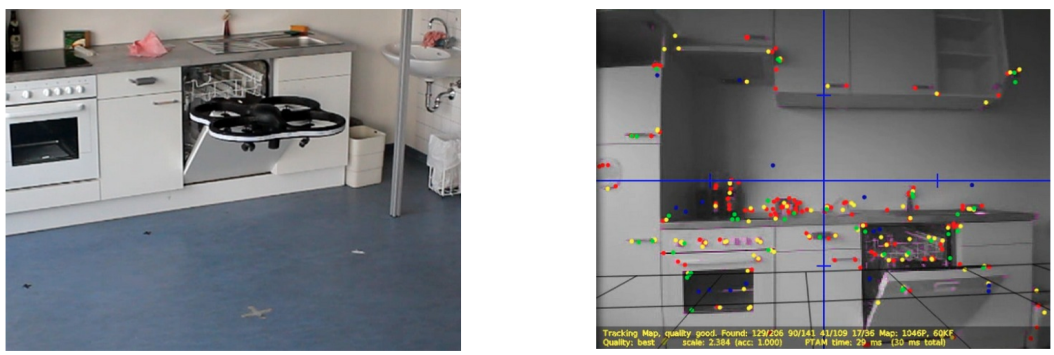


Figure 2.1: 3D Model of Building By Irschara et al. [18]

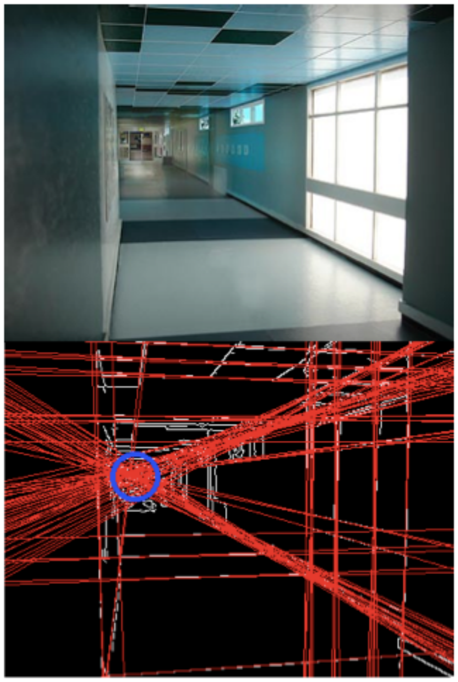


(a) Flying environment (b) Camera view with detected features

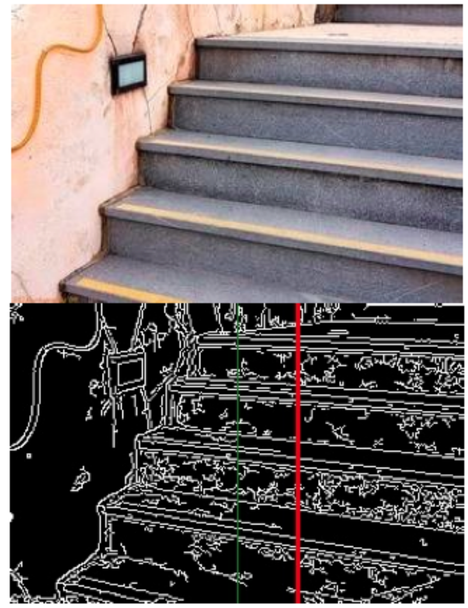
Figure 2.2: AR.Drone Localization Using Monocular SLAM [13].

Most relevant to this project, Engel et al. have published multiple papers on the camera-based navigation and localization of the AR.Drone [14, 13]. While they were able to achieve very accurate navigation, their work relies on the drone facing a mostly planar surface during the entire flight. As constructing models requires capturing imagery from a variety of positions and headings, this constraint is not acceptable.

Bills et al. present a method for navigating in an unknown interior space with the AR.Drone. By performing edge detection on the forward and downward facing cameras, their algorithm is able to determine the type of environment the quadcopter is in, either hallway or staircase, and issues the appropriate commands to autonomously navigate the space [10]. While this system is a great step in exploring interior space,



(a) Vanishing point detection



(b) Stair detection

Figure 2.3: Localization Using Perspective Cues [10].

it does not provide a solution to our problem. Unlike the work by Bills et al., our system must operate in essentially open space, with few visual cues, such as vanishing points, available for reference.

Chapter 3

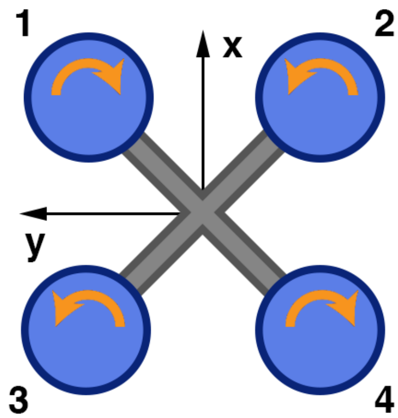
System Design

3.1 Quadcopter Characteristics

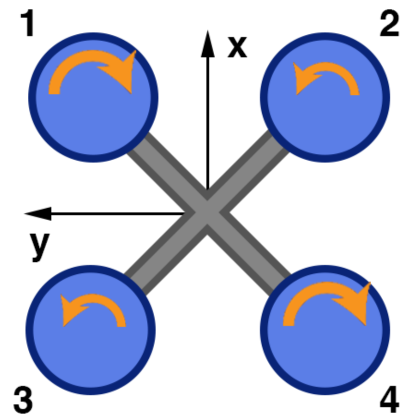
Quadcopters and other multi-rotor aircraft are mechanically much simpler than their conventional rotorcraft counterparts. In order to change roll, pitch, and yaw, a traditional helicopter uses a combination of mechanical linkages changing the pitch of the main rotor blades and a tail rotor. Quadcopters, on the other hand, consist of four fixed-pitch rotors which can be independently controlled to achieve a full range of motion. Because of their mechanical simplicity, Quadcopters are relatively simple to maintain and repair. Additionally, quadcopters are very highly maneuverable due to the ability to rapidly change the thrust of any individual rotor. Also, as there are four rotors contributing to the vertical thrust, each individual rotor possesses less kinetic energy than a single main rotor on a traditional helicopter, making them safer to be used in indoor spaces and around humans.

3.1.1 Basics of Quadcopter Flight

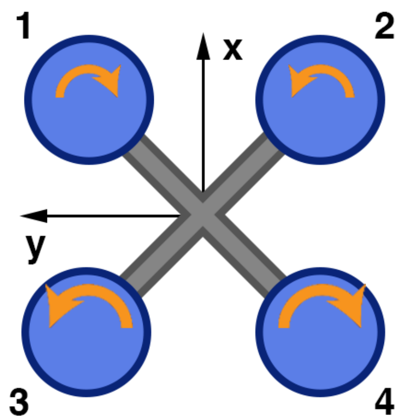
Each rotor produces a downwards thrust and torque opposite to the direction of rotation. In order to counter this torque, rotors along a single arm rotate in one



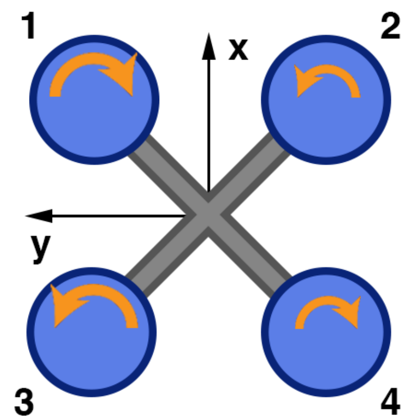
(a) Hover



(b) Yaw



(c) Pitch



(d) Roll

Figure 3.1: Quadcopter Flight Control.

direction, with the other rotors rotation in the other direction, as seen in Figure 3.1. If all of the rotors are turning at the same rate, then the torque forces will cancel out and the quadcopter will hover without rotating (Figure 3.1a). In order to change yaw, two of the rotors spinning in the same direction increase their thrust, and therefore their torque, and the other two rotors slow (Figure 3.1b). This induces a change in yaw without affecting the overall thrust. In order to change pitch or roll, two rotors on the same side increase thrust, with the other rotors decreasing thrust (Figures 3.1c and 3.1d) [12].

