Localization and Navigation of a Holonomic Indoor Airship Using On-board Sensors

Mikelis Valdmanis

Department of Mechanical Engineering
McGill University, Montreal

October 2010

A thesis submitted to McGill University in partial fulfilment of the requirements of the degree of

Master of Engineering

Abstract

Two approaches to navigation and localization of a holonomic, unmanned, indoor airship capable of 6-degree-of-freedom (DOF) motion using on-board sensors are presented. First, obstacle avoidance and primitive navigation were attempted using a light-weight video camera. Two optical flow algorithms were investigated. Optical flow estimates the motion of the environment relative to the camera by computing temporal and spatial fluctuations of image brightness. Inferences on the nature of the visible environment, such as obstacles, would then be made based on the optical flow field. Results showed that neither algorithm would be adequate for navigation of the airship.

Localization of the airship in a restricted state space – three translational DOF and yaw rotation – and a known environment was achieved using an advanced Monte Carlo Localization (MCL) algorithm and a laser range scanner. MCL is a probabilistic algorithm that generates many random estimates, called particles, of potential airship states. During each operational time step each particle's location is adjusted based on airship motion estimates and particles are assigned weights by evaluating simulated sensor measurements for the particles' poses against the actual measurements. A new set of particles is drawn from the previous set with probability proportional to the weights. After several time steps the set converges to the true position of the airship. The MCL algorithm achieves global localization, position tracking, and recovery from the "kidnapped robot" problem. Results from off-line processing of airship flight data, using MCL, are presented and the possibilities for on-line implementation are discussed.

Abrégé

Deux approches de navigation et localisation d'un drone intérieur équipé de capteurs et capable de six degrés de liberté seront présentées. Premièrement, des vols ayant comme simple but d'éviter des obstacles et de naviguer le drone ont été exécutés à l'aide d'une caméra vidéo. Deux algorithmes de flux optique ont été étudiés. Le flux optique estime le déplacement de l'environnement relatif à la caméra en calculant les variations dans la clarté de l'image. Les traits caractéristiques de l'environnement, comme les obstacles, sont alors déterminés en se basant sur le champ de flux optique. Les résultats démontrent que ni l'un ni l'autre des algorithmes sont adéquats pour naviguer le drone.

La localisation du drone dans une représentation d'état, caractérisée par trois degrés de liberté en translation et par la vitesse de lacet, ainsi que dans un environnement connu a été accomplie en utilisant l'algorithme avancé de Localisation Monte Carlo (MCL) et un télémètre laser. MCL est un algorithme probabiliste qui génère aléatoirement plusieurs estimés, nommés particules, d'états potentiels du drone. À chaque incrément de temps, la position de chaque particule est ajustée selon les déplacements estimés du drone et ces particules sont pondérées en comparant les valeurs estimées du capteur avec les valeurs actuelles. Ensuite, un nouvel ensemble de particules est créé à partir du précédent en considérant la pondération des particules. Après plusieurs incréments de temps, l'ensemble converge vers la position réelle du drone. L'algorithme MCL accompli alors une localisation globale, un suivi de position et une résolution du problème du robot «kidnappé ». L'analyse hors-ligne des résultats avec l'algorithme MCL est présentée et les possibilités d'implémenter cette méthode en ligne sont discutées.

Acknowledgments

First I would like to thank my supervisor, Prof. Inna Sharf, for her immeasurable support and guidance that she has provided throughout my degree, as well as the financial support she provided. Further, I would like to thank my parents and family for everything that they have done for me, and Meagan Miller for her unwavering moral support. I also thank everyone at the Aerospace Mechatronics Laboratory and in particular Yin Yang for his work on improving the airship controller and transferring it to Simulink as well physically assisting with flying the airship, and Francis De Broux for designing and constructing the new airship frame as well as providing a French translation of the abstract.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The past decade has seen increasing interest in autonomous, indoor, aerial robots. Potential uses for these systems include commercial applications such as surveillance and security, public service applications such as search and rescue in unsafe structures, and academic applications in the research of advanced control and navigation techniques for aerial vehicles without the cost of developing large-scale prototypes. Concurrently, McGill's Aerospace Mechatronics Laboratory (AML) has been home to an indoor airship that is unique to the field. Though not initially designed as stand-alone, rather as a substitute for a free-floating satellite for research into space robotics, the airship presents an opportunity for a one-of-a-kind indoor robotic platform. For the airship to be truly useful in such a role, however, further work to improve its capabilities must be undertaken.

1.1 Background and Motivation

Previous work at the AML has focused on the autonomous capture of free-floating objects in space, such as satellites or space junk, using a robotic manipulator [45, 58]. Experimentally studying such systems on Earth posed an obvious problem due to the effects of gravity. To simulate such a free-floating object, a small, indoor airship was developed. The novel spherical design of the airship – further details of which will be presented in the following chapter – provided the capability for "holonomic" motion in six degrees-of-freedom (DOF).[1]

Once the airship had been constructed and the previous work had been completed, further interest in the airship as a stand-alone platform arose. Initially the focus was on improving the functionality of the airship by adding a rigid frame and developing a stable closed-loop controller for the airship [48]. A commercial motion-tracking system was installed in the

[1]In the context of mobile robotics, a system is called holonomic if the number of controllable degrees-of-freedom is equal to the total degrees-of-freedom.

AML to track the position and orientation of the airship. The motion-tracking system provided fast, accurate and reliable pose information, allowing for the development of the controller.

To further realize the airship as a stand-alone platform, such as an indoor surveillance robot, the bonds with the motion-tracking system must be broken. Since the motion-tracking system is physically fixed to the building, moving it is not an easy operation. Thus the airship is effectively constricted to flight within the relatively cramped volume visible to the motion-tracking system. Ideally, the airship would be able to navigate autonomously through any space without any preparation of the space, such as placing landmarks throughout, or previous knowledge of the space, such as a map. Achieving those goals would of course require that the airship be completely self-contained, including all required sensing and computational equipment. Though these goals are lofty for any autonomous robotic vehicle, much less an omni-directional aerial platform, they will serve to provide direction.

1.2 Problem Specification

With the ultimate goal for the airship laid out, the primary challenge was to move all sensory and computational equipment on-board the airship. The previous controller relied on the motion-tracking system and the work presented here is all computationally heavy, so moving the computer on-board was deemed infeasible for the time being. Instead, the first step to realizing the airship as stand-alone was to only move all sensing equipment on-board the airship. Of course, the motion-tracking system could not be mounted to the airship, so new methods of sensing were required.

Thus, the goal of this research project was to investigate and implement navigation and localization methods for the indoor airship solely using on-board sensors.

1.3 Review of Related Work

1.3.1 Indoor Airships

Over the past decade, many teams have developed small airships for indoor flight, the majority of which were of the traditional, ellipsoidal shape with thrusters at the rear and fins for stabilization. These small airships face many challenges, not the least of which is developing stable control systems due to the complicated, non-linear dynamics of airships and the severely restricted payload limits on small airships.

In [39], an airship with three propellers was used to investigate developing a control law with reinforcement learning. The initial attempt failed, but by using an "evolutionary segmentation" algorithm to optimally segment the state space, the reinforcement learning algorithm was able to generate a stable control law. In this case, the airship was restricted to motion in a vertical plane, since the controller's only feedback was through an off-board camera tracking the position of the airship in the plane.

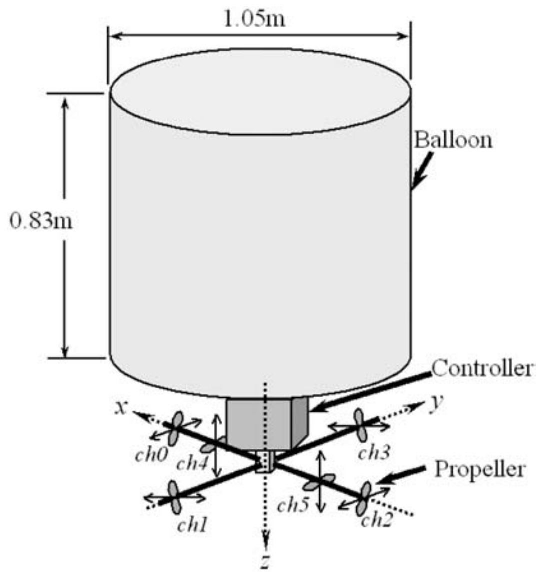
Reinforcement learning was again used by Ko et al. [32]. Rather than directly learn the control law, an improved dynamics model of the airship – again of the typical airship design – was first developed. A basic, human-derived model was improved through the use of a regression model to more accurately model the airship's dynamics. The improved dynamics model was then fed to the reinforcement learning algorithm to develop the control law. The airship in this work had 4-DOF and its position and orientation were tracked using a commercial motion-tracking system, much like that in the AML.

Another typical airship was used in [42] with the goal of a tele-operation system to control the airship over the internet. Due to the lag induced by tele-operation, "predictive motion control" was proposed, where the remote operator would be presented with the predicted state of the airship a few moments in the future, rather than the last measured state. The localization problem was not solved in this work, since the airship was tele-operated, though short-term navigational estimates were required. The predictive motion control system was not experimentally tested.

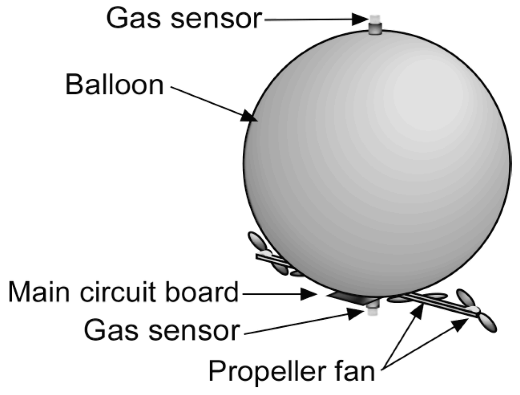
To further research on small, indoor airships, González et al. [18] proposed a low-cost autonomous blimp. They began with a commercially available blimp of the traditional shape, added a more powerful propeller to control height, a high-capacity battery, and two sonic range finders – one measuring altitude and one directed forward for basic obstacle avoidance. The 1.32 m long blimp was controlled by a ground-station PC that communicated wirelessly with an on-board microcontroller. Finally, two controllers were investigated using this airship: a PID controller fared poorly, but a fuzzy-logic controller was able to successfully control the altitude and orientation of the blimp while avoiding obstacles in its path.

Moving away from the traditional airships above, Kang and colleagues [28] presented an interesting airship that incorporated a wheeled base. In theory the base would be retractable, though in two physical implementations the carts were fixed to the airships using light-weight rods. The airships' buoyancies were set slightly below neutral with the propellers providing further lift to fly. The carts themselves were free-wheeled – all control and power were provided by the airships – but the motion sensors were mounted to the cart. An inclinometer measured the yaw rotation of the system, while an optical sensor like those found in optical computer mice measured displacement.

A further departure from the traditional airship design was developed by Takaya et



(a) The cylindrical airship of [29, 49].



(b) The spherical airship of [27].

Figure 1.1 Other non-traditional airships: the only airships found in the literature that did not feature the traditional ellipsoidal shape.

al. [49]. Their airship had a unique cylindrical design with six propellers, as shown in Fig. 1.1(a). The airship was symmetrical about the vertical axis and could thus perform holonomic maneuvers in four degrees-of-freedom (DOF); the arrangement of the propellers prohibited control of roll and pitch motions. Initially a PID landing controller was developed that used a downward facing camera to track a known pattern on the laboratory floor. Later, the same airship was used to investigate a learning landing controller [29].

Finally, a single other spherical indoor airship was found in the literature [27]. Used for measuring the gas distribution throughout a room, the blimp had a 1.17 m diameter and featured four thrusters – two directed downward, two horizontally – all along a single arm tangential to the sphere at the bottom of the airship, as depicted in Fig. 1.1(b). With this arrangement four DOF were achievable, which were used to cover the entire space of the room. A single wide-angle camera on the floor tracked the blimp's horizontal position and an on-board sonic range finder measured altitude, though the position data was only used to know the gas distribution, not for control of the airship itself. The blimp's lateral navigation commands were selected randomly, while its vertical motion was set every 5 seconds according to its height, always directed toward the half-height of the room.

1.3.2 Indoor Navigation and Localization

Indoor navigation and localization present unique challenges compared to outdoor navigation, where the global positioning system (GPS) is generally considered the solution to localization. Many methods of localization and navigation have been developed over recent years. Methods that rely on off-board equipment are common. For example, among the systems described in §1.3.1 were off-board vision cameras that tracked the positions of the airships in planar environments [27, 39] and a commercial motion tracking systems that used infra-red cameras to track markers on the airship [32]. A similar motion tracking system has also been used to track the airship within the AML. Such methods are abundant and beyond the scope of the present work.

Systems that do not rely on off-board sensors generally fall into two broad categories: vision-based navigation techniques and probabilistic localization techniques. Vision-based techniques will use images or image sequences from one or more cameras mounted to the robot and extract information about the environment and/or the motion of the robot from those images. Examples of common approaches are landmark detection and tracking for localization, and optical flow for obstacle detection and avoidance. A detailed review of vision based navigation approaches will be presented in Chapter 3.

Probabilistic localization techniques model the state of the robot as a probability density across the state space. If data from the sensors suggest a specific state, the area surrounding that state will be assigned a higher probability of being the true robot state. Probabilistic techniques are sensor agnostic compared to vision based approaches, but common choices are range scanners and cameras, so probabilistic techniques and vision-based navigation are not mutually exclusive. Probabilistic localization will be reviewed in detail in Chapter 4.

1.4 Thesis Organization

The remainder of this thesis is structured as follows. Chapter 2 describes the laboratory facilities and all of the major pieces of equipment used throughout the research, including the airship and the sensors mounted to it. Chapters 3 and 4 then cover the two approaches taken to solving the navigation and localization problems. In Chapter 3 an attempt at vision-based navigation is presented and in Chapter 4, a probabilistic localization technique using a laser range-scanner. Each of those chapters review related work in the field, give theoretical descriptions of the approaches, describe the practical implementations, and discuss their results. Finally, Chapter 5 draws conclusions on the works of Chapters 3 and 4 and presents prospects for future work on these topics.

Chapter 2

Experimental Facilities and Equipment

All of the work presented here was performed in the Aerospace Mechatronics Laboratory and made use of several pieces of equipment found there. The laboratory has space to fly a small airship and is equipped with a commercial motion-capture system to track it. The laboratory also houses a 6-DOF CRS robot mounted on a linear track, however the robot was used only peripherally to the main focus of this research. Details of this equipment presented in §2.1 while the description of the airship airship and its associated components follows in §2.2. The final section of the chapter, 2.3, is dedicated to the detailed description of one interoceptive and two exteroceptive sensors mounted to the airship: an inertial measurement unit (IMU), and a wireless video camera and a laser range scanner (LIDAR), respectively. For the LIDAR specifically, a detailed uncertainty model is developed as it will be employed in Chapter 4 of this thesis.

2.1 Laboratory Equipment

Optical Motion-Capture System

A key piece of equipment for previous iterations of the airship controller was the VICON optical motion-tracking system. It consists of six infra-red video cameras with the lens of each surrounded by a ring of infra-red light-emitting diodes (LEDs). One of the cameras is pictured in Fig. 2.1. The cameras are distributed along a shallow arc across the laboratory ceiling, bolted directly to the ceiling or support beams. Under this arrangement, the operational area in which they can track movement is approximately 7 m $\times$ 3 m $\times$ 3 m (width $\times$ depth $\times$ height). The six cameras are all connected to a control computer that uses a proprietary system to track the 3-dimensional motion of pre-defined objects at approximately 120 Hz.

The light from the LEDs reflects off of small markers that are placed in set patterns on

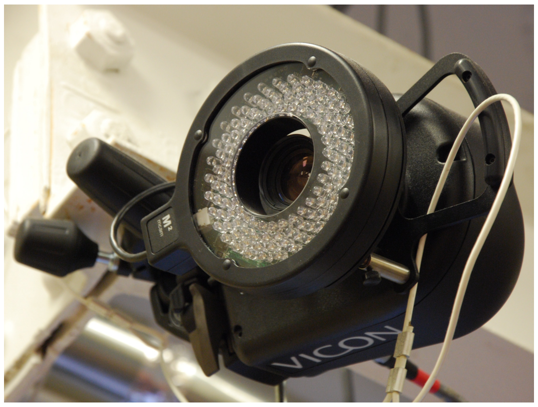


Figure 2.1 One of the six motion-capture cameras of the VICON system. The infra-red LEDs – not powered in this photograph – form a ring around the camera's lens. The camera is mounted directly to one of the building's structural support beams.

the objects to be tracked. The markers are coated in a highly reflective material that the cameras can reliably track. At least three markers are required to fully define a rigid-body. To track the airship, six groups of four markers are placed on the frame of the airship. Each group of four is defined as a unique rigid body with all six bodies defined such that they provide the same position and orientation no matter which groups are visible to the cameras. With this arrangement, the system is able to track the position and orientation of the airship reliably in the whole operating space without concern for the orientation of the airship.

Previous versions of the airship controller relied on the motion-tracking system for position and orientation information. During the present work the previous controllers were used to fly the airship for test data gathering, however, the motion-tracking system was not in any way incorporated into the localization or navigation algorithms.

Track-Mounted Robotic Arm

In the centre of the laboratory sits a track-mounted robotic arm. The CRS A465, visible in Fig. 2.2(a), is a 6-DOF manipulator with a horizontal reach of 710 mm and a vertical reach of 1040 mm. It is mounted on a 3 m track with a maximum velocity of 0.8 m/s.

In previous work [45, 58] the manipulator was used for autonomous capture of the airship. In the present work it plays only a minor role. It was used to gather the image sequence displaying uniform, constant motion for the optical flow work of Chapter 3.

