Particle Filter Localization for Unmanned Aerial Vehicles Using Augmented Reality Tags

Edward Francis Kelley V

Princeton University

Department of Computer Science

Advisors:

Professor Szymon Rusinkiewicz

Professor Robert Stengel

May 2013

Abstract

This is my abstract.

Acknowledgements

I want to thank me.

To my parents.

Contents

List of Figures

Chapter 1

Introduction

In recent years, research using micro aerial vehicles (MAVs) 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.[] Once prohibitively expensive, 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 these aircraft 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.This thesis explores the development of the localization component for an autonomous 3D model capture system using consumer-grade quadcopters.

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 have associated limitations and drawbacks.

1.2 Current Acquisition Methods

1.2.1 Manual Model Creation

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 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.[20]

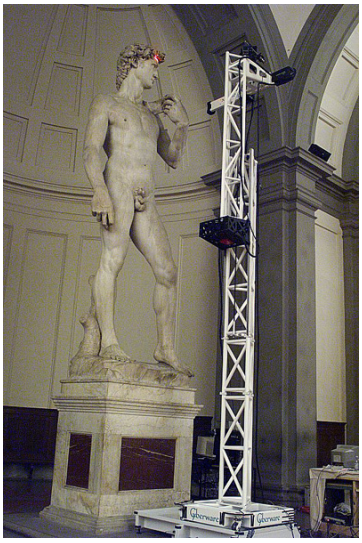

Figure 1.1: An example of a laser scanner setup used by the Digital Michelangelo Project [20].

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 scan of David took over a thousand man-hours to scan and even more than that in post processing [20].

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, a 3D model can be generated. 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 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

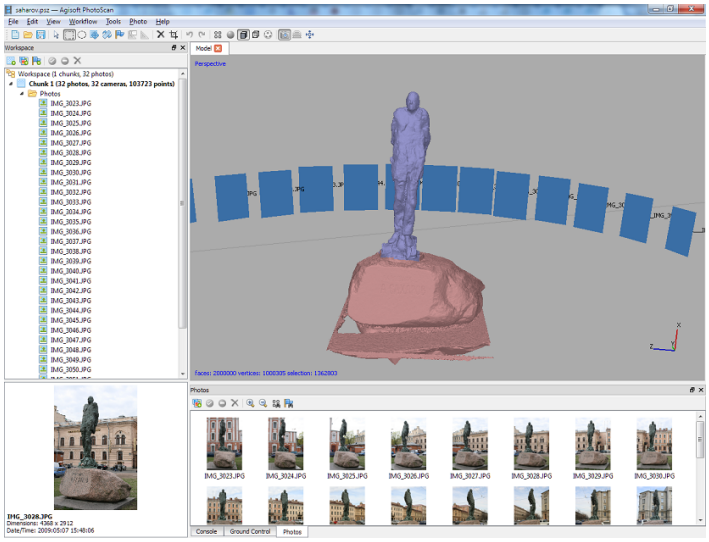

Figure 1.2: A 3D model of a statue generated by Agisoft Photoscan. Notice the derived camera planes encompassing the statue [1].

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 problem of gathering imagery becomes substantially more difficult.

1.2.4 Stereo Vision with Infrared Pattern

1.3 Problem Definition

We look to create a 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. 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 imagery needed for use in multi-view stereo software. By flying a quadcopter with a mounted camera around the target object, we can quickly and thoroughly capture images of the target from a wide variety of positions. Using quadcopters has many advantages.

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

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

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

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

Chapter 2

Related Work

2.1 Quadcopters for Model Acquisition

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 [16]. Steffen et al. studied surface reconstruction using aerial photography captured by UAVs [26].

Most relevant to our work, Engel et al. published multiple papers on the camera-based navigation and localization of the AR.Drone. While they were able to acheive very accurate navigation, their work relies on the drone facing a mostly planar surface during the entire flight, a constraint that is not possible in our application.

2.2 GPS-denied Navigation of Quadcopters

2.3 Research projects using the AR.Drone

Chapter 3

System Design

3.1 Quadcopter Characteristics

Quadcopters and other multi-rotor aircraft are mechanically much simpler than their conventional rotorcraft counterparts.

(a) Indoor Hull (b) Outdoor Hull

Figure 3.1: Parrot AR.Drone 2.0[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]

3.1.1 Basics of Quadcopter Flight

3.2 Parrot AR.Drone 2.0

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.

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.

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. The 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]

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.

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. Typically difficult actions, such as takeoff and landing, are completely handled by the onboard 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 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 get 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 in drift and makes flying autonomously in a precise pattern an extremely challenging task.