3.2 Parrot AR.Drone 2.0

(a) Indoor Hull

(b) Outdoor Hull

Figure 3.2: Parrot AR.Drone 2.0 [8].

This system will use the AR.Drone 2.0, the second generation of the consumer-grade quadcopter released by Parrot in 2010. The AR.Drone is a stabilized aerial platform that can be controlled by a user-friendly interface on a variety of mobile devices such as the Apple iPhone or iPad. The quadcopter is equipped with cameras and can be used for recording videos and playing augmented reality games.

Figure 3.3: AR.Drone FreeFlight Application for the Apple iPhone [8].

Forward Camera	HD, 720p 92° diagonal viewing area
Bottom Camera	QVGA, 320x240 64° diagonal viewing area
Computational Resources	1 GHz ARM Cortex-A8 CPU 800 MHz Video Digital Signal Processor 256 MB (1 Gbit) DDR2 RAM
Networking	802.11n WiFi
Sensors	3-axis gyroscope (2000 degree/second) 3-axis accelerometer (+/- 50 mg precision) 3-axis magnetometer (6 degree precision) Pressure sensor (+/- 10 Pa precision) Ultrasound altitude sensor

Table 3.1: AR.Drone 2.0 Technical Specifications [11].

3.2.1 Features

Considering its target audience of consumers, the AR.Drone is actually a very powerful research platform. The quadcopter is ready-to-fly out of the box. Unlike most quadcopters which are sold as kits, there is no assembly or technology knowledge needed to get started. Additionally, with the provided SDK, it is relatively easy to get off the ground and start running code to control the quadcopter. Finally, at only $300, the AR.Drone is much easier to fit into most research budgets than kit quadcopters which can cost thousands of dollars.

As this quadcopter is designed to be used by novice pilots, the AR.Drone comes with a protective expanded polypropylene shell which prevents the blades from coming in direct contact with anything. Additionally, if any of the motors sense collision, all of the rotors immediately shut off. This makes the AR.Drone an especially safe platform which is extremely unlikely to cause damage to either the target object or people.

The AR.Drone has two cameras, one forward-facing HD camera, and one lower resolution high frame-rate camera facing downwards. The AR.Drone processes the visual imagery on board to produce a velocity estimate. Depending on the ground material and lighting quality, the AR.Drone uses either muti-resolution optical flow or FAST corner detection with least-squares minimization to estimate movement between. The AR.Drone also uses the gyroscope and accelerometer on the navigation board to produce a velocity estimate and fuses this estimate with the vision-based velocity to create a relatively robust velocity estimation [11].

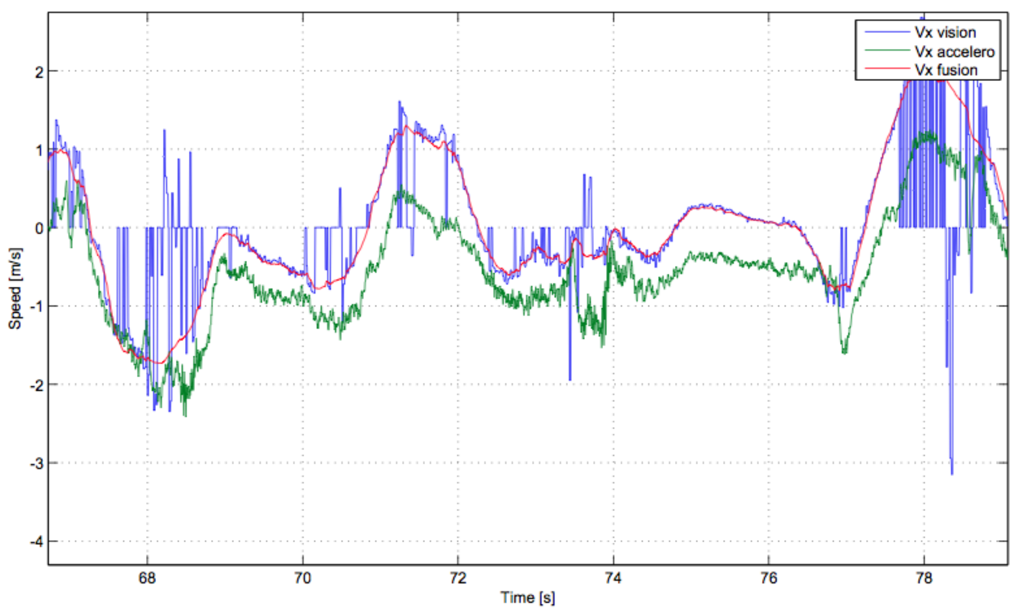


Figure 3.4: Graph showing the fusion of accelerometer and vision-based velocity measurements [11].

For altitude estimation, the AR.Drone uses a combination of an ultrasonic range sensor and pressure sensor. At heights under 6 meters, the AR.Drone relies solely on the ultrasonic sensor. Above those heights, where the ultrasonic sensor is not in its operational range, the AR.Drone estimates altitude based on the difference between the current pressure and the pressure measured on takeoff.

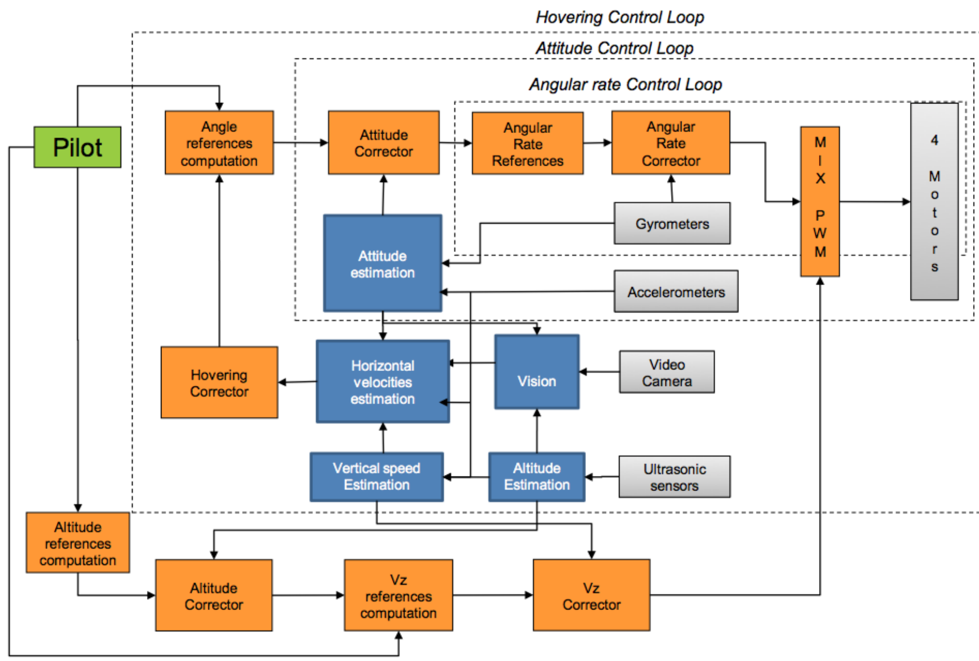


Figure 3.5: On-board AR.Drone Control Architecture [11].