2.2 A Holonomic Indoor Airship

The indoor airship is unique from all other airships encountered in the literature. With few exceptions, the airships studied in the literature match the traditional 'cigar' shape common to airships, with a roughly ellipsoidal hull, rear-facing thrusters, several stabilizing fins at the rear of the hull, and a gondola at the bottom of the hull, beneath the center of buoyancy. Such a configuration is designed for motion in a distinct forward direction and has two properties that greatly improve stability. Stabilizing fins help to maintain airships' headings when flying through air currents, and the arrangement of components ensures that the center of mass is well below the center of buoyancy, preventing roll and pitch motions from de-stabilizing the airship. On the other hand, the airships are under-actuated and display non-holonomic motion, typically lacking lateral motion actuators and unable to significantly control for rolling motions.

2.2.1 Physical Construction

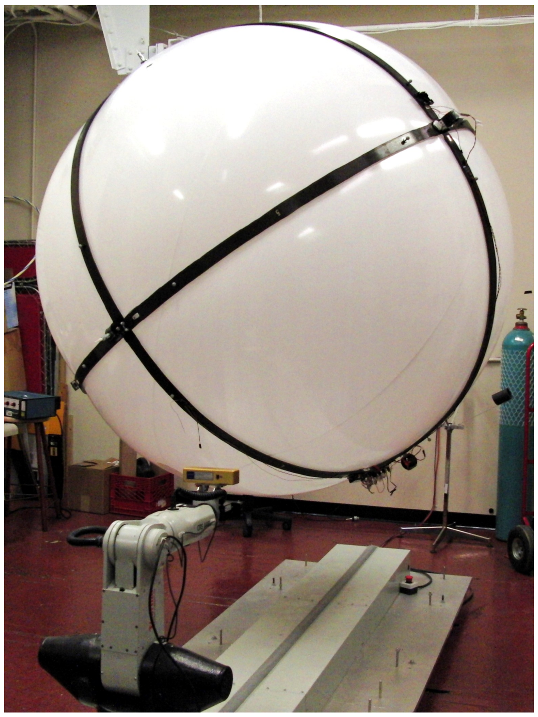
The indoor airship of the present work is nominally spherical in shape with a 1.85 m (6 ft) diameter. It consists of a 0.05 mm (2 mil) thick malleable plastic hull and three carbon-fiber hoops with light-weight honey-comb cores arranged normal to each other to provide a rigid frame. At each of the six hoop intersection points is a thruster. The six thrusters are oriented such that thrusters diametrically opposite are vectored in the same direction, and each of the three pairs of thrusters are vectored along different principal axes. A picture of the airship, together with the track-mounted robot, and a diagram of the hoop structure and thruster directions are displayed in Fig. 2.2.

Two iterations of the airship frame were used during this research. Both versions maintained the same overall design and materials, with the second iteration reducing the weight of the frame from 1.75 kg down to 1.28 kg. The original frame was used for the work presented in Chapter 3 and the new frame for the work discussed in Chapter 4.

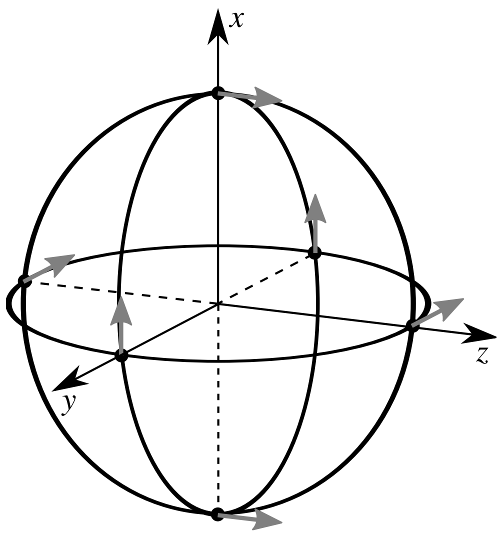
Airship Balancing

Originally developed as a simulator of free-floating satellites, the key design aspect was the ability to arbitrarily control the position, orientation and motion of the airship. This 6-DOF holonomic control was achieved by the spherical shape of the airship, the symmetrical placement of the thrusters and by meticulously preparing the airship prior to flight such that it has neutral buoyancy and is balanced. The airship can be made neutrally buoyant by adjusting the amount of helium in the hull, since the flexible hull material allows for slight over-inflation if necessary.

The airship is balanced such that its center of gravity (CG), center of buoyancy (CB), and center of volume (CV) are all coincident. This is achieved if the weight of the airship is evenly distributed about its hull. Ideally, for control purposes, the airship's inertia tensor



(a) The airship mid-flight in the AML above the robot track. The robotic manipulator is visible in the foreground.



(b) Diagram of airship's physical frame and coordinate frame, with thruster directions indicated by gray arrows

Figure 2.2 The spherical indoor airship.

would also be diagonal so that any rotation of the airship experiences equal inertia and no wobble would be induced by non-diagonal terms. Expressed mathematically, these two conditions are

$$\sum_{i=1}^{N} m_i \mathbf{r}_i = \mathbf{0} \tag{2.1a}$$

$$I = \begin{bmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & a \end{bmatrix} \tag{2.1b}$$

where N is the number of components of the airship of the airship, m_i is the mass of component i, $\mathbf{r}_i$ is the position of the CG of component i relative to the CV of the airship, and $a > 0$.

A script was written in MATLAB to assist with determining optimal payload arrangements on the airship that would achieve the conditions of eq. (2.1) as nearly as possible. Though the resulting arrangements remained imperfect – diagonalizing the inertia tensor was impossible, the airship's hull and hoops did not have perfectly uniform mass distributions, and precisely placing components onto the hoops was challenging – they were superior to manual placement of the components. The condition of eq. 2.1a was ultimately achieved through the use of small balancing masses attached to the frame.

Finally, it must be noted that for the work of Chapter 4, the above conditions were ignored to reduce the airship's state space to 4-DOF, though maintaining holonomic motion. Roll and pitch rotations were eliminated by moving CG as low as possible directly below CB. This was done for two practical considerations: the LIDAR was a large, concentrated mass that was difficult to balance and destabilized the airship's orientation controller; and computational concerns that will be explained in §4.3.3.

2.2.2 Airship Control

The airship is controlled from a ground-station PC that transmits commands to the airship wirelessly over a Futaba radio. The ground-station PC performs all computations for the controller. Previous work [48] has focused on the low-level stability and control of the airship. The full-state feedback controller uses the VICON motion-capture system for position and orientation data and calculates velocities by taking finite differences of the previous 10 samples. Ongoing work to improve the orientation controller with data from the inertial measurement unit was occurring concurrently with the present research and the test flights of Chapter 4 benefited from the improved orientation control.

The original controller was created in LabVIEW, but was recently ported to the Simulink environment with the QuaRC toolbox and soft real-time target developed by Quanser. A key function that is enabled by QuaRC is the possibility to run asynchronous threads alongside the primary, synchronous computations performed by Simulink, which permit separate asynchronous communications with the sensors. The importance of these will be displayed in §2.3.4.

2.2.3 Thrusters and Power

The six ducted fans consist of DC motors driving 48 mm diameter propellers within 35 mm long plastic cylinders. Each thruster is fixed in position and orientation and is capable of providing thrust in both forward and reverse directions. At a nominal voltage of 8.4 VDC, each thruster is capable of producing up to 0.45 N thrust in its primary direction or up to 0.25 N in its reverse direction.

Commands from the ground-station PC are received by the receiver of the Futaba wireless radio in the form of pulse-width modulation (PWM) values for each motor. A custom circuit board then drives the six DC motors according to the PWM values, since the Futaba radio is designed to control servo motors. All six thrusters, the Futaba receiver and the circuit board are powered by the same battery circuit. Originally, two parallel batteries of six 1.2 VDC AAA-sized cells were used. At the same time as the frame redesign, a single 8.4 VDC, 400 mAh lithium-polymer battery was purchased to enable longer flights. With a mass of 173 g, the new battery was the single heaviest component mounted to the airship.

2.2.4 Airship Component Masses and Lift

Payload restrictions are an important concern for any aerial vehicle, even more-so for indoor aerial vehicles due to their small size. Several of the component masses have been listed in their respective sections, and they are all tabulated in Table 2.1. The total lift and available lift are only approximations for three reasons: the lift provided by a constant volume of helium varies according to the ambient air pressure and temperature; the amount of balancing masses required depends on the components on the airship and their arrangement; and most significantly, the plastic hull is slightly flexible, allowing for slight over-inflation if necessary.

2.3 Sensors On-board the Airship

The components discussed to this point – the hull, frame, propellers, battery, radio, VICON system, and the ground-station PC – are enough to fly the airship under closed-loop control.

Table 2.1 Airship component masses and lift for both iterations of the airship frame. See text for notes on why the total and available lifts are approximate.

	Mass/Lift [g]	
Component	Original frame	Redesigned frame
Frame	1753	1278
Hull	640	640
Batteries	188	173
Thrusters	211	211
RF Receiver	25	25
Circuitry	53	53
Wiring	116	116
Total mass	2986	2496
Total lift (approx.)	3500	3500
Available lift (approx.)	500	1000

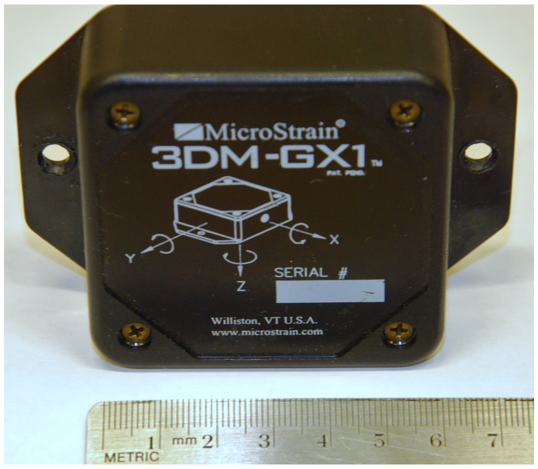
To attempt to control the airship without relying on the VICON system, further sensors were mounted to the airship's frame. These included a wireless video camera, an inertial measurement unit, a laser range scanner, and two Bluetooth transceivers for communication with the IMU and the LIDAR. All of these components are pictured in Fig. 2.3 and details of each will be discussed in the following sections.

To protect the sensors from unsteady power supplies due to the constant changes in current drawn by the motors, they were powered indepedently of the motors. The wireless camera was powered by a standard 9 V battery, while the remaining components were powered by a separate, 29 g, 7.2 VDC lithium-polymer battery paired with a regulator to maintain a constant voltage of 5.0 VDC to the sensors. This was primarily required to protect the LIDAR, since it has no power correcting circuitry built-in, unlike the other components. Finally, all of the components, including the Futaba radio and motor circuit board, but not the motors, were mounted to the airship using hook-and-loop fasteners for easy attachment and removal as necessary.

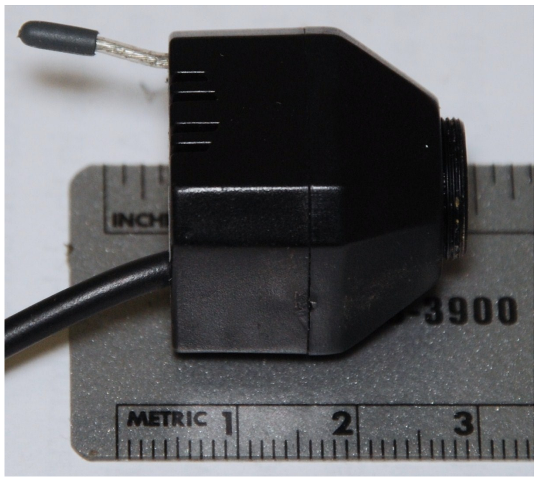
2.3.1 Inertial Measurement Unit

The MicroStrain 3DM-GX1 IMU shown in Fig. 2.3(b) was mounted to the airship primarily to provide orientation information, though it is also capable of measuring linear and angular accelerations as well as angular rates. With dimensions of 65 mm $\times$ 90 mm $\times$ 25 mm and a mass of 75 g, it is not the smallest IMU available, however it was already on hand from previous research.

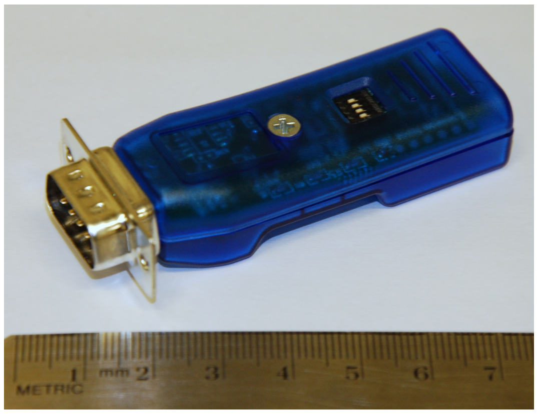
(a) Laser range scanner



(b) Inertial measurement unit



(c) Wireless camera



(d) Bluetooth transceiver

Figure 2.3 Airship sensors and components.

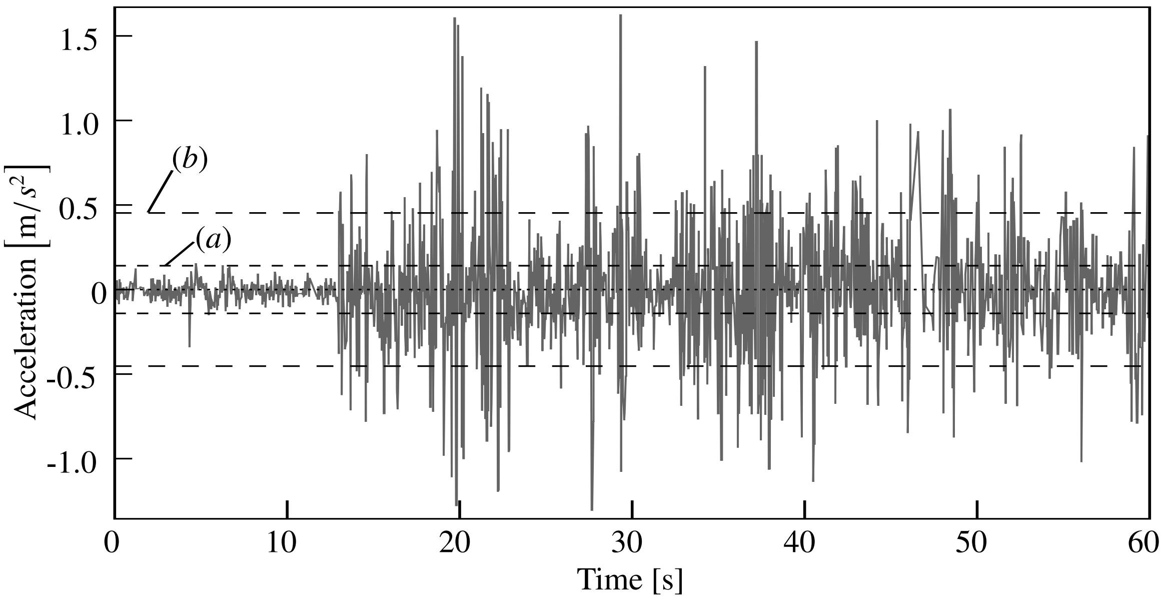


Figure 2.4 Effect of motor vibrations on IMU: the airship's thrusters were engaged after 13 s free floating. (*a*) Maximum linear thruster acceleration in any arbitrary direction (0.45 m/s^2); (*b*) Absolute maximum linear thruster acceleration (0.14 m/s^2).

Linear Acceleration Measurement

The accelerometers within the IMU have a full-scale range of 49.05 m/s^2 (5G). According to thruster characterizations, the ducted fans have a maximum thrust of 0.45 N in their primary direction and 0.25 N in their reverse direction. Moreover, at neutral buoyancy the airship has a total mass of approximately 3.5 kg. Thus, the absolute maximum linear acceleration of the airship, with all thrusters operating at maximum power in their forward direction, is 0.45 m/s^2 – 0.91% of the full-scale range. The maximum acceleration in any arbitrary direction, the case of only two thrusters operating at their maximum reverse power, is 0.14 m/s^2 – 0.29% of the full-scale range.

Compounding the small accelerations achievable by the fans are the vibrations they induce on the frame. As shown in Fig. 2.4, the accelerations on the IMU caused by the motors reached magnitudes of 1.5 m/s^2 – more than three times the maximum of the thrusters, completely overpowering the desirable acceleration signal.

Orientation Measurement

In contrast to the acceleration measurements, the orientations measured by the IMU are generally very accurate and do not suffer from the vibrations induced by the motors. Fig. 2.5 displays the roll, pitch and yaw rotations of the airship in flight as measured directly by the VICON motion tracking system and as measured by the IMU, after transforming the coordinate frame to match that of the VICON system.

The IMU was only used for the work of Chapter 4, where the only orientation required was the yaw orientation of the airship. Fortunately, of the three orientations, the yaw was most accurately measured by the IMU, with a mean error of only 0.7°, assuming the VICON system measured the ground-truth orientation. The roll and pitch orientations, while still quite accurate, displayed larger errors, with mean errors of 4.1° and 6.9°, respectively.