Finally, as the AR.Drone was designed to be used by inexperienced pilots, extra emphasis was put on making the quadcopter durable when it is inevitably crashed. Due to this, the polystyrene case and hull, particularly the indoor hull, around the body are much larger than that 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

This system will use the Robot Operating System (ROS) 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 applications. Supported by Willow Garage, ROS has a large amount of documentation and packages which can handle a large number of 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.[24][10]

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 3.2, and calculate the transformation between camera and tag orientation.

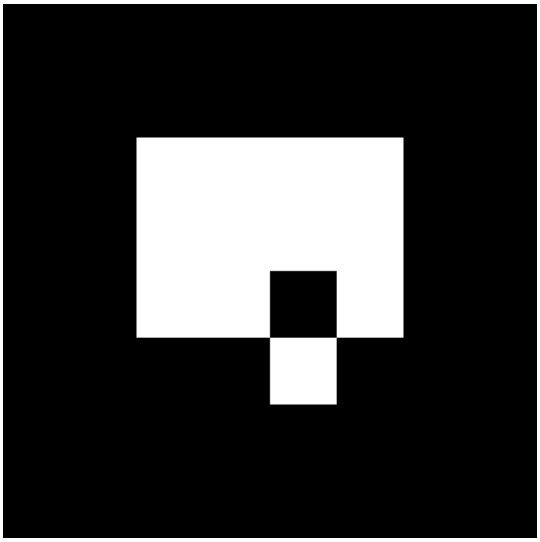

Figure 3.2: Augmented Reality Tag With ID 42

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]

3.3.4 Localization

Localization is the task of determining the position and orientation of the quadcopter.

3.3.5 Controller

3.3.6 Agisoft Photoscan

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, both proprioceptive and exteroceptive, 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 will be using the AR.Drone 2.0, the localization algorithm will be 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 will only be able to use sensors already included in the AR.Drone.

Therefore, the localization must produce an estimate by using a 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.

The extended Kalman Filter (EKF)...

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.

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

```
1: for all t do
2:     if buffer_full() then
3:         propogate(t_δ, v_x, v_y, altd, θ)
4:     end if
5:     if recieved_tag() then
6:         ar_correct(M)          ▷ Transformation matrix from camera to marker
7:     end if
8:     x_est ← get_estimate()
9: end for
```

4.4.1 Buffering Navdata

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

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 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.

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:

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

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:

$$X_t = \{x_t[0], x_t[1], ..., x_t[N]\}$$

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

$$W_t = \{w_t[0], w_t[1], ..., w_t[N]\}$$

Normalized, such that

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

Coordinate Frame Conventions

The coordinate frame follows the standard set by the ardrone_autonomy package. As shown in Figure [INSERT FIGURE], 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 as forward. When the particle filter is initialized, the global coordinate frame is set equal to the first 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 value, z_{ultra}.

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 $x_t[i]$ is generated using the previous pose, $x_{t-1}[i]$ and the values $\Delta t, v_x, v_y, \Delta\Theta$, and z_{ultra}.

Algorithm 2 Particle Filter Prediction Step

```
1: function PREDICT(t_δ, v_x, v_y, altd, θ)
2:     for i = 1...N do
3:         x_current ← x_{t-1}[i]
4:         x_t[i] ← n
5:     end for
6: end function
```

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, the new heading of the particle is determined by

$$\theta_t[i] = \theta_{t-1}[i] + \Delta\theta_{noisy}$$

Then, the global velocity can be found by

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

Position Update

Now that the velocity is in the global coordinate frame, Euler integration is used to determine the new position of the particle

$$x_t[i] = \Delta t * v_{x,global} + x_{t-1}[i]$$

$$y_t[i] = \Delta t * v_{y,global} + y_{t-1}[i]$$

4.4.4 Correction Step

Algorithm 3 Particle Filter Augmented Reality Tag Correction

```
1: if i ≥ maxval then
2:     i ← 0
3: else
4:     if i + k ≤ maxval then
5:         i ← i + k
6:     end if
7: end if
```

Determining Global Position from Augmented Reality Tag

Weighting Particles

Weighted Sampling of Particles

Random Resampling

Chapter 5

Results and Analysis

5.1 Sensor Testing

5.1.1 Gyroscope

5.2 Localization

5.3 Controller

5.4 Model Generation

Chapter 6

Conclusion

Appendix A

Implementation

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] Brian Berard. *Quadrotor uav interface and localization design*. PhD thesis, WORCESTER POLYTECHNIC INSTITUTE, 2010.

[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] 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.

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

[18] 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.

[19] 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.

[20] 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.

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

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

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

[24] 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.

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

[26] 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.

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