The on-board processor handles low-level stabilization and wind compensation, allowing the quadcopter to hold position when not receiving control inputs. Commands to the AR.Drone are sent in the form of desired pitch and roll angles for translational movements, angular rate for yaw adjustment, and velocity for altitude adjustments. These high level commands are then translated by the on-board controller into rotor speed adjustments. More difficult actions, such as takeoff and landing, are completely handled by the on-board control. When the takeoff command is issued, the AR.Drone quickly takes off to a default height and hovers before accepting any movement commands.

3.2.2 Limitations

While the AR.Drone is a great platform for many research projects, it does have limitations when compared to hobbyist or professional-grade quadcopters.

The hardware design allows for very little customization. While most professional-grade quadcopters have ports for adding additional sensors, there is no straightforward way to add any extra electronics to the AR.Drone. Even if it were possible to customize, the AR.Drone is designed to only lift its own weight, with most hobbyists claiming to add a maximum of 100 grams payload before the flight characteristics are significantly affected [2]. Professional quadcopters of a similar size are typically able to fly with payloads between 400 and 600 grams [7].

Another limitation of the AR.Drone is the flight time. The maximum flight time of the AR.Drone is only around 15 minutes, with the additional weight of the indoor hull bringing this down to 10-12 minutes. Similar sized quadcopters, such as the Mikrocopter, typically achieve around 30 minutes of flight time, depending on weight and battery size [7].

Additionally, the AR.Drone has no built in GPS system, meaning that the on board measurements provide only relative measurements. This leads to errors due to drift and makes flying autonomously in a precise pattern an extremely challenging task.

A downside of the extra protection offered by the AR.Drone hull is that it is much larger than the hull of the Mikrocopter or similar quadcopters. This results in a larger surface area that can be affected by the wind, making outdoor flights particularly difficult even with the on-board stabilization.

3.3 System Architecture

3.3.1 Robot Operating System

The Robot Operating System (ROS) is used to organize the interaction between programs and libraries. Although not an "operating system" in the traditional sense, ROS is an open source communication layer used in a wide variety of robotics ap-

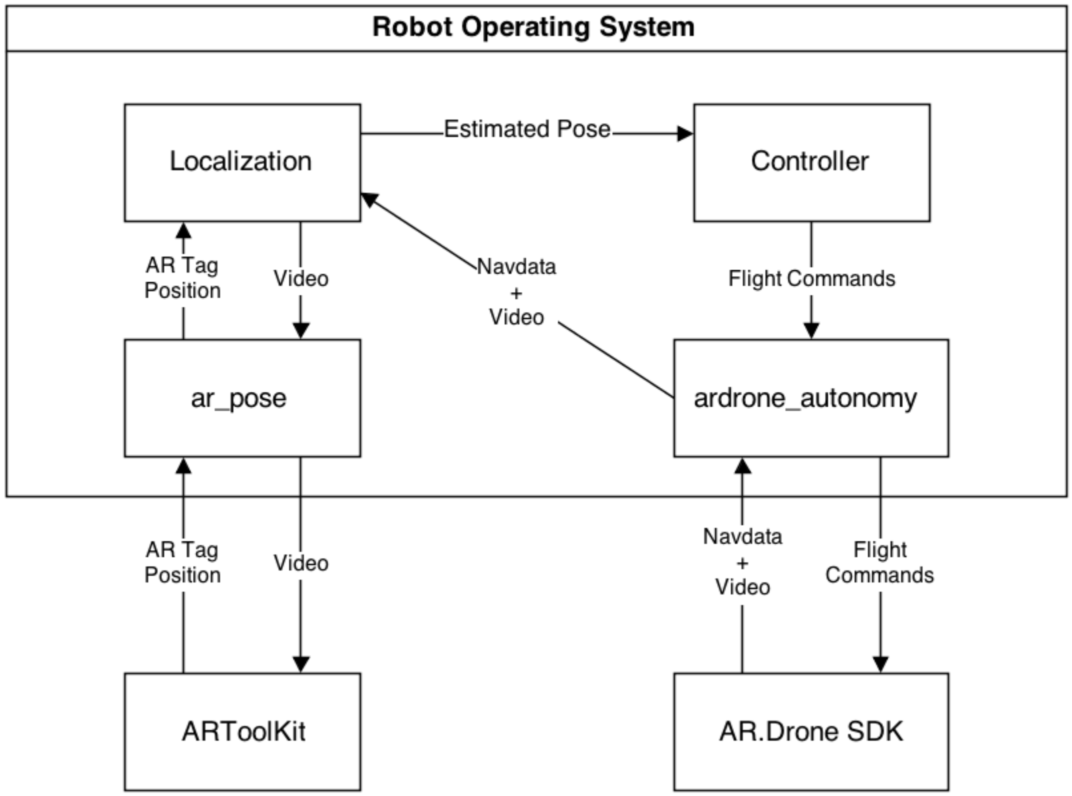


Figure 3.6: System Architecture Diagram.

plications. Supported by Willow Garage, ROS has a large amount of documentation and packages which can handle many common tasks in robotics. Many of these packages are hardware-independent, meaning that they can be quickly implemented on an array of different robotics system. ROS also provides a standard message protocol, allowing packages to work together in a language agnostic manner [27].

3.3.2 ardrone_autonomy

`ardrone_autonomy` is an open-source ROS wrapper for the Parrot AR.Drone SDK developed in the Autonomy Lab at Simon Fraser University [3]. This package handles the interface of navdata messages, video feeds, and control commands between ROS and the AR.Drone. This allows the use of many existing ROS packages in localizing and controlling the quadcopter.

3.3.3 ARToolKit

ARToolKit is an open-source software library designed to be used for creating augmented reality applications. Developed by Dr. Hirokazu Kato and maintained by the HIT lab at the University of Washington, ARToolKit uses computer vision algorithms to identify fiduciary markers, such as the one in Figure 3.7, and calculate the transformation between camera and tag.

For augmented reality applications, this can be used to superimpose 3D graphics onto a video feed in real time based on the tag position and orientation. In this system, the tags will be used to generate global positioning estimates for the quadcopter by combining estimated tag transformations with known tag locations.

Specifically, ARToolKit will be implemented using a slightly modified version of the `ar_pose` library, a ROS wrapper for ARToolKit developed by Ivan Dryanovski et al. at the CCNY robotics lab [4].