2.3.2 Wireless Camera

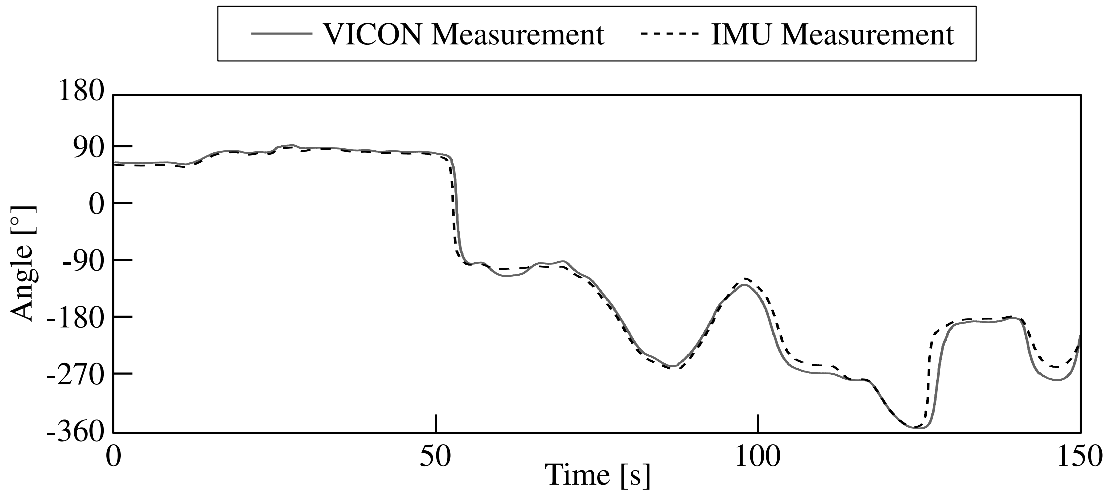
The vision-based navigation discussed in Chapter 3 used a small wireless camera to transmit a video stream. At only 25 mm × 22 mm × 22 mm, the camera is smaller than the standard 9 V battery required to power it. The CMOS camera, shown in Fig. 2.3(c), transmits at 30 colour frames per second with a resolution of 640 × 480 pixels, though in all of the present work this was down-scaled to 320 × 240 grayscale pixels to reduce computational loads. The camera was purchased second-hand and unfortunately no manufacturer information could be found, though the model number is C1182 and one retailer was found.[1]

2.3.3 Laser Range Scanner

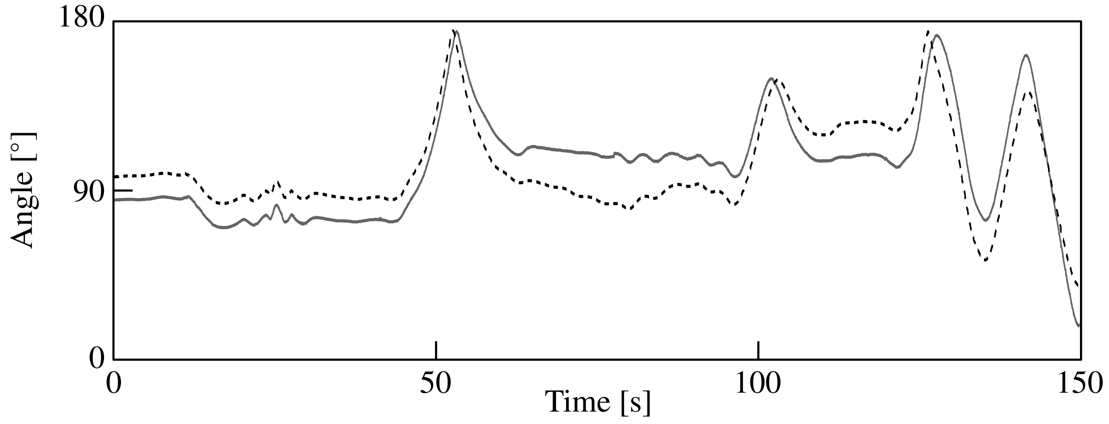
Aside from the motor battery, the largest and heaviest piece of equipment mounted to the airship was the laser range scanner pictured in Fig. 2.3(a). At a mass of only 144 g and dimensions of 50 mm × 50 mm × 70 mm, however, the Hokuyo URG-04LX is diminutive compared to the units typically used on large, outdoor vehicles. It uses a laser diode, a light sensor and a spinning mirror to measure distance. The phase difference of returned light to the sensor from that emitted is a function of the distance the light traveled, so the distance can be reliably calculated.

The spinning mirror allows the single diode and sensor pair to measure many distances along a circular arc. The LIDAR measures up to 682 evenly spaced distances along a 240° arc at a frequency of 10 Hz with a maximum range of 4095 mm and a distance resolution of 1 mm. It is also capable of averaging several adjacent measurements together to reduce data transmission loads – for example sending 341 distances per scan, each an average of two actual measurements. The manufacturer's specifications state the unit's accuracy at ±10 mm for measurements less than 1 m and ±10% of the reading beyond 1 m. Initial

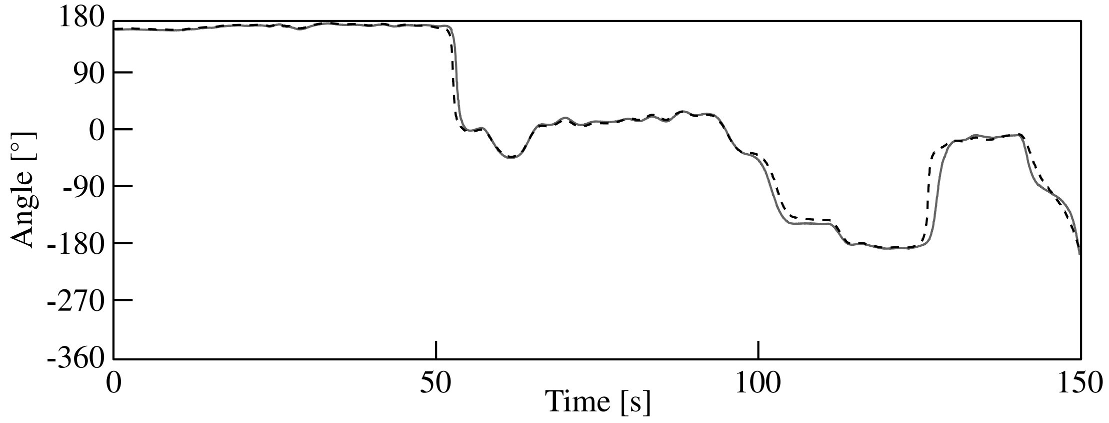
[1]SpyVille: available online at `http://www.spyville.com/mini-wireless-camera-5-8ghz.html`. Accessed January 24, 2011.



(a) Roll measurements



(b) Pitch measurements



(c) Yaw measurements

Figure 2.5 IMU orientation accuracy compared to the orientations measured by the VICON system.

testing confirmed these values as accurate, but a more detailed analysis was undertaken to develop a LIDAR uncertainty model.

LIDAR Uncertainty Model

The work of Chapter 4 requires an accurate uncertainty model for the LIDAR. The uncertainty model selected was based on that presented in [50]. Each measurement by the LIDAR is treated as an independent beam originating at the LIDAR. The probability density function (PDF) of the length of the beam is modeled as a combination of four types of uncertainty.

Local Measurement Noise Modern LIDARs tend to measure distances very accurately, however they are not free from small measurement errors. These small errors about the expected distance to hitting an object are modeled as Gaussian PDFs:

$$p_{\text{hit}}(r_t \mid x_t, m) = \begin{cases} \eta \mathcal{N}\left(r_t; r_t^*, \sigma_{\text{hit}}^2\right) & \text{if } 0 \leq r_t \leq r_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

where x_t is the robot state at time t, m is the map, r_t is the measured distance, r_t^* is the expected distance to the nearest obstacle along the beam at time t, $r_{\max}$ is the maximum range of the LIDAR, and $\mathcal{N}\left(r_t; r_t^*, \sigma_{\text{hit}}^2\right)$ is the normal distribution about mean r_t^* with standard deviation σ_{hit}:

$$\mathcal{N}\left(r_t; r_t^*, \sigma_{\text{hit}}^2\right) = \frac{1}{\sqrt{2\pi\sigma_{\text{hit}}^2}} e^{-\frac{1}{2}\frac{\left(r_t - r_t^*\right)^2}{\sigma_{\text{hit}}^2}} \tag{2.3}$$

The value η in eq. (2.2) is a normalizing constant that ensures $p_{\text{hit}}(r_t \mid x_t, m)$ integrates to one, since the two ends of the distribution are truncated at 0 and $r_{\max}$:

$$\eta = \left(\int_0^{r_{\max}} \mathcal{N}\left(r_t; r_t^*, \sigma_{\text{hit}}^2\right) dr_t \right)^{-1} . \tag{2.4}$$

Note that the value of σ_{hit} is not necessarily constant with r. For example, the specifications of the Hokuyo URG-04LX state $\sigma_{\text{hit}} = 0.01 r^*$ for $r^* \geq 1000$ mm.

Unexpectedly Short Measurements The uncertainty model assumes knowledge of the environment in which the robot is operating in the form of a map, m. Unless the map perfectly models the environment, a chance exists that the LIDAR scan will be interrupted by some unmodeled obstacle, such as a person moving through the environment. Since these obstacles are themselves unmodeled they must be accounted for within the LIDAR's uncertainty model. Here, an exponential distribution is used:

$$p_{\text{short}}(r_t \mid x_t, m) = \begin{cases} \eta \lambda_{\text{short}} e^{-\lambda_{\text{short}} r_t} & \text{if } 0 \leq r_t \leq r_t^* \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

where λ_{short} is the exponential distribution parameter and again η is a normalization variable:

$$\begin{aligned} \eta &= \left(\int_0^{r_t^*} \lambda_{\text{short}} e^{-\lambda_{\text{short}} r_t} dr_t \right)^{-1} \\ &= \left(-e^{-\lambda_{\text{short}} r_t^*} + e^{-\lambda_{\text{short}} 0} \right)^{-1} \\ &= \left(1 - e^{-\lambda_{\text{short}} r_t^*} \right)^{-1} . \end{aligned} \tag{2.6}$$

Unmodeled obstacles are approximated with an exponential distribution since the farther from the LIDAR the beam gets, the less likely an unmodeled obstacle is to be encountered. To justify this, consider a scenario in which multiple unmodeled obstacles lie on the beam's path; only the closest would be detected by the LIDAR. Moreover, any unmodeled obstacles beyond the nearest modeled obstacle would also go undetected, so the distribution is truncated at r_t^*.

Measurement Failures Occasionally, laser range scanners fail to detect an obstacle. The majority of such cases are simply due to all obstacles being beyond the detection range of the LIDAR, however there are other scenarios that result in failed measurements. Since the measurements depend on reflected light, any material that fails to reflect the light back to the LIDAR will go undetected. In practice, the only material observed that reliably failed detection by the LIDAR within the operating space was the carbon-fiber of the airship's hoops, though glass is often cited as another material not observable by LIDARs. The LIDAR interprets no reflected light as no obstacle and returns its maximum distance measurement, thus these errors are modeled as a spike at the maximum range:

$$p_{\max}(r_t \mid x_t, m) = \begin{cases} 1 & \text{if } r_t = r_{\max} \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

Strictly speaking, eq. (2.7) does not represent a probability distribution, but is treated as one for the sake of simplicity, as in [50].

Random Measurements The fourth type of measurement noise encompasses any other random, spurious measurements returned by the LIDAR. These are modeled as a simple uniform distribution over the whole LIDAR range:

$$p_{\text{rand}}(r_t \mid x_t, m) = \begin{cases} \frac{1}{r_{\max}} & \text{if } 0 \le r_t \le r_{\max} \\ 0 & \text{otherwise.} \end{cases} \tag{2.8}$$

Finally, the four types of sensor noise described above are combined as a weighted sum:

$$p(r_t \mid x_t, m) = \begin{pmatrix} \alpha_{\text{hit}} \\ \alpha_{\text{short}} \\ \alpha_{\max} \\ \alpha_{\text{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} p_{\text{hit}}(r_t \mid x_t, m) \\ p_{\text{short}}(r_t \mid x_t, m) \\ p_{\max}(r_t \mid x_t, m) \\ p_{\text{rand}}(r_t \mid x_t, m) \end{pmatrix} \tag{2.9}$$

where the four α values must sum to one and are determined empirically.

Noise Model Parameter Values All told, the LIDAR noise model requires six parameters – σ_{hit} from eq. (2.2), λ_{short} from eq. (2.5), and the four α weights of eq. (2.9) – and each of the parameters may be a function of the expected obstacle distance, r^*. To determine the parameters, over 600,000 distance measures spanning the LIDAR's entire detection range were taken with the LIDAR at known positions within a typical environment. Not all of the obstacles in the environment were modeled to account for unmodeled obstacles in the operating environments. The LIDAR's scan range was discretized into 0.1 m segments of the expected (as opposed to measured) distances. For each segment, the six parameters were jointly calculated using an iterative solver that found the best fit for the six parameters to the measured data.

It was found that λ_{short} and the four α parameters were all independent of measurement distance, while the value of σ_{hit} was a function of r^*:

$\sigma_{\text{hit}} = 0.01r^* + 5$ mm	$\lambda_{\text{short}} = 0.0005$	$\alpha_{\max} = 0.005$
$\alpha_{\text{hit}} = 0.98$	$\alpha_{\text{short}} = 0.01$	$\alpha_{\text{rand}} = 0.005$

2.3.4 LIDAR and IMU Wireless Communication

Both the LIDAR and the IMU are designed to communicate using the RS-232 serial communication protocol. Of course, when mounted to the airship, a direct serial cable connection to the ground-station PC is impossible. To achieve wireless communication with the sensors, two pairs of Roving Networks FireFly Bluetooth serial adapters were used. One such transceiver is shown in Fig. 2.3(d). These gumstick-sized adapters allow transparent serial communication over a Bluetooth connection.

Wireless Communication Transmission Rates

The Bluetooth transceivers do incur a reduction in transmission rates that must be taken into consideration. When transmitting all available data, the LIDAR saturates a serial connection operating at a baud rate of 115.2 kbps (the maximum rate common to the LIDAR and the host PC) such that new data is only available at a frequency of 8 Hz. If the amount of data is reduced, for example by specifying that the LIDAR send averages of pairs of distances as described above, then the nominal LIDAR frequency of 10 Hz can be reached. Transmitting this reduced data set over the Bluetooth connection using a naïve approach – only asking for new data once all previous data has arrived – reduces the frequency to 5 Hz. Even reducing the amount of data by half again shows no improvement.

This performance reduction is caused by a combination of the Bluetooth transceivers and the LIDAR itself. Once started, the spinning mirror in the LIDAR operates at a constant rate of 10 rotations per second. The LIDAR computes distances for every revolution and transmits the distances if data has been requested by the host PC. If the LIDAR receives a request for data during a revolution, it waits until the start of the next revolution before sending measurements. For their part, the Bluetooth transceivers introduce a delay into the system since they must first establish that the transmission channel is clear prior to transmitting data. Thus, after transmitting all of the LIDAR data to the host PC and only then sending the next request for data to the LIDAR, the next revolution of the mirror has already begun and the LIDAR must wait until the following data set can be gathered. As a result, the LIDAR effectively operates at half of its rated frequency.

To compound the problem, if the LIDAR receives a request for data prior to calculating all of the distances of the previous request, it returns an error, so it is not possible to send multiple requests for data in quick succession. There is only a small window, approximately 1/30 s, in which a new request for data can be sent to be able to receive sequential data sets.

Performance was improved with a more efficient communication sequence. By sending a request for new data as soon as the initial LIDAR response begins arriving at the host PC, the request usually arrives in the allowable window to catch the following data set, though not every time. This is where the asynchronous threads possible with the QuaRC toolbox become indispensable. Without them, the uncertainty introduced by the Bluetooth transceivers would force the system to run at 5 Hz.

To evaluate the new communication sequence, a series of tests were run, the results of

Table 2.2 LIDAR sampling periods: Minimum periodic sample times that ensure new data at several levels of certainty when communicating over a serial cable and using two communication sequences over the Bluetooth transceivers.

		Minimum periodic sample time [s]		
Cluster size	Reliability [%]	Serial cable	Bluetooth, naïve	Bluetooth, efficient
1	100	0.14	0.34	0.21
	99	0.14	0.31	0.18
	95	0.13	0.29	0.16
2	100	0.10	0.23	0.20
	99	0.10	0.21	0.12
	95	0.10	0.19	0.10
3	100	0.10	0.24	0.21
	99	0.10	0.21	0.13
	95	0.10	0.19	0.12
4	100	0.10	0.23	0.19
	99	0.10	0.21	0.13
	95	0.10	0.20	0.11

which are summarized in Table 2.2. Each test consisted of constantly communicating with the LIDAR for 30 minutes and recording the exact times at which full LIDAR data sets became available to the host PC. A total of twelve trials were run, four each for the serial cable connection, naïve Bluetooth communication, and efficient Bluetooth communication. The four trials run for each communication sequence were for commanding the LIDAR to send different amounts of data: every individual data point and averages of two, three, and four data points. The minimum periods were then calculated for scenarios in which new LIDAR data would be available for the controller with 100%, 99% and 95% reliability.

Clearly, the Bluetooth transceivers do introduce delays, however they can be largely overcome by using the efficient data transmission sequence and programming the controller such that it is able to cope with occasionally not having new data available. Commanding the LIDAR to send averages of more than two data points shows no improvement in transmission rates and reduces the precision of the measurements, so averages of two distances are optimal. Allowing for only 99% reliability, the controller could be run with a period of 0.12 s (frequency of 8.3 Hz) when communicating wirelessly.

Finally, it should be noted that the amount of data sent by the IMU was small enough that the Bluetooth transceivers did not significantly affect performance. As well, operating both the LIDAR and the IMU simultaneously using separate pairs of Bluetooth transceivers resulted in no further reductions in performance.

Chapter 3

Optical Flow for Navigation

Vision-based robot navigation is a natural progression of technology, since we as humans rely tremendously on our vision for navigation and have manufactured our environment around this. Any robot that is expected to operate in our environment would thus be greatly served by the ability to navigate by sight. Relation to our own navigation systems is not the only attractive aspect of vision-based navigation, however. Modern digital cameras are able to provide immense amounts of data from very small, light-weight, and relatively low-power packages. Another advantage of vision, particularly for military applications, is that cameras are typically passive, self-contained sensors, unlike most other exteroceptive sensor systems. That is, cameras can sense their environment without broadcasting their presence. It was for these reasons, particularly the size and weight aspects, that a vision-based navigation system was sought for the airship.

3.1 Review of Vision-Based Robot Navigation

3.1.1 Mobile Robot Vision-Based Navigation

Any review of vision-based robot navigation techniques would be well served to begin with the excellent survey of the field by DeSouza and Kak [13]. They split the field into two broad categories – indoor and outdoor navigation – and then further subdivide each of those into various approaches. The focus here will be on indoor navigation, which the authors split into three sub-categories: map-based, map-building, and mapless navigation.