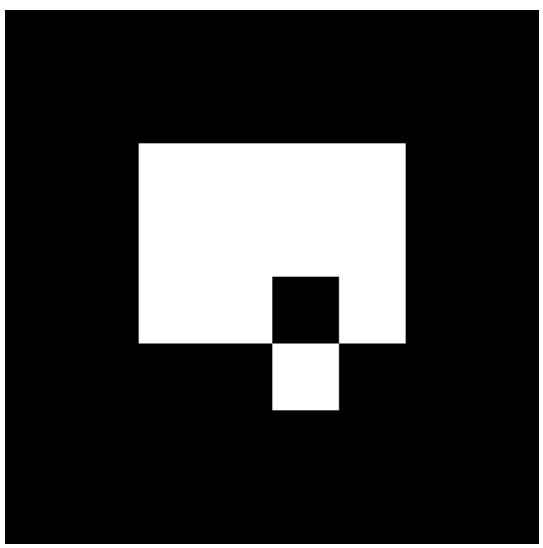


Figure 3.7: Augmented Reality Tag With ID 42.

3.3.4 Localization

The purpose of the localization module is to produce an estimated pose of the quadcopter. The localization module receives the navdata and video feed from the AR.Drone via ardrone_autonomy. The localization module then sends the video to `ar_pose` in order get any augmented reality tag transformations. By using measurements from the navdata messages and detected tags from `ar_pose`, the localization module produces a global pose estimation of the quadcopter.

3.3.5 Controller

The purpose of the controller is to produce the control inputs which move the quadcopter from its current pose to a desired pose. The controller receives the estimated pose from the localization module and sends the flight commands to the AR.Drone via `ardrone_autonomy`.

3.3.6 3D Reconstruction Software

After the flight is complete, 3D reconstruction software is used to turn the collection of images into a 3D model. There are multiple off the shelf libraries, including open-source Clustering Views for Muti-view Stero (CVMS) and commercial Agisoft Photoscan [16, 1].

Chapter 4

Localization

4.1 Problem Description

For a robot to perform precise maneuvers, it must first have an understanding of its position and orientation, or pose. This is the problem of localization. By using a variety of sensor measurements, the localization algorithm must produce a single estimate of the quadcopter's pose for use in the controller.

4.2 Considerations of the AR.Drone

As this project uses the AR.Drone 2.0, the localization algorithm is built around the capabilities and limitations of this hardware. Considering the low load-capacity of the AR.Drone and the fact that this project aims to use off-the-shelf hardware, the localization algorithm may only use the sensors included in the AR.Drone.

Therefore, the localization must produce an estimate by using some combination of the forward camera, downward camera, accelerometer, gyroscope, magnetometer, ultrasound altimeter, and pressure altimeter.

4.3 Localization Methods

Localization for mobile robots fall into three main categories: Kalman, Grid-Based, and Monte Carlo.

4.3.1 Extended Kalman Filter

The extended Kalman Filter (EKF) is used extensively in mobile robot localization. The EKF is a nonlinear version of the Discrete Kalman Filter first introduced by Rudolf Emil Kalman in 1960 [19]. The EKF linearizes about the current mean and covariance in method similar to a Taylor series. The EKF contains two stages, a prediction step and a correction step. In the prediction step, the new state and error covariance are projected. Then, in the correction step, the estimate and error covariance are updated in response to sensor measurements [32]. However, Kalman filters have the disadvantage of only being able to approximate normal distributions.

4.3.2 Grid-Based Markov Localization

Grid-based localization uses a "fine grained" grid approximation of the belief space, that is the space that covers all of the potential position and orientations of the robot [15]. For each time step, the probability of a robot being in any one of the grid cells is calculated, first by using odometry and then by exteroceptive sensors such as range finders. While relatively straightforward to implement, this process has many drawbacks. First of all, picking the size of the grid cells can be difficult. If the cells are too large, then the estimate will not be precise. However, if the cells are too small, the algorithm will be slow and very memory-intensive. Additionally, grid-based localization performs poorly in higher-dimensional spaces, as the number of cells grows exponentially with the number of dimensions.

4.3.3 Particle Filter

A particle filter is a type of Monte Carlo simulation with sequential importance sampling [9]. Essentially, a particle filter keeps track of a large number of particles, which represent possible pose estimation. The particle filter typically moves these particles using proprioceptive sensor measurements convolved with Gaussian noise [15]. Then, the particles are weighted with respect to exteroceptive sensor measurements. These particles are then randomly resampled based on these weight values, producing a corrected distribution of particles.

There are many advantages to using a particle filter. Due to the way the process creates an approximation by a set of weighted samples without any explicit assumptions of the approximation's form, it can be used in applications where the assumption of Gaussian noise doesn't necessarily apply [9].

4.4 Particle Filter with Augmented Reality Tags

Algorithm 1 Particle Filter with Augmented Reality Tag Correction

```
for all t do
    if buffer_full() then
        predict(Δt, v_x, v_y, altd, θ)
    end if
    if recieved_tag() then
        correct(P)                ▷ Transformation matrix from camera to marker
    end if
    x_est ←get_estimate()
end for
```

The particle filter has been chosen for this project due to its flexibility, ease of implementation, and performance. In typical implementations of particle filters for mobile robots, the prediction step uses proprioceptive sensors, such as accelerometers, gyroscopes, and rotary encoders. Then, this estimate is typically corrected by using exteroceptive sensors, such as infrared, laser, or ultrasound.

However, due to the lack of horizontal range sensing, this particle filter uses a different division between the prediction and correction steps. The prediction step uses the stock configuration of sensors in the AR.Drone, specifically the fused velocity, gyroscope, and ultrasound altimeter measurements. Then, the correction step uses an estimated global position determined by augmented reality tags to resample the particles.

4.4.1 Buffering Navdata

The localization module receives navdata at 50Hz. Depending on the number of particles and computational resources available to the localization algorithm, this can be a higher rate than the particle filter can run the propagation step. Reducing the rate at which propagation is run allows the particle filter to use more particles, providing better coverage of the pose estimate space and increasing the likelihood of convergence.

Additionally, while a more rapidly updated pose estimate would be preferable, the accuracy of the measurement is not such that it is especially useful to update at a rate of 50Hz. For example, the maximum velocity that the quadcopter should ever achieve in this system is around 500mm/s. In .02 seconds, the quadcopter will have only moved 10mm, or 1cm. Considering that the desired accuracy of the localization is on the order of tens of centimeters, updating the estimated pose at a reduced rate is acceptable.

As the navdata is recieved, the navdata measurements, such as velocity and yaw, are added to a buffer of size n. Every n measurements, the propagate step is called with the simple moving average of the previous n values and the sum of the ΔT values since the last call to propagate. This results in a propagate rate of $50/n$Hz.

Although the buffer size is currently a hard-coded value, this could be dynamically changed based on the amount of delay between receiving navdata measurements and processing them in the propagate step. This would result in the highest propagate rate possible given a fixed number of particles. On the other hand, the buffer size could remain fixed with the particle filter adjusting the total number of particles, allowing for the best possible coverage at a guaranteed update rate.

4.4.2 Initialization

The particle filter is initialized by creating a set of N particles. Each of these particles represents a potential pose in the configuration space. In particular, each particle at a given time step t is of the form:

$$\mathbf{x}_t = \begin{bmatrix} x_t \\ y_t \\ z_t \\ \theta_t \end{bmatrix}$$

Where x_t, y_t, z_t are the position, in mm, and θ_t is the heading, in degrees, of the particle in the global coordinate space. As the low level stabilization and control of the quadcopter is handled by the on-board processor, it is not necessary to include roll and pitch in the pose estimate of the quadcopter as these are not needed for the high level control. The entire set of particles of size N at time step t can be described as:

$$\mathbf{X}_t = [\mathbf{x}_t[0], \mathbf{x}_t[1], ..., \mathbf{x}_t[N]]^T$$

Additionally, for each time step, there is a set of associated weights

$$\mathbf{W}_t = [w_t[0], w_t[1], ..., w_t[N]]^T$$

Normalized, such that

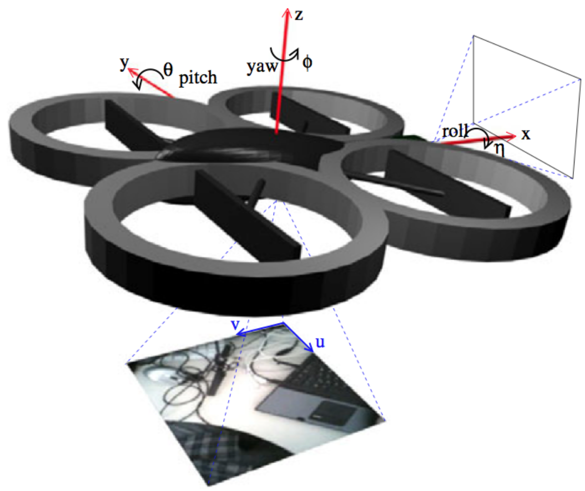


Figure 4.1: AR.Drone Coordinate System [21].

$$\sum_{i=0}^{N} w_t[i] = 1$$

Coordinate Frame Conventions

The coordinate frame follows the standard set by the ardrone_autonomy package. As shown in Figure 4.1, the coordinate frame is right-handed, with positive x as forward, positive y as left, and positive z as up. In terms of rotation, a counter clockwise rotation about an axis is positive. Heading ranges from -180 degrees to 180 degrees, with 0 centered along the x-axis. When the particle filter is initialized, the global coordinate frame is set equal to the first instance of the local coordinate frame.

4.4.3 Prediction Step

The first component of a particle filter is the prediction step. In this step, the position of every particle is updated by using the sensor measurements contained in the navdata messages. Specifically, the prediction step uses the elapsed time, Δt, fused velocity measurements, v_x and v_y, yaw reading, θ, and ultrasound altimeter value, z_{ultra}.

Algorithm 2 Prediction Step

1: **function** PREDICT($\Delta t, v_x, v_y, \theta, z_{ultra}$)
2: $\quad \Delta\theta \leftarrow \theta - \theta_{t-1}$
3: $\quad$ **for** $i = 0 \rightarrow N$ **do**
4: $\quad\quad \Delta\theta_{noise} \leftarrow randn(\Delta\theta, \sigma_\theta)$
5: $\quad\quad \theta_t[i] \leftarrow \theta_{t-1}[i] + \Delta\theta_{noisy}$
6: $\quad\quad (v_{x,global}, v_{y,global}) \leftarrow transform(v_x, v_y, \theta_t[i])$
7: $\quad\quad v_{x,global,noisy} \leftarrow randn(v_{x,global}, \sigma_v)$
8: $\quad\quad v_{y,global,noisy} \leftarrow randn(v_{y,global}, \sigma_v)$
9: $\quad\quad x_t[i] \leftarrow \Delta t * v_{x,\ global,\ noise} + x_{t-1}[i]$
10: $\quad\quad y_t[i] \leftarrow \Delta t * v_{y,\ global,\ noise} + y_{t-1}[i]$
11: $\quad\quad z_t[i] \leftarrow randn(z_{ultra}, \sigma_{ultra})$
12: $\quad$ **end for**
13: **end function**

The prediction step then subtracts the value of θ_{t-1}, the value of theta from the previous estimate, from θ to produce $\Delta\theta$. Then, for every particle i, a new pose $\mathbf{x}_t[i]$ is generated using the previous pose, $\mathbf{x}_{t-1}[i]$ and the values $\Delta t, v_x, v_y, \Delta\theta$, and z_{ultra}.

Adding Noise to Sensor Measurements

In order to update the pose estimation for a given particle, noise is first added to the sensor measurements in order to model the sensor noise. By adding noise to the particles, the distribution of particles should expand to include all of the possible belief states.

For each sensor value, a new "noisy" value is generating by a sampling from a Gaussian distribution with a mean of the sensor reading and a standard deviation, σ, which models in accuracy of the sensor.

Converting Local Velocity to Global Velocity

In order to update the pose of each particle in the global frame, the values $v_{x,noisy}$ and $v_{y,noisy}$, must be transformed from the local coordinate frame of the quadcopter to the global coordinate frame. First, on Line 5, the new heading of the particle is determined by adding the value $\Delta\theta_{noisy}$ to the estimated heading of the last time step, $\theta_{t-1}[i]$.

Then, on Line 6, the global velocity is found by

$$\begin{bmatrix} v_{x,global,noisy} \\ v_{y,global,noisy} \end{bmatrix} = \begin{bmatrix} cos(\theta_t[i]) & -sin(\theta_t[i]) \\ sin(\theta_t[i]) & cos(\theta_t[i]) \end{bmatrix} \begin{bmatrix} v_{x,noisy} \\ v_{y,noisy} \end{bmatrix}$$

Position Update

Once the velocity is in the global coordinate frame, Euler integration is used on Line 9 to determine the new position of the particle.

4.4.4 Correction Step

Determining Global Position from Augmented Reality Tag

The main concept in the correction step is to use a known marker position and a calculated camera-to-marker transformation in order to determine the global position of the quadcopter. Using this calculated global position, the particles, weighted by their similarity the calculated global position, are resampled to correct for the drift accumulated by using local measurements.

The first step is to determine the pose of the quadcopter in the global coordinate frame using augmented reality tags. These tags, each with unique binary patterns, are placed in known positions and orientations (e.g. axis aligned in the corners of a 5m square). These positions are published as ROS static transforms, allowing them to be retrieved by a *lookupTransform()* command. When a tag is detected by the `ar_pose` library, a marker is published containing the id of the tag and the transformation matrix from the camera to the tag, **P**. Then, the transformation matrices from the world frame to the tag,**M**, and from quadcopter base to camera,**B**, are retrieved using

Algorithm 3 Correction Step

function CORRECT($marker\_id, \mathbf{P}$) ▷ $\mathbf{P}$ is transformation from camera to marker

    $\mathbf{M} \leftarrow$ *lookupTransform(world, marker_id)*
    $\mathbf{B} \leftarrow$ *lookupTransform(base, camera)*
    $\mathbf{T} \leftarrow \mathbf{M}\mathbf{P}^T\mathbf{B}^T$
    $(x_{correct}, y_{correct}, z_{correct}) \leftarrow getTranslation(\mathbf{T})$
    $\theta_{correct} \leftarrow getHeading(\mathbf{T})$