One approach to map-based vision navigation is absolute localization, where the vision system attempts to locate known, static landmarks and determine the robot's position within the map based on the relative positions of the landmarks. The greatest drawback here, of course, is that if not enough landmarks are visible, the robot cannot fully specify its location. A similar approach is to match features in the observed images to features in a stored database of mapped images. Probabilistic localization is the typical approach here, as the feature matching may not be unique (e.g., there may be many doors in a building that all look identical). The third map-based localization approach expands on this probabilistic localization. Rather than relying on a database of mapped images, the robot is given a map (typically topological) of its environment and localizes itself within the map by determining the positions of physical features of the environment, such as corners at adjoining walls, that can be matched to the map.

In all three of the above techniques, it was generally assumed that the camera could not take on any arbitrary pose. For example, in feature matching, the features were often assumed to be at a known distance and orientation from the camera, such as a line marked on the floor over which the robot drives, or that the camera is facing normal to the robot's motion so that it can view landmarks on a wall which is at a constant distance from the robot. In the third approach, the cameras' orientations relative to the ground were fixed so that vertical environment features could be reliably located.

At the time of writing their paper [13], vision-based map-building navigation was still an immature field. One approach was to use vision in coordination with other sensors to build a 2-dimensional occupancy grid of the environment that dictates where the robot may and may not travel. These fare poorly in large, complex environments due to inaccuracies in odometry and sensor uncertainties. The other approach discussed was building topological representations of space with identifiable nodes. Node definitions varied by implementation, but the common difficulty was reliably identifying previously visited nodes.

In the intervening years since the publication of [13], vision-based simultaneous localization and mapping (SLAM) has been a popular research area. The most promising recent work has been in the area of real-time single camera SLAM. Davison et al. [11] showed that it was possible to build a stochastic map of a cluttered environment by extracting salient features from the video stream. Their system was dubbed MonoSLAM. Unlike other SLAM systems (e.g. [31, 47]) MonoSLAM required no learning step prior to autonomous operation. The major drawback of this system was that the complexity of the computations greatly increased as the size of the environment was increased – at the rate of $O\left(N^2\right)$. The robot was effectively restricted to operation within a single room. To overcome this challenge, Eade and Drummond [14] combined MonoSLAM with the FastSLAM algorithm of [38] to enable the system to scale to larger environments while still working in real-time.

Other vision-based SLAM systems have been implemented as localization algorithms in mobile robots as well. Cuperlier, Quoy and Gaussier [10] used a SLAM approach to build and navigate a cognitive map – a map of locations of interest joined by paths, as opposed to a metric map that contains absolute positions of objects and locations. The robot was ground-based and equipped with a 360° camera. A neural-network based algorithm was used to extract landmarks and develop the map, which was built in a learning step prior to autonomous navigation.

Returning to the survey by DeSouza and Kak [13], their third indoor vision-based navigation category was mapless navigation. One such technique is similar to the map-based technique of matching image features to known images, however in this case the known features are not located within a map. Rather, they are generic features of an environment, such as doorways, that give clues to the robot on how to proceed, or not, in the case of walls. On their own, such systems are typically relegated to just roaming the environment, however the idea can be expanded to include more general object recognition systems that are able to interpret commands, such as "go to the door," where the robot would then navigate itself to the door based on what it sees.

The final method of indoor vision-based navigation discussed in [13], optical flow, operates at a lower level than those described above. Rather than attempting to correlate image features to known objects or environmental features, optical flow based controllers attempt to navigate purely on the apparent motion seen in a sequence of images, without regard to what is actually seen in the images. Optical flow techniques will be discussed in much greater detail in §3.2.

3.1.2 Aerial Robot Vision-Based Navigation

In all of the work described in the preceding section, there was not a single mention of aerial robots. Making the jump from ground-based to aerial robotics increases the complexity of the task tremendously: the degrees-of-freedom of the system double, from three to six; direct physical odometry, while unreliable for ground-based robots, becomes entirely impossible; weight restrictions often severely limit the types and amounts of sensing and computation equipment that may be used, for both weight and power reasons; and failure of the robot can be potentially catastrophic, e.g., if a fixed-wing aerial robot loses control, it will likely crash destructively. With these extra challenges, it is not surprising that early research on robotic vision systems focused on ground vehicles. That is not to say that there has been no research on aerial robotics with vision-based navigation.

Work on vision-based navigation for autonomous airships began in the late 1990s. In [7], an autonomous indoor blimp was developed that visually tracked a target with known geometrical and visual properties. By applying basic image processing and tracking techniques, it was possible to fully determine the position and orientation of the airship relative to the target. While effective, tracking a known target is not an ideal solution, since the airship is only able to operate within visual range of the target, severely restricting its possible motions. A slightly different approach with similar results was undertaken in [62]. In that case, visual servoing was performed to control an indoor blimp. A floating ball was tracked, but rather than determine the position of the ball in Cartesian coordinates and from that derive the pose of the airship, all operations were performed in the image plane. Further, the dynamics of the airship were also calculated in the image plane. Of course, this method suffers from the same drawbacks as any method that only tracks a known object. These concepts were generalized somewhat by the work of van der Zwaan, Bernardino and Santos-Victor [57]. Station keeping and docking of an indoor blimp was achieved by visually tracking an image feature, however that feature was not known *a priori* but was specified by the operator at runtime.

Several more advances in vision-based navigation of airships soon followed. Using insect-inspired optical flow algorithms, an indoor blimp was able to autonomously navigate to a goal, avoid obstacles and stabilize its course [25]. This was all achieved using a single 360° camera. This work was later expanded on to achieve odometry from the same data [26]. Another team [66] followed similar ideas of biologically-inspired computer vision for the control of an indoor blimp using genetic algorithms to evolve the controller. Their airship's goal was to maximize speed and no explicit goal of wall avoidance was given, yet the airship learned to avoid walls to maximize speed. More recently, Kawamura et al. [29] used a learning controller for the vision-based control of an indoor airship – the cylindrical airship mentioned in §1.3.1. Their controller learned what outputs to give based on the current and desired state variables. The downward-facing vision system tracked a known pattern on the floor of the environment to determine the airship's position and orientation. The same, airship had previously been used to develop vision-based landing using a PID controller, also estimating pose using the pattern on the floor [49].

Vision-based navigation on aerial vehicles other than airships has also been researched. Oh, Green and Barrows [41] demonstrated obstacle avoidance and landing maneuvers for a fixed-wing micro-air-vehicle. They achieved this using two specialized 1-dimensional optical flow sensors, one facing the direction of motion and the other facing the ground. Further, they used a neural-network trained to deal with various lighting conditions. Beyeler et al. [4] simulated a similar platform along with basic inertial and airspeed sensors to determine the altitude and pitch of the vehicle in a simulated environment.

More recently, rotor-craft have become popular research platforms, quadrotor aircraft in particular. An interesting approach to vision-based control of a quadrotor was presented in [52]. The controller tracked a known target on the floor for relative position, as many others have done, however the creative use of Moiré patterns allowed finer-grained measurements of translation and yaw rotation. In [6], a quadrotor was again controlled by tracking known target on the ground. In this case the target was much simpler, but multiple control algorithms were investigated that were all based on visual servoing. The authors concluded that several of the investigated algorithms were all acceptable for this type of control, suggesting that there is no one “right way” of performing vision-based control of aerial robots.

Others have used methods similar to those discussed in §3.1.1 for rotor-craft navigation. Courbon et al. [9] pre-recorded a set of key images along a path for a quadrotor to refer to and be able to navigate the same path. Çelik et al. [8] used a standard monocular vision system for a FastSLAM-like algorithm for navigation of a small helicopter down a corridor. The landmarks they tracked were architectural features of the corridor, much like the third map-based localization approach described in [13].

Finally, optical flow based techniques have not been ignored either. Ahrens et al. [1] used a pyramidal Lucas-Kanade optical flow algorithm for feature tracking along with ego-motion estimation from an IMU to perform SLAM and obstacle avoidance with a quadrotor. Optical flow was also used for corridor navigation in [64]. A wide-angle camera recorded images of the corridor walls on both sides and the relative flow fields were used to maintain the quadrotor at the center-line of the corridor. Kendoul, Fantoni and Nonami [30] proposed a control algorithm for a quadrotor based on optical flow, an IMU and a nested extended Kalman filter. In indoor and outdoor flight tests, autonomous take-off, hovering and landing of the quadrotor was achieved with a few meters of drift.

As a final note, in none of the above listed works was a true holonomic 6-degree-of-freedom robotic motion achieved with an aerial platform using a vision-based system. A vehicle capable of such motion would provide a unique test-bed for developing robust navigation and control algorithms.

3.2 Optical Flow

Since none of the vision-based navigation techniques discussed in §3.1 have yet been implemented on a holonomic 6-DOF aerial platform, there was no clear selection that could be made for implementation on such a platform. Optical flow was selected for its relative simplicity and since basic navigational tasks have been performed using optical flow without knowledge of absolute position or orientation of the robots. For example, station keeping [30], obstacle avoidance [1, 41, 65], corridor navigation [64], terrain following [23], altitude and pitch estimation [4, 65], visual odometry [26], and even aircraft landing [41, 56] have all been proposed using optical flow techniques as their basis.

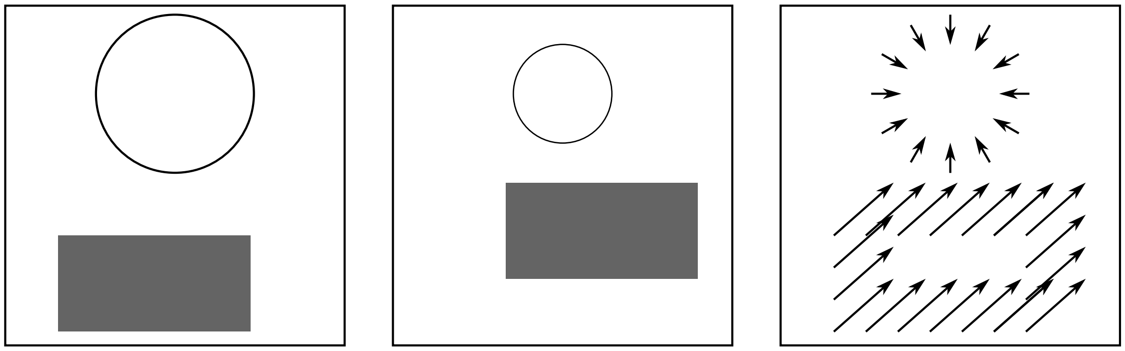


Figure 3.1 An example of optical flow: Two sequential images of a simple scene and a subset of the flow vectors between the two images. The shaded rectangle undergoes pure translational motion, while the circle could be interpreted as either shrinking or moving away from the camera.

3.2.1 The Visual Motion Field and Optical Flow

The basis of optical flow is the visual motion field, which is defined as "the 2-dimensional vector field of velocities of the image points, induced by the relative motion between the viewing camera and the observed scene" [54]. The motion field is effectively the 3-dimensional vectors of the relative motion between the scene and the camera projected onto the camera's image plane. Optical flow is an approximation of the motion field, or the apparent motion of the image brightness over time.

Figure 3.1 shows two sample images and a subset of the flow vectors between them. Though the vectors shown could be drawn from either the visual motion field or the optical flow field, it is important to note that the two fields would not be identical, even for such a simple case as this. The rectangle has a uniform gray shade and a portion of the rectangle in the second image overlaps the rectangle in the first image. The change in brightness between the two images in the overlapping region remains constant, so the apparent motion of the image brightness (the optical flow) would be zero for this region. The visual motion field, however, would be dense with equal vectors within the rectangle.

The key assumption that must be made for calculation of optical flow is that the apparent brightness of objects in the scene remains constant through motion and time, which can be expressed mathematically as

$$\frac{dI}{dt} = 0$$

$$\frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0$$

$$\nabla I^T \cdot \mathbf{v} + I_t = 0 \tag{3.1}$$

where $I = I(x, y, t)$ is the image brightness, $\mathbf{v}$ is the motion field, and the subscript t denotes partial differentiation with respect to time. This equation has been dubbed the image brightness constancy equation [54]. The ultimate goal of all optical flow algorithms is to calculate $\mathbf{v}$ from a series of images.

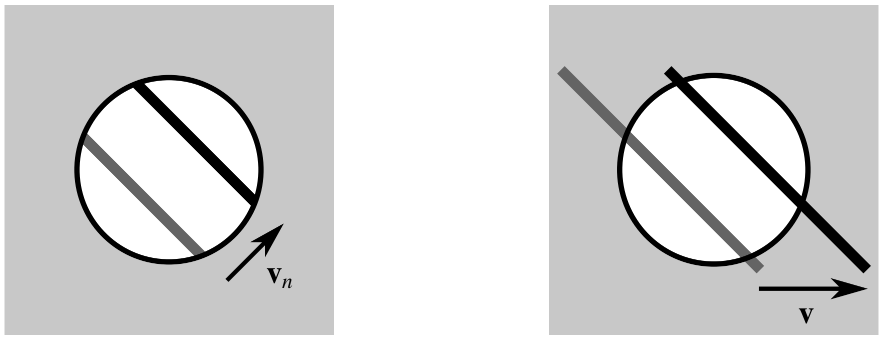
Many algorithms have been developed, using several different techniques, to estimate optical flow. Barron, Fleet and Beauchemin [3] performed a survey and evaluation of various techniques, including: differential techniques, which use spatio-temporal derivatives of image intensity; region-based matching techniques, which track corresponding regions in consecutive images; energy-based methods, which operate in the Fourier domain using velocity-tuned filters; and phase-based techniques, which use the phase behaviour of band-pass filter outputs. They implemented a total of nine algorithms, testing them with real and synthetic image sequences. They found the first-order differential techniques most reliable overall. Other optical flow algorithms have also been proposed more recently, such as the SIFT algorithm [34]. However, they are computationally expensive and the goal was for real-time control on standard computer hardware, thus were not implemented.

Two first-order differential optical flow algorithms were selected for implementation. The first was one of the original optical flow algorithms developed – the Lucas and Kanade algorithm [35] as described in [54]. The second algorithm, developed by Fleet and Langley [16], is unique for its use of recursive temporal filtering. It was selected specifically because of its performance in [3]. Both of these algorithms calculate optical flow based on spatio-temporal derivatives of the image intensity about small image areas. The details of each will be discussed in later sections.

The Aperture Problem

As per eq. (3.1), the 2-dimensional motion field is estimated using a single equation. Thus, only a single component of the motion field – that in the direction of the spatial gradient, v_n – can be determined [54]. This is known as the aperture problem, since it can be visualized by imagining looking at an image sequence through a narrow aperture, as depicted in Fig. 3.2. Since no uniquely identifiable points (i.e., corners) are visible within the small aperture, Fig 3.2(a), the true motion can not be recovered. Only the motion normal to the line of constant intensity can be calculated. Note that there are an infinite number of motions that would result in the same apparent velocity $\mathbf{v}_n$.

Notice that eq. (3.1) calculates the motion field at a single point based on the image gradient at the same point, rather than a small area as depicted in Fig. 3.2. As a result, the act of calculating flow based on a small region around the pixel, as is done in both of the selected algorithms, reduces the aperture problem by introducing multiple equations for



(a) When seen through a small aperture, the motion appears normal to the line.

(b) The actual motion was horizontal.

Figure 3.2 The aperture problem: The gray and black lines show the positions of the same object at two different times.

a single point's motion vector. The problem can still occur if a straight edge of uniform intensity crosses the whole region, as is the case in Fig. 3.2.

3.2.2 Lucas and Kanade: A Basic Optical Flow Algorithm

The Lucas and Kanade optical flow algorithm starts from the assumptions [54] that:

1. the apparent brightness of objects in the image remains constant through motion and time; and
2. within any small patch of the image, the motion field can be approximated by a constant vector field.

The first assumption is just the image brightness constancy equation (3.1). If the above conditions are applied point-wise to all pixels $\mathbf{p}_i$ in an $N \times N$ region Ω of the image, where the spatio-temporal derivatives are calculated at each of the pixels, then the optical flow can be computed as the minimization of

$$\sum_{\mathbf{p}\in\Omega} W(\mathbf{p}) \left[\nabla I^T(\mathbf{p}, t) \cdot \mathbf{v} + I_t(\mathbf{p}, t)\right]^2 \tag{3.2}$$

where $W(\mathbf{p})$ is a weighting function to give pixels near the center of the region greater influence [3]. The solution to eq. (3.2) is then found by solving the linear system:

$$A^T W A \mathbf{v} = A^T W \mathbf{b} \tag{3.3}$$

where

$$A = \left[\nabla I(\mathbf{p}_1, t), \quad \nabla I(\mathbf{p}_2, t), \quad \ldots, \quad \nabla I(\mathbf{p}_{N\times N}, t)\right]^T$$
$$W = \mathrm{diag}\left[w(\mathbf{p}_1), \quad w(\mathbf{p}_2), \quad \ldots, \quad w(\mathbf{p}_{N\times N}, t)\right]$$
$$\mathbf{b} = -\left[I_t(\mathbf{p}_1, t), \quad I_t(\mathbf{p}_1, t), \quad \ldots, \quad I_t(\mathbf{p}_{N\times N}, t)\right]^T.$$

Finally, so long as $A^T WA$ is nonsingular, solving the system of equations in eq. (3.3) yields [3, 54]

$$\mathbf{v} = \left(A^T WA\right)^{-1} A^T W\mathbf{b}. \tag{3.4}$$

The matrix $A^T WA$ is a 2×2 matrix and $A^T W\mathbf{b}$ is a 2-element vector:

$$A^T WA = \begin{bmatrix} \sum WI_x^2 & \sum WI_xI_y \\ \sum WI_yI_x & \sum WI_y^2 \end{bmatrix} \tag{3.5a}$$

$$A^T W\mathbf{b} = \begin{bmatrix} \sum WI_xI_t \\ \sum WI_yI_t \end{bmatrix} \tag{3.5b}$$

where the subscripts x and y denote partial differentiation of the image intensity in the respective directions, and all of the sums are taken over the N^2 points in the region Ω [3].

3.2.3 Fleet and Langley: Recursive Temporal Filtering

The recursive temporal filtering algorithm by Fleet and Langley [16] builds on the Lucas and Kanade algorithm. Typically, images used for optical flow calculations are spatio-temporally filtered to reduce noise. If these filters are assumed to be separable in space-time, then the resulting filtered image at time t is

$$R(\mathbf{p}, t) = E(t) * [B(\mathbf{p}) * I(\mathbf{p}, t)] \tag{3.6}$$

where $E(t)$ is the temporal filter and $B(\mathbf{p})$ is the spatial filter. Fleet and Langley took the temporal filter to be a low-pass causal filter

$$E(t) = \tau_1 \mathrm{e}^{-\tau_1 t}, \quad t \geq 0 \tag{3.7}$$

where τ_1^{-1} is the time constant that determines the length of time each image frame affects the temporal filter, known as its temporal support duration.

To alleviate noise and temporal aliasing problems (see [16] for details), the temporal filter can be convolved repeatedly to form a simple cascading filter with useful properties:

$$E_n(t) = [E(t)]^{*n} = \frac{(t\tau_1)^{(n-1)}}{(n-1)!} E(t) . \tag{3.8}$$

where the $*n$ superscript denotes n convolutions. Using eq. (3.8) with eq. (3.6) and taking the temporal derivative gives

$$\frac{\partial R_n(\mathbf{x}, t)}{\partial t} = \frac{dE_n(t)}{dt} * [B(\mathbf{p}) * I(\mathbf{p}, t)] . \tag{3.9}$$

It can be shown [16] that for $n \geq 2$ the temporal derivative of the temporal filter is a weighted difference of $E_{n-1}(t)$ and $E_n(t)$:

$$\frac{dE_n(t)}{dt} = \tau_1 [E_{n-1}(t) - E_n(t)] , \quad n \geq 2, \tag{3.10}$$

allowing the same filter to easily be used for both temporal filtering and temporal differentiation, reducing computational costs.

Discretization of the Temporal Filter

The recursive temporal filter in eq. (3.10) is still in the continuous time form. The details of the discretization will be omitted here, see [16], but the resulting discretized, low-pass filter with $n = 3$ and t now a discrete variable is broken up into multiple steps. First, for clarity, take $q = \frac{\tau_1}{(\tau_1+2)}$ and $r = \frac{(\tau_1-2)}{(\tau_1+2)}$. Then, the first stage of the cascade is

$$\begin{aligned} w(t) &= I(t) - 2rw(t-1) - r^2 w(t-2) \\ R_2(t) &= q^2 w(t) + 2q^2 w(t-1) + q^2 w(t-2) \end{aligned} \tag{3.11}$$

and the second stage is

$$\begin{aligned} y(t) &= R_2(t) - ry(t-1) \\ R_3(t) &= qy(t) + qy(t-1) \end{aligned} \tag{3.12}$$

and finally, the temporal derivative is given by

$$R_t(t) = \frac{dR_3(t)}{dt} = \tau_1 [R_2(t) - R_3(t)] . \tag{3.13}$$

Application to Optical Flow

To use eqs. (3.12) and (3.13) to calculate the flow field, first generate a set of intermediate images:

$$\begin{aligned} &R_x^2(\mathbf{p},t), \quad R_y^2(\mathbf{p},t), \quad R_x(\mathbf{p},t)\,R_y(\mathbf{p},t), \\ &R_x(\mathbf{p},t)\,R_t(\mathbf{p},t), \quad R_y(\mathbf{p},t)\,R_t(\mathbf{p},t). \end{aligned} \tag{3.14}$$

where $R = R_3$ from eq. (3.12) and subscripts again denote partial differentiation. These intermediate images are then convolved with a weighting matrix W to form

$$\overline{A}(\mathbf{p},t) = \begin{bmatrix} \sum WR_x^2 & \sum WR_xR_y \\ \sum WR_yR_x & \sum WR_y^2 \end{bmatrix} \tag{3.15a}$$

$$\overline{\mathbf{b}}(\mathbf{p},t) = \begin{bmatrix} \sum WR_xR_t \\ \sum WR_yR_t \end{bmatrix} \tag{3.15b}$$

where, as before, the sums are taken over all of the pixels in the local region Ω [16]. Note the similarity to eqs. (3.5). The above values are then augmented with the temporal accumulation of constraints by the following

$$A(\mathbf{p},t) = \alpha A(\mathbf{p},t-1) + (1-\alpha)\,\overline{A}(\mathbf{p},t) \tag{3.16a}$$

$$\mathbf{b}(\mathbf{p},t) = \alpha \mathbf{b}(\mathbf{p},t-1) + (1-\alpha)\,\overline{\mathbf{b}}(\mathbf{p},t) \tag{3.16b}$$

where $\alpha = \exp(-\tau_2)$ and τ_2^{-1} is the time constant for the temporal support window [16]. Finally, when $A(\mathbf{p},t)$ is nonsingular, the optical flow is calculated as the solution to the resultant set of normal equations

$$\begin{aligned} A(\mathbf{p},t)\,\mathbf{v} &= \mathbf{b} \\ \mathbf{v} &= [A(\mathbf{p},t)]^{-1}\,\mathbf{b}(\mathbf{p},t). \end{aligned} \tag{3.17}$$

3.3 Optical Flow Implementation and Testing Setup

The algorithms described in the previous section were discussed purely on a theoretical basis. Several considerations must be taken into account for implementation and testing.

3.3.1 Algorithm Implementation

Both optical flow algorithms were implemented in MATLAB. The implementations were fairly straight-forward from the algorithm descriptions with only one special note to make. The temporal filter of the Fleet and Langley algorithm assumes a set of previous values for every iteration, but there is no such set for the first image, and the paper outlining the algorithm does not specify what the authors used. For simplicity, the filter was initialized with all "previous" values set to zero.

The other concern for implementation of these algorithms is the selection of spatio-temporal filters. As mentioned briefly in §3.2.3, separable spatial and temporal filters are applied to the images to reduce the effects of image noise. The recursive temporal filter is of course the primary aspect of the Fleet and Langley algorithm, but the temporal filter for Lucas and Kanade and the spatial filters for both algorithms must also be specified. Based on the recommendations given in [16] and [54], all of these filters were implemented as Gaussian filters. The sizes and specifications of each will be listed in the following section.

Algorithm Parameters

The algorithms each require several parameters be set. Those parameters common to both algorithms were set equal for the sake of comparison. Further, the values of the parameters were selected based on the recommendations given in [16] and [54], which conveniently gave the same suggestions for the common parameters. The common parameters were: the size of the region Ω in eq. (3.2), which was set to be a 5×5 window; the weighting matrix $W(\mathbf{p})$ of eq. (3.2), set as a Gaussian distribution with a standard deviation of $\sigma_W = 1.2$; the spatial filter – also a 5×5 Gaussian, but with a standard deviation of $\sigma_s = 1.5$; and the spatial derivative kernel, where a second derivative central difference approximation was selected, $g_s = \begin{bmatrix} -1 & 8 & 0 & -8 & 1 \end{bmatrix}/12$.

The only other parameters required for the Lucas and Kanade algorithm related to the temporal filter. Based on the recommended values in [54], the Gaussian temporal filter had a size of 5 images and a standard deviation of $\sigma_t = 1.5$. The temporal derivative was calculated using the same second derivative central difference approximation as the spatial derivatives.

The recursive temporal filter of the Fleet and Langley algorithm requires two further parameters. The first time constant from eq. (3.10) was set as $\tau_1^{-1} = 1.25$, while the second time constant from eqs. (3.16) was set indirectly such that $\alpha = \exp(-\tau_2) = 0.3$. As with the other parameters, these values were selected based on the results presented in [16].

Identification of Unreliable Flow Values

The optical flow algorithms themselves do not evaluate the estimates for reliability in any way. As a result, highly erratic flow vectors can be calculated in regions of low spatial contrast. An approach to evaluate the optical flow estimates was discussed in [3]. The matrices $A^T WA$ in eq. (3.3) (for Lucas & Kanade) and $A(\mathbf{p}, t)$ in eq. (3.17) (for Fleet & Langley) can be treated as covariance matrices for the optical flow estimates. To identify unreliable estimates, the two eigenvalues of the matrices are calculated and ordered $\lambda_1 \geq \lambda_2$ and their values inspected. The eigenvalues depend on the magnitudes of the optical flow estimates and their range of orientations. If the value of λ_2 for a particular flow vector is too small, then there is a high likelihood of that vector being unreliable. As in [3], a threshold of 1.0 was used. If $\lambda_2 < 1.0$, then the corresponding flow vector was omitted from the results.

3.3.2 Image Sequences for Testing

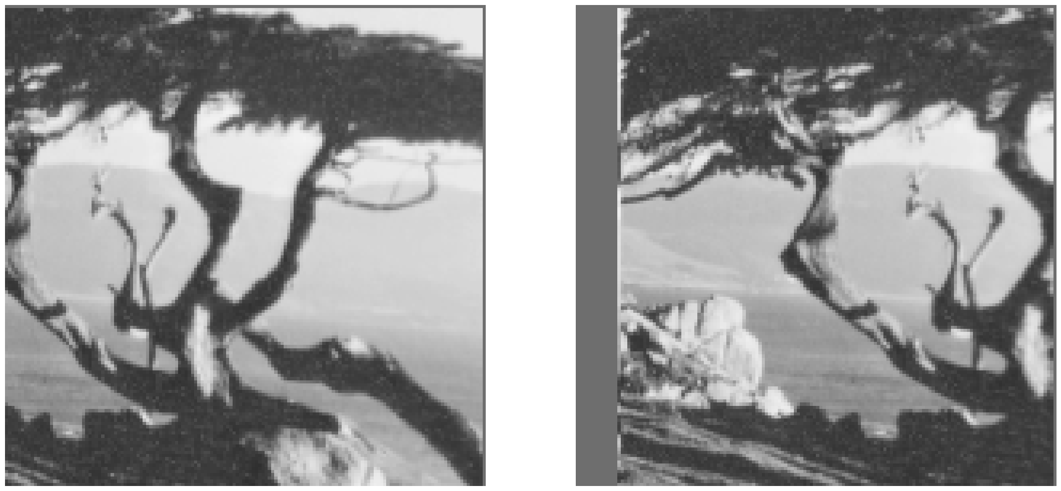
The optical flow algorithms were tested on three synthetic image sequences and two real image sequences. The first and final images of the synthetic image sequences are shown in Fig. 3.3. The first two synthetic sequences were the similar 40-image "Translating Tree" and "Diverging Tree" sequences[1] and the third was the 15-image "Yosemite Fly-through" sequence.[2] The translating tree sequence is a single image of a scene moved to the right two pixels per image. The diverging tree sequence is the same tree image with the camera moving closer to the scene at a constant rate along a path normal to the image plane. The Yosemite fly-through sequence was generated by projecting aerial images of Yosemite National Park in the United States onto a computer-generated model of the terrain. The camera "flies" through the simulated 3D space at a constant rate along a slightly curved path.

The three synthetic image sequences were all provided with the corresponding ground truth visual motion fields, which are shown in Fig. 3.4. All three show clean, regular motion fields that would be expected of synthetic image sequences.

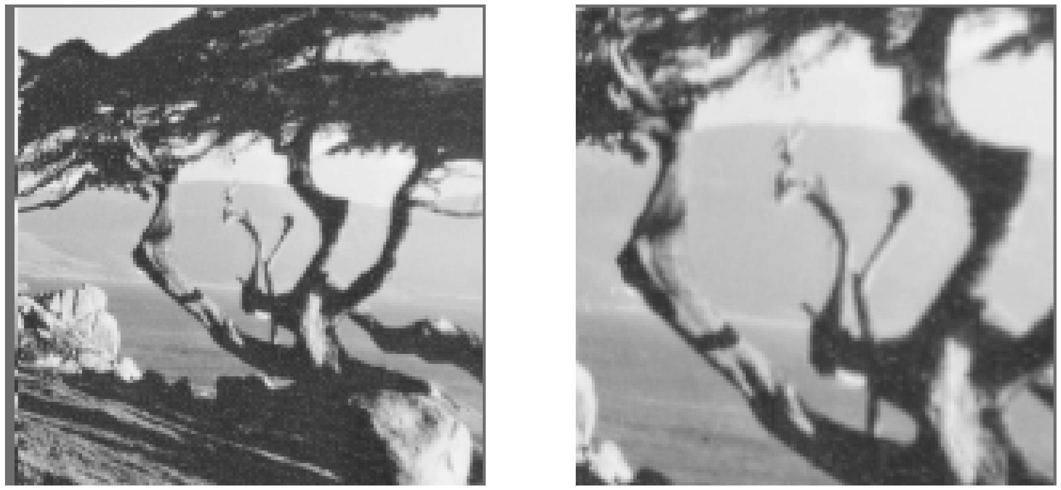
The ground truth fields, as well as all of the calculated flow fields in subsequent sections, have been displayed in such a manner as to maximize visual clarity. To that end only a subset of the full flow fields are shown. The full flow fields are dense with one flow vector per image pixel while those depicted are each restricted to 30 evenly-spaced columns of flow vectors and with the number of rows as large as possible to match the same pixel spacing. All of the flow fields for a given image sequence show the same subset of flow vectors,

[1]Both tree sequences by David J. Fleet and available at `http://www.fz-juelich.de/icg/icg-3/Mitarbeiter/Scharr/Testdata`.

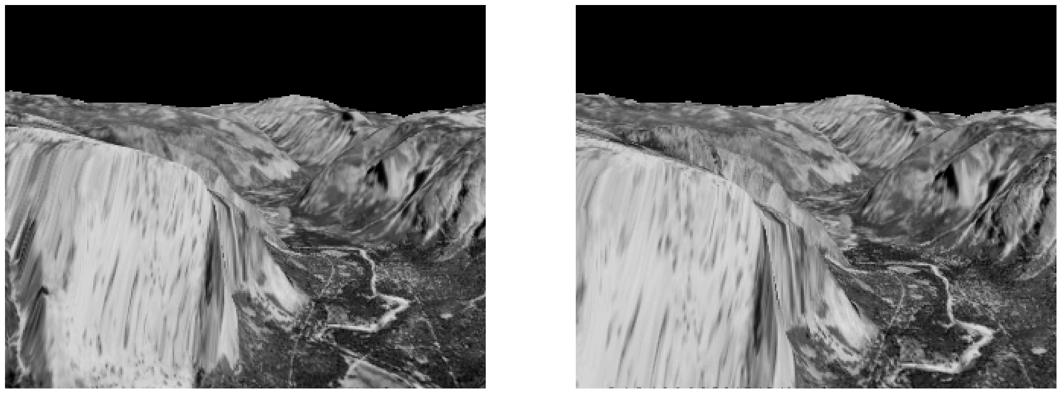
[2]The original Yosemite fly-through sequence was by Lynn Quam. The version used here was altered by Michael J. Black and is available at `http://www.cs.brown.edu/people/black/images.html`.



(a) The translating tree sequence. Camera motion is right-to-left parallel to the image plane.

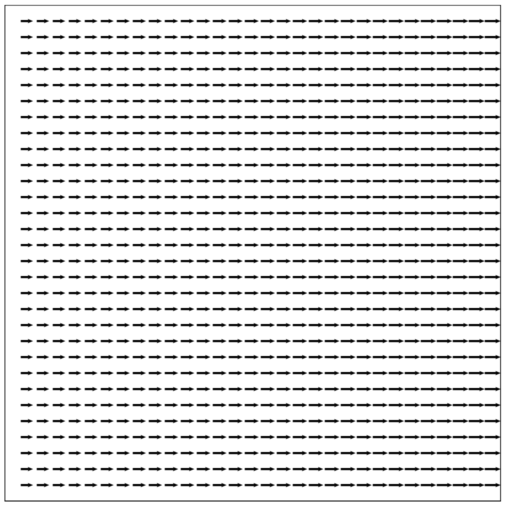


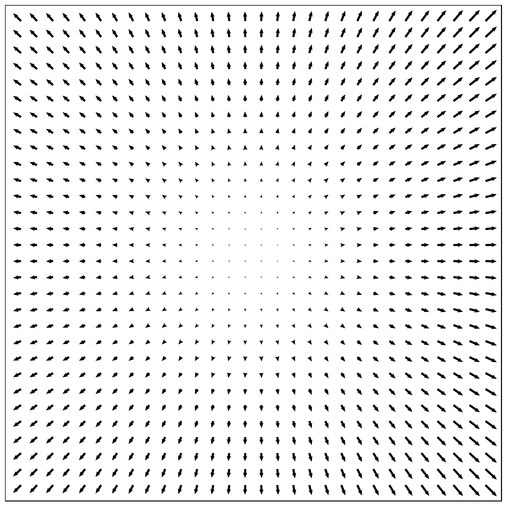
(b) The diverging tree sequence. Camera motion is toward the scene normal to the image plane.



(c) The Yosemite fly-through sequence. Camera motion is toward the scene on a slightly curved path veering right.

Figure 3.3 Synthetic image sequence samples: The first images of the three synthetic image sequences are on the left and the final images of the sequences are on the right.


(a) The translating tree sequence.


(b) The diverging tree sequence.

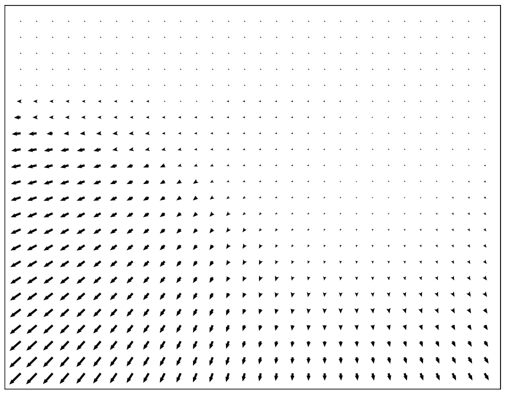

(c) The Yosemite fly-through sequence.