    $w_{sum} \leftarrow 0$
    for $i = 0 \rightarrow N$ **do** ▷ Weighting Particles
        $dist \leftarrow \sqrt{(x_t[i] - x_{correct})^2 + (y_t[i] - y_{correct})^2 + (z_t[i] - z_{correct})^2}$
        $\theta_{diff} \leftarrow |\theta_t[i] - \theta_{correct}|$
        $w_{dist} \leftarrow pdf(dist, 0, \sigma_{x,artag})$
        $w_\theta \leftarrow pdf(\theta_{diff}, 0, \sigma_{\theta,artag})$
        $w_t[i] \leftarrow w_{dist} + w_\theta$
        $w_{sum} \leftarrow w_{sum} + w_t[i]$
    end for

    for $i = 0 \rightarrow N$ **do** ▷ Normalizing Weights
        $w_t[i] \leftarrow w_t[i]/w_{sum}$
    end for

    $walkerInitialize(\mathbf{X}_t, \mathbf{W}_t)$
    for $i = 0 \rightarrow N$ **do**
        $randVal \leftarrow rand(0, 1)$
        if $randVal < resampleThreshold$ **then** ▷ Random Resampling
            $x_{new} = [x_{correct}, y_{correct}, z_{correct}, \theta_{correct}]^T$
        else ▷ Weighted Sampling of Particles
            $x_{new} =$ *walkerSelect()*
        end if
        $\mathbf{X}_{temp} \leftarrow \mathbf{x}_{new}$
    end for

    $\mathbf{X}_t \leftarrow \mathbf{X}_{temp}$ ▷ Copy new set of particles

end function

lookupTransform(). Through matrix multiplication, the transformation from world to quadcopter, $\mathbf{T}$, can be obtained as follows:

$$\mathbf{T} = \mathbf{M}\mathbf{P}^T\mathbf{B}^T$$

On lines 6 and 7, the ROS transformation library is used to derive the estimated position and orientation of the quadcopter, $x_{correct}$, from the transformation matrix $\mathbf{T}$.

Weighting Particles

Each particle is assigned a weight by comparing the particle pose, $x_t[i]$, to the pose estimate generated using the augmented reality tag, $x_{correct}$. First, the Euclidean distance between the two positions and the difference between θ estimates are calculated on lines 12 and 13.

Then, on line 14, position and orientation weights are determined by inputing *dist* and *diff* into zero-mean probability density functions with standard deviations derived from testing.

$pdf(x, \mu, \sigma)$ is defined as:

$$\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

Where

x	Input value
μ	Mean of normal distribution
σ	Standard deviation of normal distribution

Random Resampling

Every time the correction step is called, a small percentage of particles are replaced using the pose as estimated by using the augmented reality tags. As the quadcopter will often fly for several meters without picking up a tag, the particles will become very spread out to account for sensor drift. Without replacement, there is a chance that the quadcopter would not be able to recover from a large amount of drift as no particle would be close enough to the actual position to accurately correct. By resampling, the particles will more rapidly cluster when the quadcopter is over a tag. There is a tradeoff of doing this, however. On rare occasions, the `ar_pose` library mistake the *marker_id* of the tag, resulting in an incorrect transformation. If the resample rate is too high, then a large number of particles will essentially "teleport" to the wrong position, resulting in a large jump of estimated pose. If this resample rate is small enough, then it will still accomplish the goal of quickly clustering particles when above tags, but will be robust enough to recover from incorrect tag transformations.

Weighted Sampling of Particles

The rest of the particles are chosen using a weighted sampling. Essentially, particles that are closest to the position calculated using the augmented reality tags will be chosen with a higher probability, allowing other particles to die out. This is often called "survial of the fittest" sampling. The naive method of performing weighted sampling would be to keep an array of cumulative weights. This way, a random number generated between 0 and 1 could be used to search through the array and pick particles with the correct distribution. This would result in a lookup time of $O(\log N)$ However, using Walker's alias method, a small amount of preprocessing can reduce these lookups to $O(1)$ [31]. As there are many particles and the correction step can be called at a very high frequency, it is important to reduce the lookup time as much as possible.

4.4.5 Pose Estimate from Particles

At the end of call to the localization algorithm, a new pose estimate is generated using the particle distribution. This is done by taking a linear combination of all of the particles. This method works reasonably well for this application as the particles tend to stay in a single cluster. In applications where multiple belief clusters are likely to exist, such as in a particle filter with an unknown start position, this solution would not be sufficient and a technique such as mean shift clustering should be used.