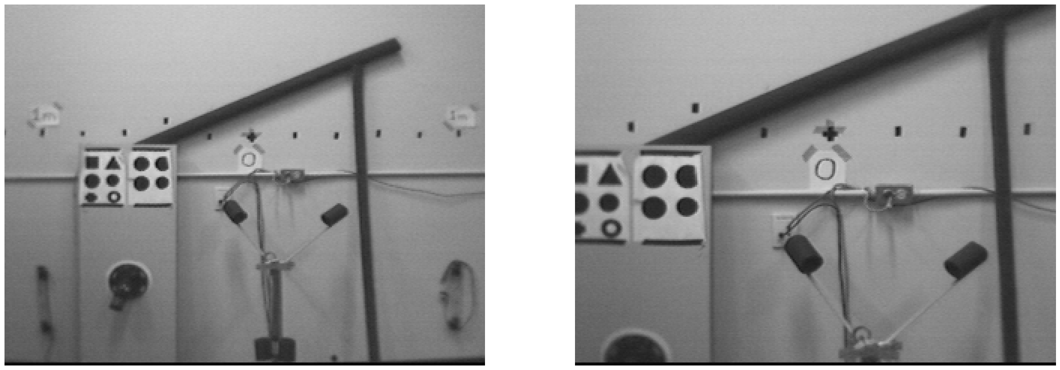
Figure 3.4 Synthetic image sequence ground truth visual motion fields. See the text for notes.

but the subsets vary between image sequences. Further, the flow vectors in the flow fields have been scaled from their true values. This was again done for clarity. For the synthetic image sequences the vectors were scaled such that the largest vector in the ground-truth flow fields was the length of the spacing between the flow vectors. In the case of the real image sequences where no such known ground-truth exists, an estimate was made so that as many of the flow vectors as possible showed reasonable sizes. All flow vectors in all flow fields of a given sequence were scaled by the same amount, but the scaling factors vary between image sequences. The majority of the flow fields also show dots in some locations where flow vectors should be. These dots represent several things: points where the actual optical flow value is zero; points where the flow vector would be smaller than the dot itself and thus too small to be clearly visualized; and points where the unreliable flow values were identified and removed, as discussed above. Finally, the flow fields are all restricted to being within the borders of the original images. In several cases the flow vectors erroneously exceed these borders, so the vectors have been cut off at the image borders.

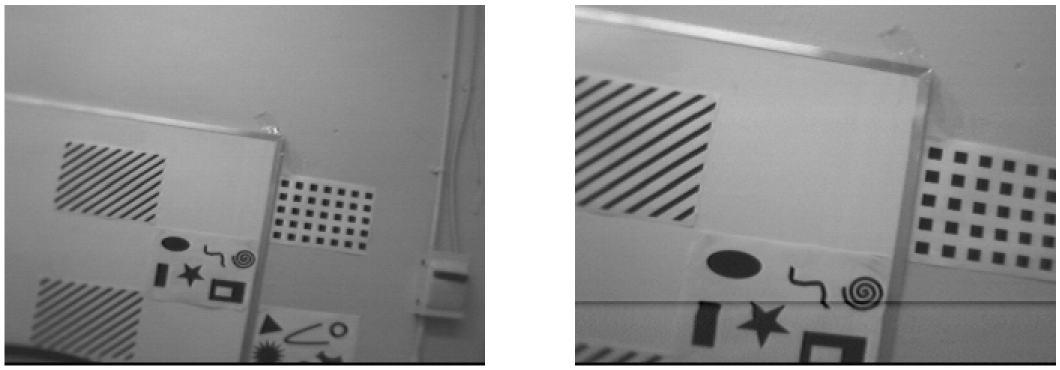
In addition to the synthetic image sequences, two real image sequences were recorded in the laboratory. The first and final images of these sequences are shown in Fig. 3.5. The first sequence was recorded by the wireless camera mounted to the robotic arm which was run along its track at constant speed. The camera's motion was smooth and along a straight line approximately normal to the image plane, toward a wall parallel to the image plane. The wall at which the camera was pointed was artificially cluttered to provide areas of high contrast for the image sequence. This sequence was relatively noise free. It will be referred to as the "track-mounted image sequence".

The second image sequence was created with the wireless camera mounted to the airship, and thus will be referred to as the "airship-mounted image sequence". The airship was flown from a steady position toward a wall. The goal was to get an image sequence similar to that of the track-mounted sequence, however the airship is a much less steady platform. As a result, there are rotational motions apparent in the sequence, the motion was not along a straight line, and the image plane is not parallel to the target wall. The wall was again artificially cluttered to provide areas of high contrast. Further, this sequence is not noise free like the others, with approximately 10% of the images displaying appreciable noise.[3] An example of the image noise is shown in Fig. 3.6. The resulting sequence is a closer approximation to the type of images that would be obtained during a true airship flight.

[3]The term "noise" in this context refers to variations in the image brightness values from the expected values, as opposed to the typical usage of the term in signal processing. Image noise can spike from one image frame to the next, while signal noise typically refers to variations from the true signal over time.



(a) Camera mounted on the track robot. Camera motion was along a straight line toward the observed wall, nearly normal to the image plane. This sequence had 80 images.



(b) Camera mounted on the airship. Camera motion was generally toward the wall but underwent significant oscillations and rotations. This sequence had 250 images.

Figure 3.5 Real image sequence samples: The first images of the two real image sequences are on the left and the final images of the sequences are on the right.

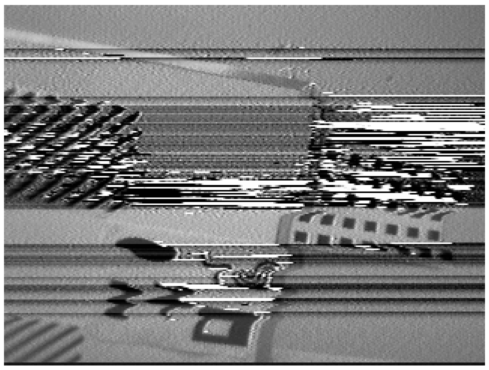


Figure 3.6 An example of the image noise from the airship-mounted image sequence.

3.4 Optical Flow Results

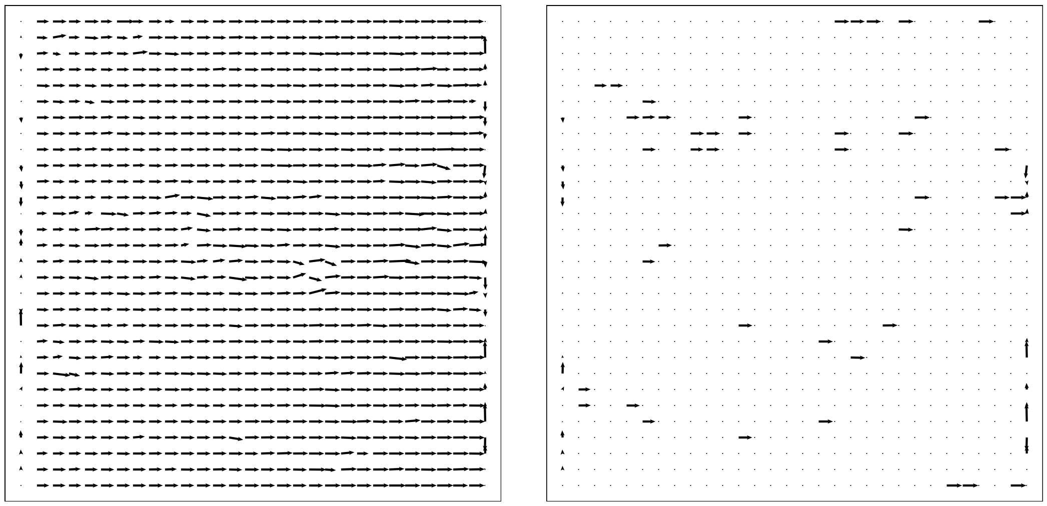
Qualitative analysis of the results of the two algorithms was performed for all five image sequences along with quantitative analysis of the flow fields calculated for the synthetic image sequences against their ground truths. The primary parameters of evaluation were the number of flow vectors that passed the thresholding test along with the magnitudes and directions of the flow vectors.

3.4.1 Performance with Synthetic Image Sequences

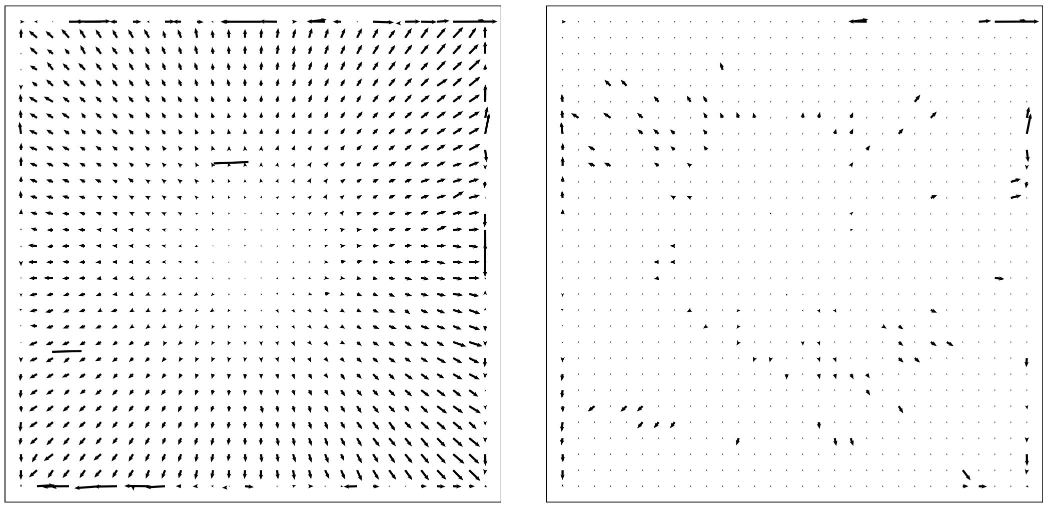
First, the results of the basic optical flow algorithm by Lucas and Kanade against the synthetic image sequences, which are displayed in Fig. 3.7. On the left are the full flow fields without any evaluation of the reliability of the results, while the fields on the right apply the reliability check described in §3.3.1. Based on visual inspection of the flow fields, it appears that the Lucas and Kanade algorithm is able to capture the flow reasonably well. Even before thresholding, the majority of the flow vectors are correctly oriented and sized, though the lower-left region of the Yosemite fly-through sequences shows more incorrect flow vectors. The thresholding applied to the right-hand images is far too strict for this algorithm, particularly in the case of the Yosemite fly-through sequence, but it was maintained for the sake of comparison with the Fleet and Langley algorithm. This does indicate that the algorithm may not be robust to more complex image sequences.

The recursive flow algorithm of Fleet and Langley also performed remarkably well on both tree sequences, as shown in Fig. 3.8. Even without applying thresholding to the values, almost perfect flow fields were returned. The thresholded flow fields match nearly perfectly the corresponding fields in [16]. Thresholding does show its worth with the Yosemite fly-through sequence, finally. Before applying thresholding, the lower-left region of the flow field showed enormous errors. This region of the images has very sparse points of high contrast, which the algorithms depend on. Most of the poor results were removed by the thresholding, leaving a very sparse region in the flow field. Even after thresholding, though, many of the remaining flow vectors were incorrect, both in magnitude and direction and the overall results are clearly not as good as for the two tree sequences.

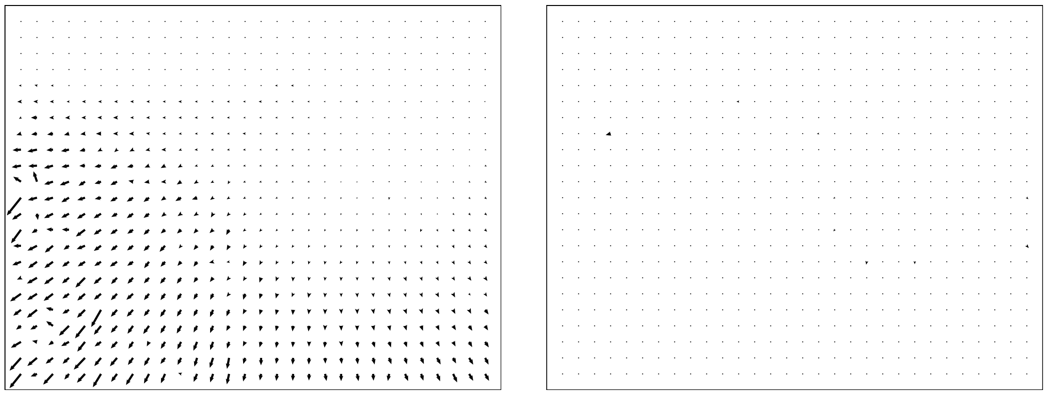
Since the synthetic image sequences were all provided with ground truth visual motion fields, quantitative comparisons could also be performed. Table 3.1 lists the magnitude and angle errors, along with the density of the thresholded optical flow fields calculated by both algorithms for all three synthetic image sequences. The errors were calculated for every flow vector that passed the thresholding test (not just those displayed in Figs. 3.7 and 3.8). Analysis of the un-thresholded results were omitted, though they were worse in nearly every



(a) The translating tree sequence.

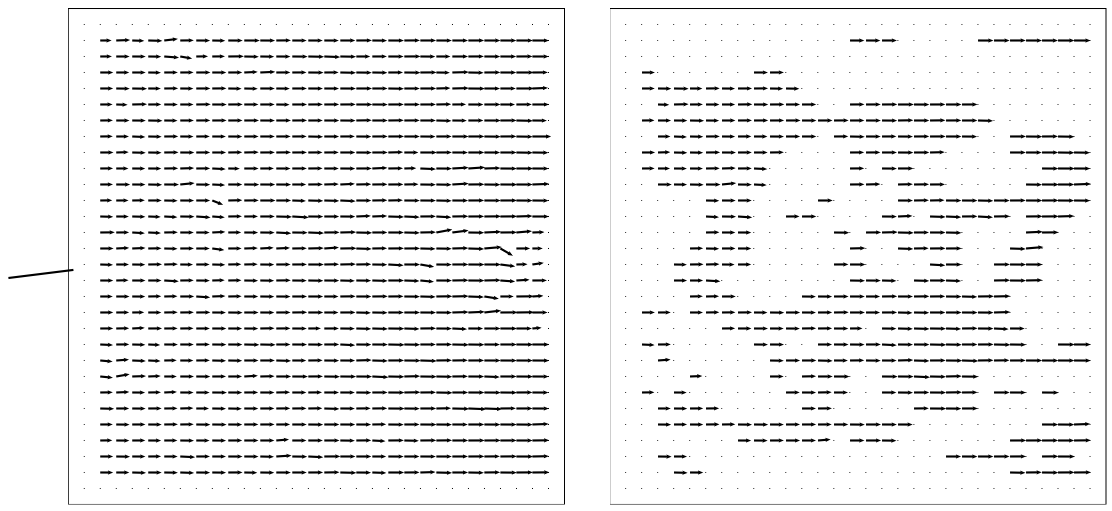


(b) The diverging tree sequence.

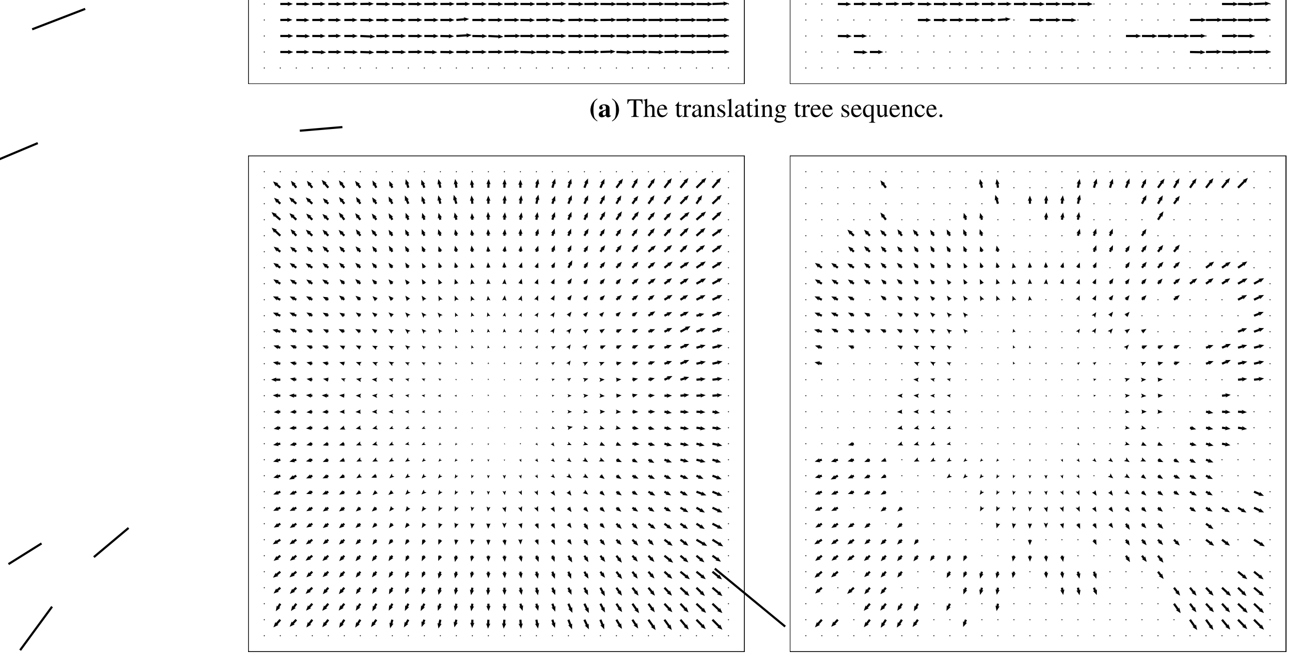


(c) The Yosemite fly-through sequence.