Chapter 5

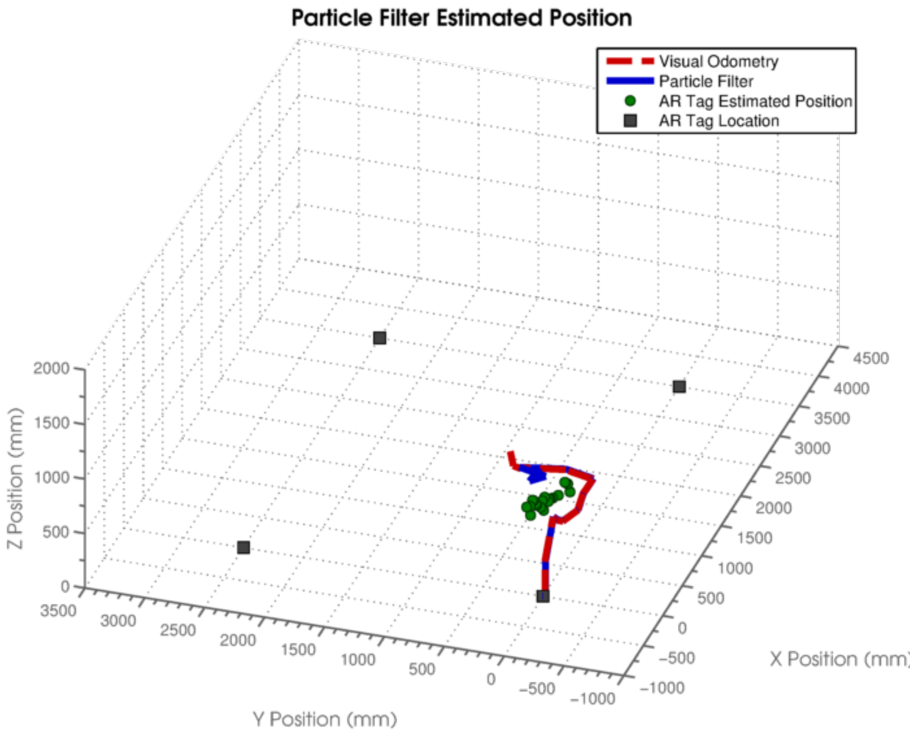
Results and Analysis

5.1 Sensor Testing

5.1.1 Gyroscope

5.1.2 Augmented Reality Tag Detection

5.2 Localization



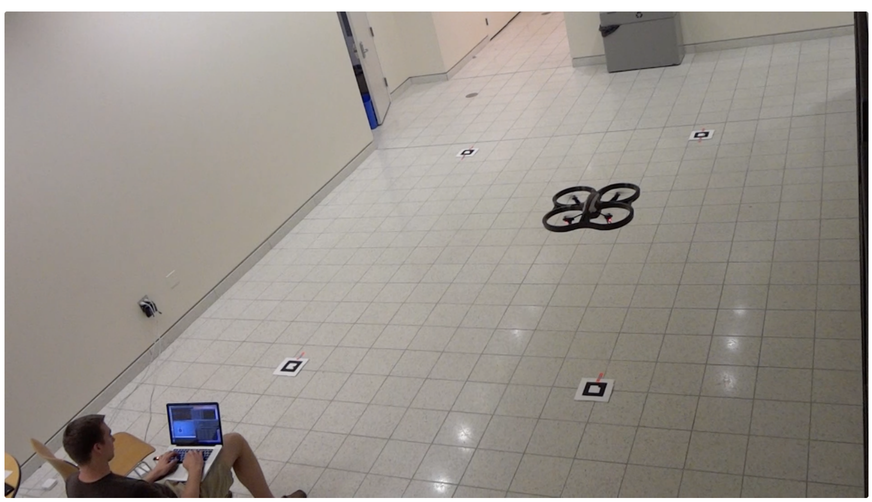


Figure 5.1: Manual Flight Test: First Tag

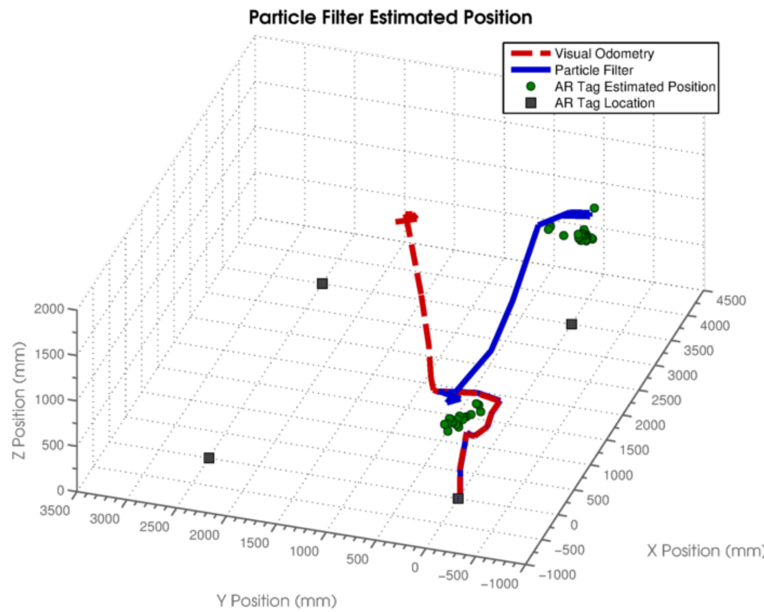
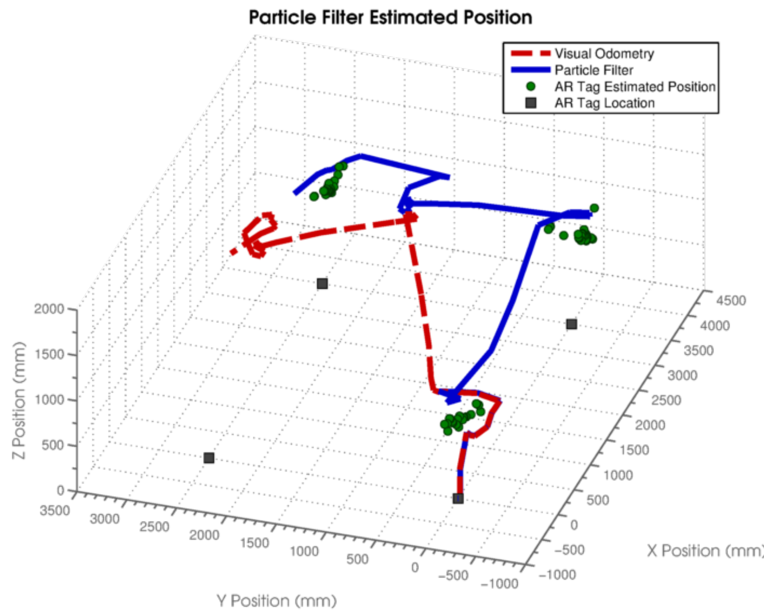


Figure 5.2: Manual Flight Test: Second Tag



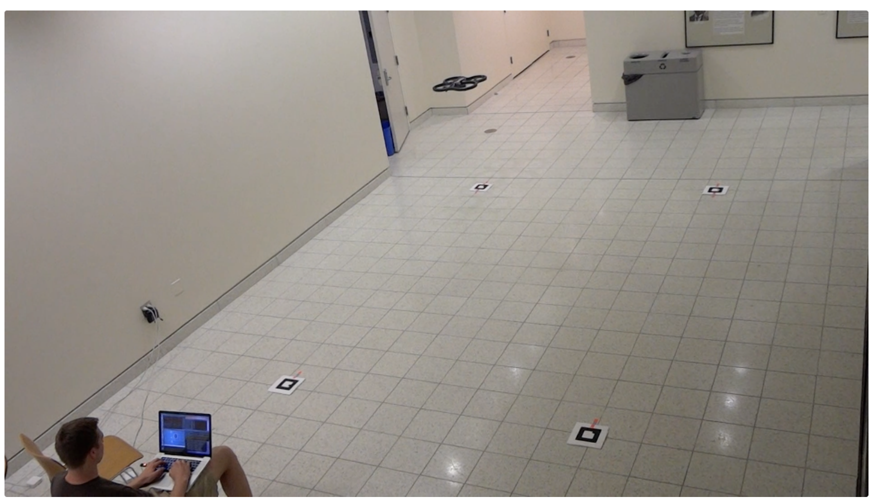
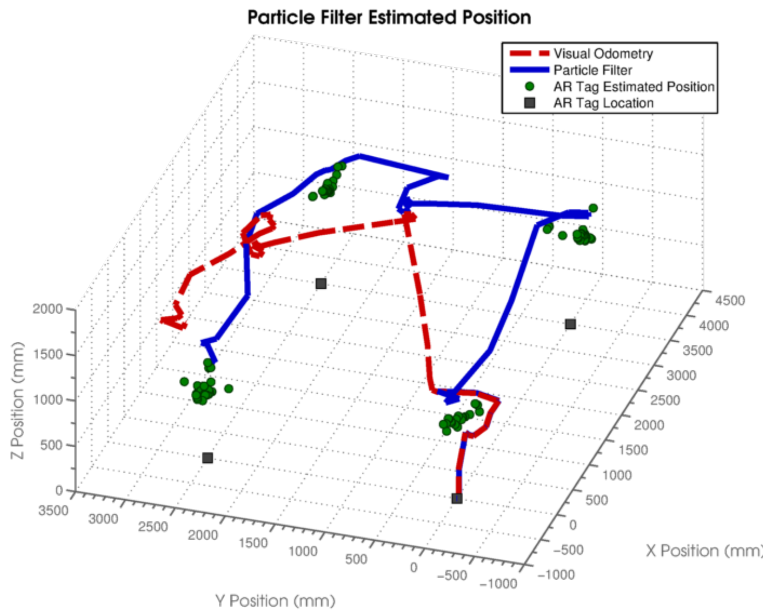


Figure 5.3: Manual Flight Test: Third Tag



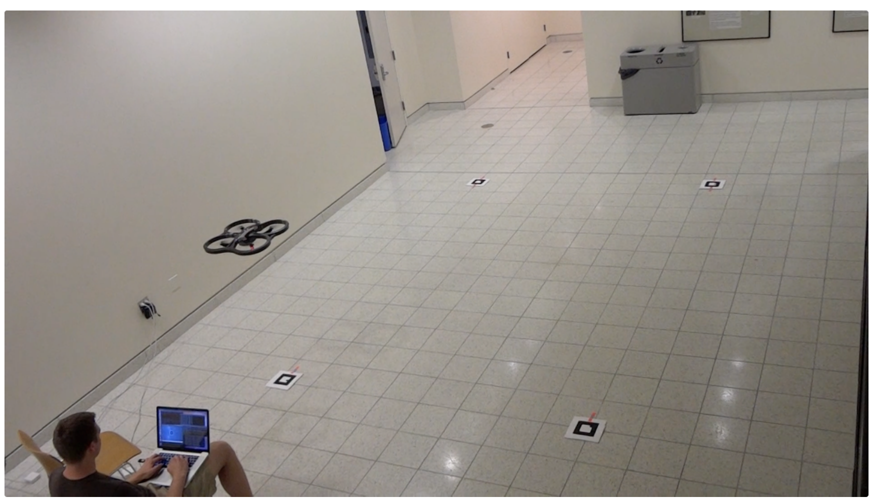


Figure 5.4: Manual Flight Test: Fourth Tag

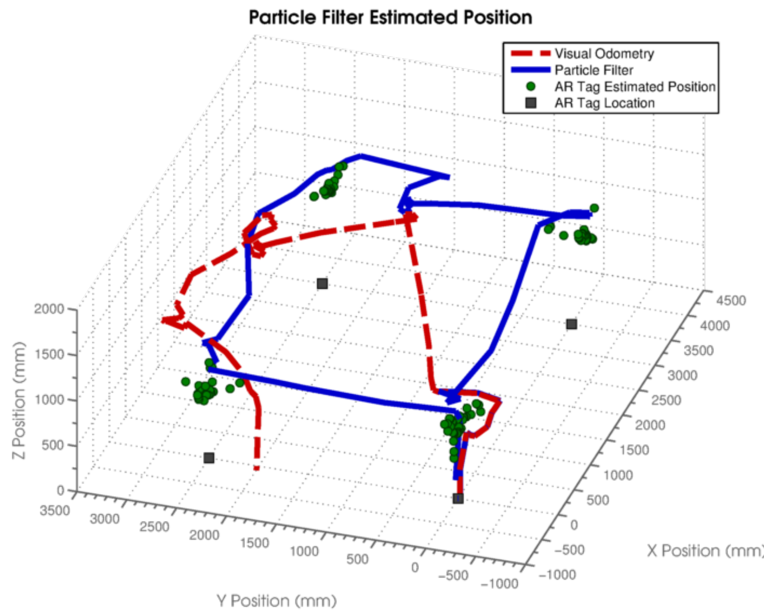


Figure 5.5: Manual Flight Test: Final Tag

Chapter 6

Conclusion

Bibliography

[1] Agisoft photoscan. `http://www.agisoft.ru/products/photoscan`.

[2] Ar.drone forum: Weight capacity. `http://forum.parrot.com/ardrone/en/viewtopic.php?id=4545`.

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

[4] ar_pose. `http://www.ros.org/wiki/ar_pose`.

[5] Artoolkit. `http://www.hitl.washington.edu/artoolkit/`.

[6] Bundler: Structure from motion (sfm) for unordered image collections. `http://phototour.cs.washington.edu/bundler/`.

[7] Mk-quadrokopter. `http://www.mikrokopter.de/ucwiki/en/MK-Quadro`.

[8] Parrot Press Photos. `http://ardrone2.parrot.com/photos/photo-album/`.

[9] Hamza Alkhatib, Ingo Neumann, Hans Neuner, and Hansjörg Kutterer. Comparison of sequential monte carlo filtering with kalman filtering for nonlinear state estimation. In *Proceedings of the 1th International Conference on Machine Control & Guidance, June*, pages 24–26, 2008.

[10] C. Bills, J. Chen, and A. Saxena. Autonomous mav flight in indoor environments using single image perspective cues. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5776–5783, 2011.

[11] Pierre-Jean Bristeau, Franois Callou, David Vissire, and Nicolas Petit. The navigation and control technology inside the ar.drone micro uav, 2011.

[12] Nick Dijkshoorn. Simultaneous localization and mapping with the ar.drone, 2012.

[13] J. Engel, J. Sturm, and D. Cremers. Accurate figure flying with a quadrocopter using onboard visual and inertial sensing. *IMU*, 320:240.

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

[15] Dieter Fox, Sebastian Thrun, Wolfram Burgard, and Frank Dellaert. Particle filters for mobile robot localization, 2001.

[16] Yasutaka Furukawa. Clustering views for multi-view stereo (cmvs). `http://www.di.ens.fr/cmvs/`.

[17] S. Gupte, P.I.T. Mohandas, and J.M. Conrad. A survey of quadrotor unmanned aerial vehicles. In *Southeastcon, 2012 Proceedings of IEEE*, pages 1–6, 2012.

[18] A. Irschara, V. Kaufmann, M. Klopschitz, H. Bischof, and F. Leberl. Towards fully automatic photogrammetric reconstruction using digital images taken from uavs. In *Proceedings of the ISPRS TC VII Symposium100 Years ISPRS*, 2010.

[19] Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[20] Hirokazu Kato. Artoolkit: library for vision-based augmented reality. *IEICE, PRMU*, pages 79–86, 2002.

[21] Tomáš Krajník, Vojtěch Vonásek, Daniel Fišer, and Jan Faigl. Ar-drone as a platform for robotic research and education. In *Research and Education in Robotics-EUROBOT 2011*, pages 172–186. Springer, 2011.

[22] K.Y.K. Leung, C.M. Clark, and J.P. Huissoon. Localization in urban environments by matching ground level video images with an aerial image. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 551 –556, may 2008.

[23] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 131–144, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[24] Joao Pedro Baptista Mendes. Assisted teleoperation of quadcopters using obstacle avoidance. 2012.

[25] Rudy Negenborn. *Robot localization and kalman filters*. PhD thesis, Citeseer, 2003.

[26] Paul Pounds, Robert Mahony, and Peter Corke. Modelling and control of a quad-rotor robot.

[27] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 2009.

[28] Szymon Rusinkiewicz, Olaf Hall-Holt, and Marc Levoy. Real-time 3d model acquisition, 2002.

[29] R. Steffen and W. Förstner. On visual real time mapping for unmanned aerial vehicles. In *21st Congress of the International Society for Photogrammetry and Remote Sensing (ISPRS)*, pages 57–62, 2008.

[30] Jing Tong, Jin Zhou, Ligang Liu, Zhigeng Pan, and Hao Yan. Scanning 3d full human bodies using kinects. *Visualization and Computer Graphics, IEEE Transactions on*, 18(4):643–650, 2012.

[31] Alastair J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Trans. Math. Softw.*, 3(3):253–256, September 1977.

[32] Greg Welch and Gary Bishop. An introduction to the kalman filter, 1995.

[33] Teddy Yap, Mingyang Li, Anastasios I. Mourikis, and Christian R. Shelton. A particle filter for monocular vision-aided odometry.