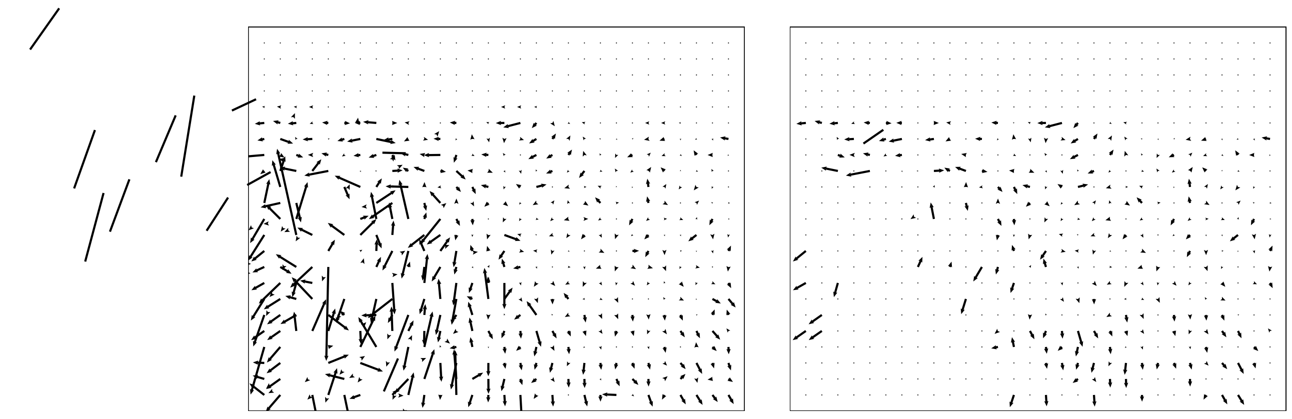
Figure 3.7 Lucas and Kanade applied to the synthetic image sequences: The fields on the left have not had reliability thresholding applied; those on the right excluded points with $\lambda_2 < 1$.



(a) The translating tree sequence.



(b) The diverging tree sequence.



(c) The Yosemite fly-through sequence.

Figure 3.8 Fleet and Langley applied to the synthetic image sequences: The fields on the left have not had reliability thresholding applied; those on the right excluded points with $\lambda_2 < 1$.

Table 3.1 Synthetic image sequence flow errors. These values correspond to the thresholded flow fields in Figs. 3.7 and 3.8.

			Abs. Magnitude Error [%]				Angle Error [°]		
		Dens.	Percentile				Percentile		
Sequence	Alg.	[%]	10th	50th	90th	100th	10th	50th	90th
Trans. Tree	L&K	5.75	0.20	1.42	67.67	99.38	0.04	0.36	86.85
	F&L	44.34	1.59	4.34	8.07	91.91	0.07	0.44	1.64
Div. Tree	L&K	11.78	0.49	2.89	24.26	254.3	0.27	1.58	43.24
	F&L	50.51	1.35	5.14	11.43	219.7	0.30	1.68	5.53
Yosemite	L&K	3.61	0.70	5.06	40.82	18143	0.56	2.92	26.16
	F&L	38.58	14.72	69.65	364.36	47861	4.54	36.88	143.96

respect except, naturally, the density of calculated flow vectors. The magnitude errors were calculated as absolute percentages of the ground truth flow vectors for the same image pixels, while the angle errors are the angles between the calculated flow vectors and the ground truth flow vectors, in the range [0°, 180°]. The densities are simply the percentage of image pixels at which optical flow was calculated compared to the number of image pixels at which the visual motion field is non-zero.

Numerical results for the Fleet and Langley algorithm against the two tree sequences were also presented in [16], however only flow densities and angle errors were shown. Further, the angle errors were presented as Gaussian distributions about non-zero means, while in reality, neither the angle errors nor the magnitude errors displayed this sort of distribution. Indeed none of the results corresponded to any standard statistical distributions, so only percentiles of the errors are provided.

Focusing first on the density of the flow fields, they reflect what is clear from the visual depictions: the Lucas and Kanade algorithm has far fewer vectors that survive the thresholding operation than the Fleet and Langley algorithm. The densities for the tree sequences calculated by the Fleet and Langley algorithm match those presented in [16]. Both algorithms fared poorest with the Yosemite fly-through sequence. The large area of poor spatial contrast in the lower-left region of the images is the primary culprit for this, once again showing that optical flow algorithms depend heavily on high spatial contrast.

Moving to the magnitude errors, it is clear that the performance of both algorithms varies immensely between image sequences and that the quantitative results tend to match the qualitative assessments described above. Both algorithms provide good results for the two tree sequences. The Fleet and Langley algorithm in particular showed exceptional results. For the translating tree sequence, with the vast majority of flow vectors having errors less than 10% of the true values. Results are similar for the diverging tree sequence, except that a small subset of the flow vectors for both algorithms displayed much larger errors with magnitudes three times the true values.

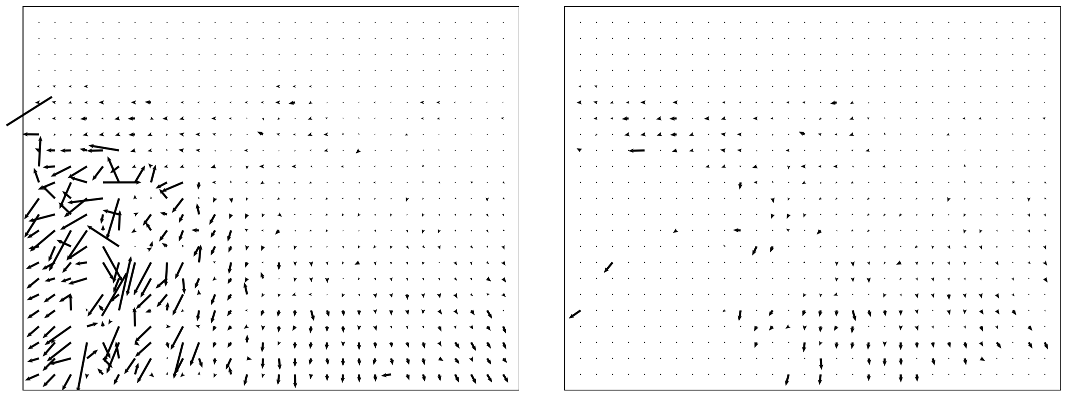
This trend is taken further with the Yosemite fly-through sequence. The majority of the flow vectors calculated by the Lucas and Kanade algorithm, as few as they were, still had reasonably small errors, but some had enormous errors with magnitudes over 180 times those expected. The Fleet and Langley algorithm was even worse. Nearly half of the vectors had errors of 70% or more with some reaching as high as 47000%. Recall that these results only include the vectors remaining *after* unreliable vectors had been removed. The algorithms are already showing weaknesses with only a slightly more complex image sequence.

The angle errors tell a similar tale. Both algorithms give excellent results with the two tree sequences, with the majority of flow vectors being less than half a degree from the true values for the translating tree sequence and less than two degrees for the diverging tree sequence. In light of these results, the 90th percentile values for the Lucas and Kanade algorithm on these two sequences seem unusually high, but are easily explained by the flow vectors along the borders of the images being calculated parallel to the image edges, as is clearly visible in Figs. 3.7(a) and 3.7(b). In practice, these sorts of erroneous vectors could be easily removed from the results, so are not a major concern. The results for the Yosemite fly-through sequence also follow the same pattern as the magnitude errors. Notably, the Fleet and Langley algorithm was far worse, with angle errors of half of the vectors greater than 36°. Signs of weakness are again showing with the more complex image sequence.

Time Dependence of the Fleet and Langley Algorithm

All of the above discussion of the Yosemite fly-through sequence has been based on the flow fields calculated after eight image frames, while those of the tree sequences were calculated after 20 frames. In all cases these corresponded with the mid-points of the image sequences. Since the Yosemite fly-through sequence showed poorer results than either of the others, it was worth inspecting the results for later images in the sequence. Fig. 3.9 shows the flow fields calculated by the Fleet and Langley algorithm after 10 and 15 frames of the sequence. The sequence is undergoing constant motion, so the ground-truth visual motion fields are the same as that shown in Fig. 3.4(c). It is immediately clear that with increased time the algorithm shows improved results. The directions of the flow vectors become much more uniform and the magnitudes settle down and become much less erratic.

These observations are reinforced by the quantitative error analysis listed in Table 3.2. While the number of vectors passing the thresholding test does not improve significantly with time, those that do pass improve immensely. After 10 frames the majority of the vectors


(a) The flow fields after the 10th image in the sequence.

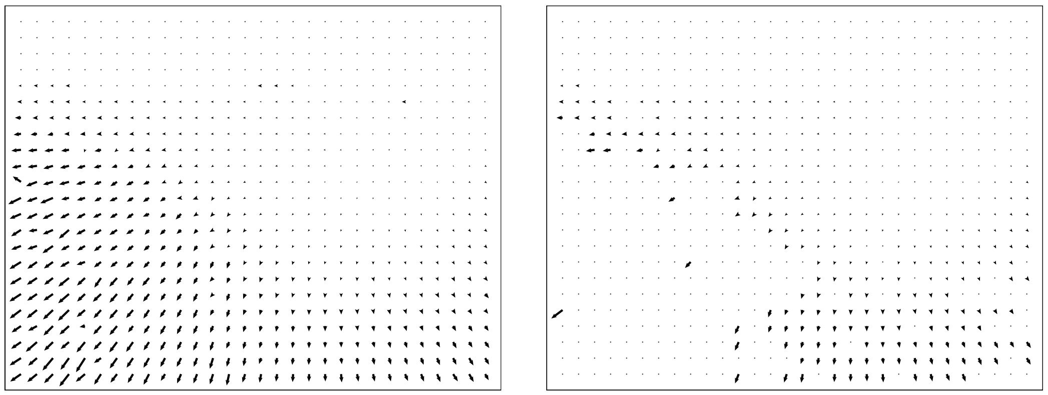

(b) The flow fields after the 15th image in the sequence.

Figure 3.9 Time dependence of Fleet and Langley: The flow fields for later images in the Yosemite fly-through sequence show improved performance. The fields on the left have not had reliability thresholding applied; those on the right excluded points with $\lambda_2 < 1$.

Table 3.2 Improved performance of Fleet and Langley with time. These values correspond to the thresholded flow fields in Figs. 3.7(c) and 3.9.

		Abs. Magnitude Error [%]				Angle Error [°]		
	Dens.	Percentile				Percentile		
Frame	[%]	10th	50th	90th	100th	10th	50th	90th
8	38.58	14.72	69.65	364.36	47861	4.54	36.88	143.96
10	39.15	6.48	29.52	102.45	7218	1.77	13.03	57.24
15	39.76	0.87	4.95	19.30	3527	0.44	2.32	8.44

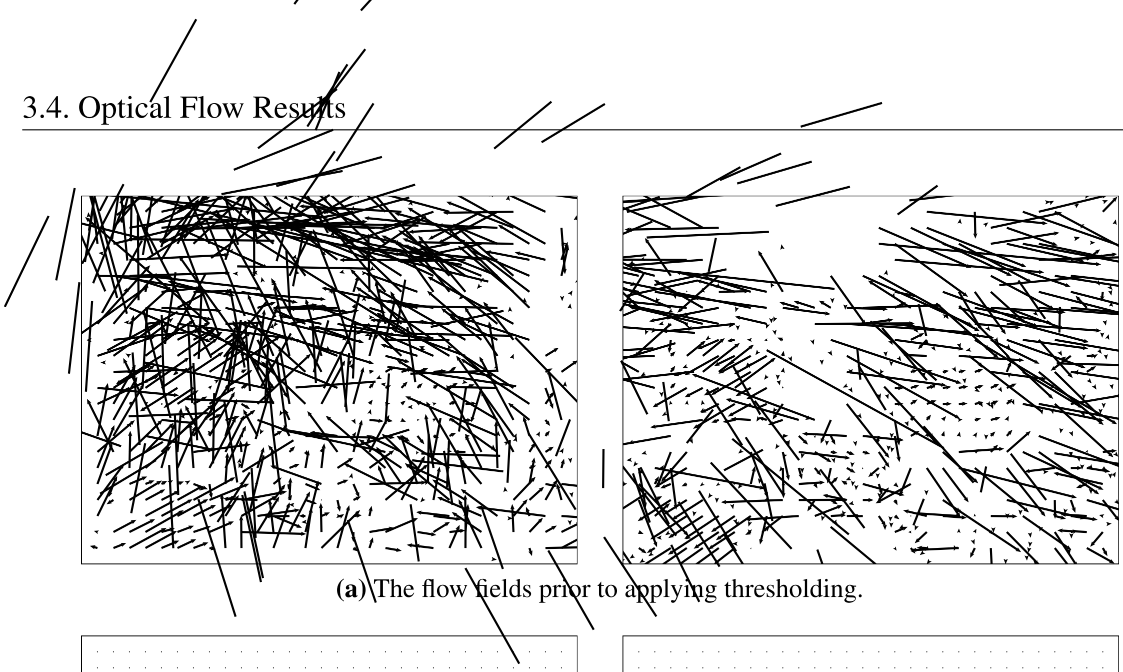
were within 30% of the true magnitude. After 15 frames the results are comporable to those of the other synthetic image sequences, with 90% of the vectors within 20% of the true magnitude and 8.5° of the true angle. Again, the performance of the algorithm is clearly improving with time.

Such an improvement should be expected for the Fleet and Langley algorithm, due to its recursive temporal filtering. The Lucas and Kanade algorithm operates on a specific range of images based on the temporal filtering and differentiation, and as the image sequence progresses frames outside of that range have no effect. The Fleet and Langley algorithm, on the other hand, estimates the temporal gradient on a moving window. Each image frame has an ever-decreasing effect on all subsequent optical flow calculations. Since the motion field in the Yosemite fly-through sequence is constant, each subsequent frame should reinforce the same optical flow values, gradually settling to a constant flow field. That the algorithm requires 15 images with constant flow to show these results, however, does not bode well for its performance under real-world conditions in which the motion field is not constant.

3.4.2 Performance with Real Image Sequences

As was noted above with the Yosemite fly-through sequence, flow field thresholding becomes important in areas with poor spatial contrast. The real image sequences were recorded after making this observation. The space in which the airship was to be flown and the real flow fields recorded contained primarily bare walls with little spatial contrast. To help alleviate this problem, the areas where the real image sequences were recorded were artificially cluttered to increase contrast. This technique to improve the performance of the optical flow algorithms is not without precedent. It is frequently used in the literature on optical flow for mobile robot navigation, for example [23–25, 56, 65].

Even with this increased contrast, thresholding remained an important step, as illustrated by the flow fields in Fig. 3.10. In it, a sample flow field from the airship-mounted image sequence (after the 20th frame) is shown before and after thresholding for both optical


(a) The flow fields prior to applying thresholding.

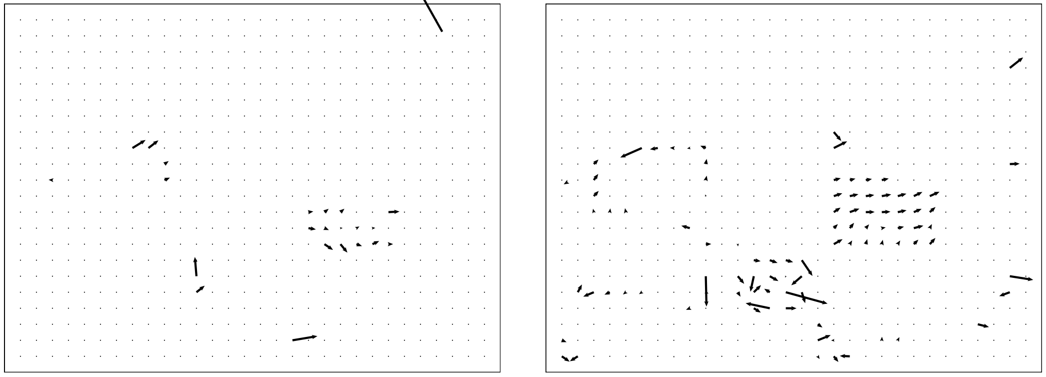
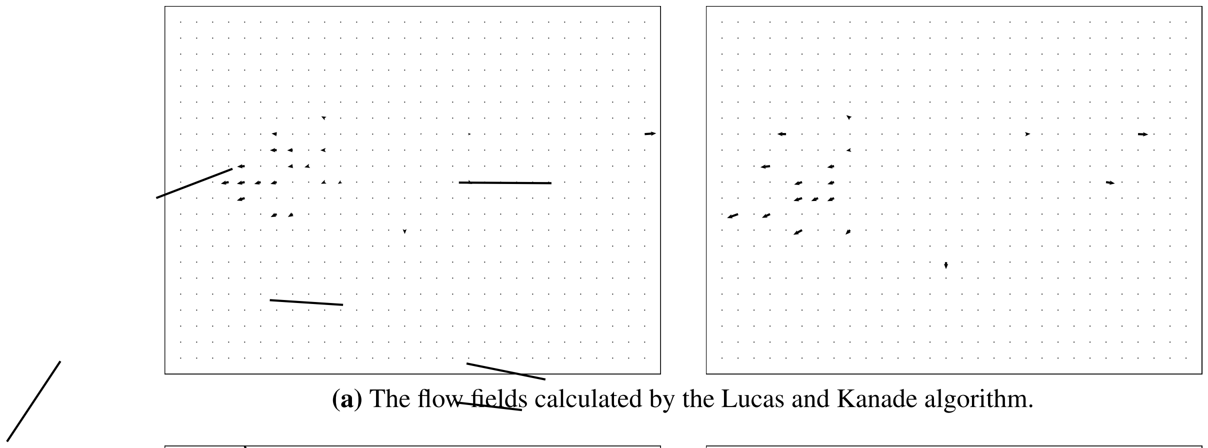

(b) The flow fields after removing elements with $\lambda_2 < 1$.

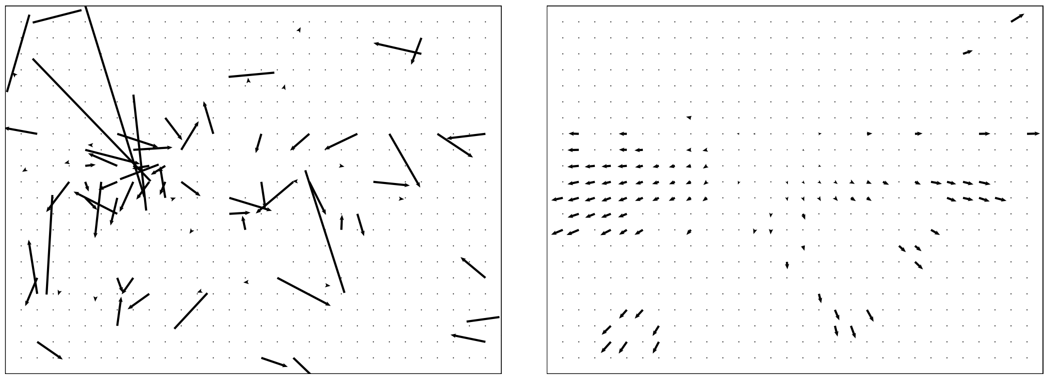
Figure 3.10 The importance of thresholding optical flow fields: The flow fields from a sample image in the airship-mounted image sequence. The flow fields on the left were calculated with the Lucas and Kanade algorithm; those on the right with the Fleet and Langley algorithm.

flow algorithms. It is immediately obvious that the flow fields prior to thresholding are completely useless because of the noise. Thresholding at least reduces the fields to values that at first glance appear manageable. In the remaining discussion on the performance of the algorithms, it will be assumed that thresholding of the flow fields has been performed, and un-thresholded flow fields will be ignored.

In the track-mounted image sequence, the camera underwent nearly constant motion normal to the image plane, very similar to the motion in the Yosemite fly-through sequence. As such it should be expected that the optical flow fields be similar. Indeed they are, as shown in Fig. 3.11. The flow fields calculated by the Lucas and Kanade algorithm were very sparse after applying thresholding, but the few remaining flow vectors appear to be correct, all roughly pointing away from a common point and scaled according to their distance from that point. Further, the results are consistent between the 5th and 50th images in the sequence. Likewise, the results of the Fleet and Langley algorithm, Fig. 3.11(b), also matched those from the Yosemite fly-through sequence. Early in the sequence the flow fields had not yet settled and showed erratic behaviour, but eventually the flow fields settled to a very good



(a) The flow fields calculated by the Lucas and Kanade algorithm.



(b) The flow fields calculated by the Fleet and Langley algorithm.

Figure 3.11 Optical flow of the track-mounted image sequence: The flow fields on the left are those after the 5th image in the sequence; those on the right after the 50th image.

representation of the expected optical flow.

More apparent in the results of the Fleet and Langley algorithm is that all of the flow vectors are diverging from a common point, called the instantaneous epipole. These characteristics are displayed more clearly in Fig. 3.12. So long as the camera motion is linear and at least partially normal to the image plane, there should exist such an epipole in the flow field [54]. Knowledge of the epipole is useful for higher-level vision algorithms, such as structure-from-motion and time-to-contact [59], the latter particularly useful for obstacle avoidance. However, these are not explored in this thesis because of the difficulties in determining the epipole location under general motion conditions of the airship. In this case, the purely translational motion and clean optical flow field mean that determining the epipole is relatively trivial – it is simply the intersection point of the lines on which the optical flow vectors sit – but under general motion the calculations are not as straight-forward [54].

The flow fields calculated for the airship-mounted sequence, Fig. 3.13, show the types of results that could be expected when attempting to use optical flow for airship navigation. Again, the flow fields of the Lucas and Kanade algorithm barely survived thresholding,

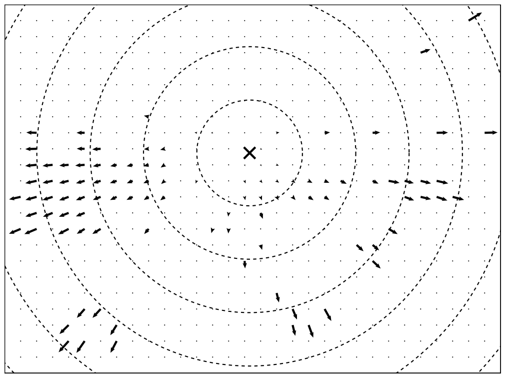
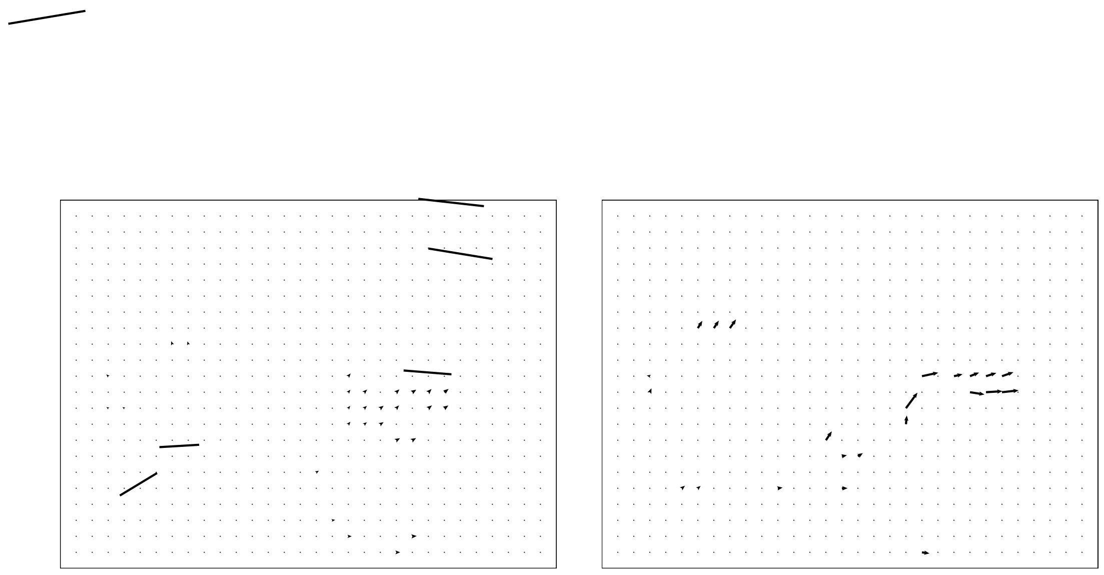


Figure 3.12 The track-mounted sequence epipole: The point marked by an $\times$, is the instantaneous epipole of the flow field. The concentric dashed lines represent lines of constant magnitude flow. Note that since the camera motion is not perfectly normal to the image plane – in which case the epipole would be in the center of the image – these are only approximate. The flow field is the same as that shown on the right in Fig. 3.13(b).

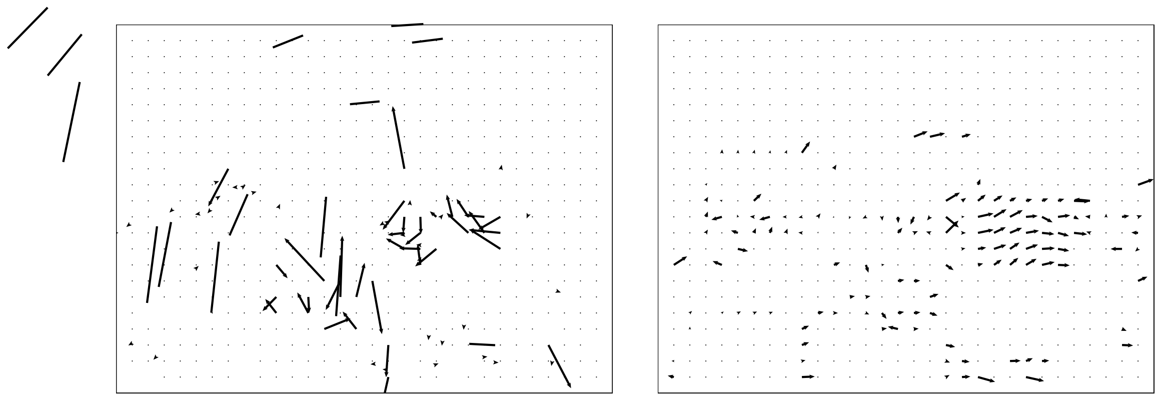
with only sparse vectors remaining. As was shown in Fig. 3.10, the thresholding is most certainly required, but even with the artificially cluttered environment the flow fields contain essentially no useful information.

The flow field calculated by the Fleet and Langley algorithm again follows the same pattern of being extremely erratic at first and eventually settling to a cleaner flow field. In this case, however, the cleaner flow field still contains many poor approximations of the motion field. Even after 50 frames in the sequence, there are regions of flow vectors that should have similar characteristics, but which have significant differences in their magnitudes or directions, some being nearly opposite one another. This poses a problem for even basic tasks such as determining the location of the epipole. Under general motion, the location of the epipole is unknown and could be anywhere. With clean and reliable flow fields, its location can be calculated with as few as six flow vectors [59]. If the flow vectors are unreliable, though, then the calculation of the epipole would fail to return its true position, so even basic calculations such as time-to-contact become impossible. Thus, despite the flow field after 50 frames appearing to be reasonably good at first glance, there is very little truly useful information contained in it.

The other major concern with the flow fields calculated by both methods is the paucity of the vectors that passed the thresholding test. Fewer than 20% of the vectors were kept in each of the flow fields shown in Fig. 3.13. Of those kept, the vast majority were in the areas of artificially increased spatial contrast. Attempting to operate based on optical flow in this space without artificially increasing the visual contrast of the surfaces would be impossible, simply due to the lack of texture.



(a) ~~The~~ flow fields calculated by the Lucas and Kanade algorithm.



(b) The flow fields calculated by the Fleet and Langley algorithm.

Figure 3.13 Optical flow of the airship-mounted image sequence: The flow fields on the left are those after the 5th image in the sequence; those on the right after the 50th image.

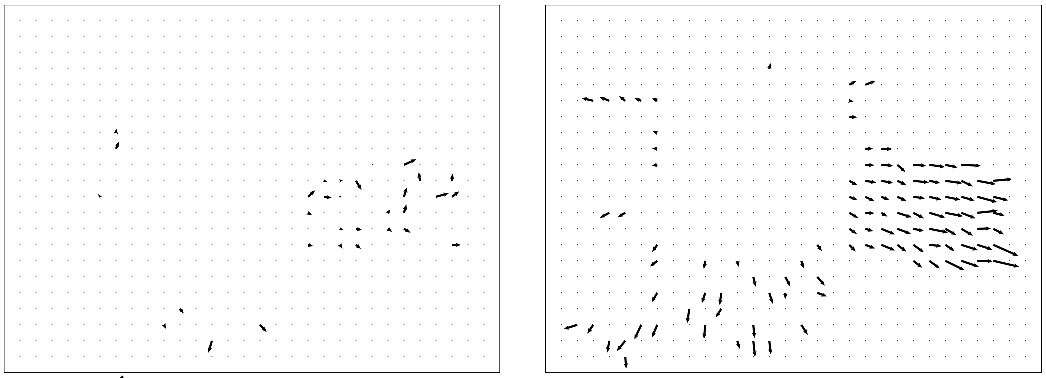
Robustness to Image Noise

When dealing with real image sequences recorded from an inexpensive wireless camera, noise is an inevitability. Thus, any algorithm making use of such a camera must be robust against noise. The spatial and temporal filtering techniques performed by the algorithms are meant to alleviate the impact of noise, but they cannot remove its impact entirely.

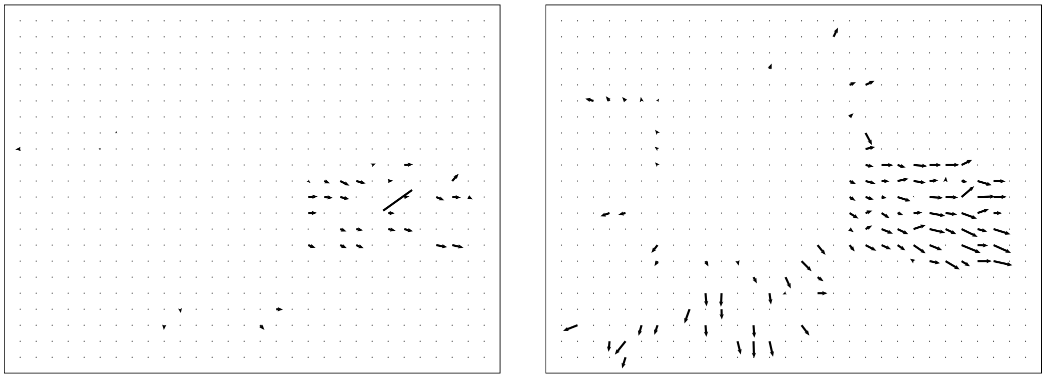
To evaluate the algorithms' robustness to noise, three consecutive optical flow fields from the airship-mounted image sequence that were calculated by each algorithm are presented in Fig. 3.14. The first flow fields are those for the frame immediately prior to the noisy frame shown in Fig. 3.6. Leading up to this frame were 60 essentially noise-free image frames. The second set of flow fields are those of the noisy image frame and the third set are for the following frame, again noise-free.

First, all of the flow fields exhibited similar behaviour to those presented earlier. Those calculated by the Lucas and Kanade algorithm are sparse and poorly represent the flow while those calculated by the Fleet and Langley algorithm have more surviving flow vectors and generally show better results. Focusing for a moment on the flow fields calculated by the Lucas and Kanade algorithm, they all display effects of the noisy image frame. The Gaussian temporal filtering and temporal derivative used by this algorithm cause each image frame to impact 9 flow fields. As a result, the three flow fields pictured are all relatively poor, though the noise-free flow fields depicted in Fig. 3.13(a) are quite poor to begin with. The image noise causes the directions of the flow vectors to be even more non-uniform than otherwise.

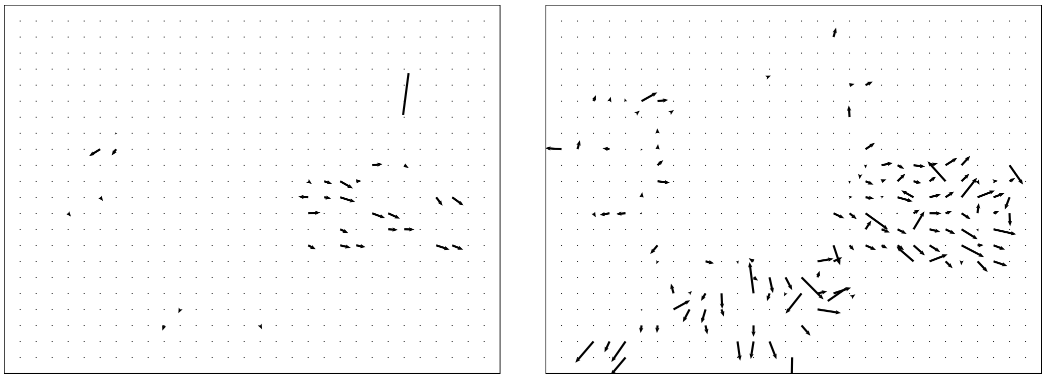
Turning to the Fleet and Langley flow fields, here the effects of the image noise are much more clearly visible. Unlike with the Lucas and Kanade algorithm, each frame has no effect on the flow fields calculated prior to it, only those after it. Thus, the first flow field is completely free of the effects of the noisy frame and displays results very similar to those in Fig. 3.13(b). Beginning with the second flow field the effects of the noise are immediately visible. Several of the flow vectors are nearly normal to the corresponding vectors in the first flow field and the magnitudes of many of the vectors are considerably different, though the temporal filter for this flow field still exerted a stabilizing effect. Once the effects of the previous noise-free frames are reduced even further after the noisy frame, the resulting flow field resembles those from early in the sequences more than those immediately preceding the noisy frame. Many of the vectors display erratic directions and several show highly anomalous magnitudes. The image noise has effectively set the algorithm back as if it were at the beginning of the image sequence and had not yet had time to settle into the true flow.



(a) The flow fields before the noisy frame.



(b) The flow fields immediately following the noisy frame.



(c) The flow fields after a noise free frame following the noisy frame.

Figure 3.14 Effects of image noise: The optical flow fields before, immediately after and further after a noisy image frame. The fields were calculated by the Lucase and Kanade algorithm; those on the right by the Fleet and Langley algorithm.

3.5 Conclusions on Optical Flow

Two optical flow algorithms were investigated with the intention of ultimately navigating the airship using their results. Both algorithms were evaluated against synthetic and real image sequences. The basic Lucas and Kanade algorithm performed reasonably well against the synthetic image sequences, but very poorly against the real image sequences. The resulting flow fields were very sparse and did not represent the true motion field. The problems were exasperated by image noise, which is common under the intended usage scenario.

The more advanced Fleet and Langley algorithm which uses recursive temporal filtering to improve performance gave excellent results for the synthetic image sequences so long as enough images of constant flow were provided. It fared poorly against the real image sequences if such constant motion was not observed. While the results were better than those of the Lucas and Kanade algorithm, under real-world conditions they were still extremely sparse and not uniform enough to provide confident measures of the true motion field under general camera motion. Again, image noise only compounded the problems, resulting in completely useless flow fields.

Based on the poor results displayed by both algorithms, further work on integrating them with the control and navigation of the airship was not even attempted. The constraints for small, light-weight, low-power, wireless cameras along with the requirement to be able to cope with fully general three-dimensional camera motion in visually bland environments were simply too much to ask of the algorithms.

Chapter 4

Probabilistic Localization With a Laser Range Scanner

While digital video cameras are capable of providing vast quantities of data, performing navigation based on that data proved a challenge due to the difficulty of extracting useful information from the video streams. In contrast, laser range scanners do not provide nearly as much raw data, but the range measurements they do provide are much more conducive to drawing useful information. Though early units were large and heavy, laser range scanners have successfully been used for localization and navigation of ground vehicles for many years. Several modern LIDAR units, the Hokuyo URG-04LX among them, fit within the airship's payload size and weight restrictions. Using one for localization of the airship was chosen as the next direction of investigation.

4.1 Review of LIDAR Based Localization and Navigation

Initially, LIDAR data was used with feature- and scan-matching techniques for localization of ground-based robots operating in planar environments. Einsele and Färber [15] developed a simultaneous localization and mapping algorithm that tracked distinctive features such as corners and planes measured by the robot to generate a map. Further localization was achieved by performing a least-squares scan-matching comparison of new LIDAR scan to the previous reference scans. A similar geometric scan-matching algorithm was used in [21] for a soccer playing robot. The laser scans were matched against the distinctive border of the playing field, however the environment was not static and obstructions caused by other robots on the field had to be accounted for.

Soon, probabilistic localization algorithms became popular in the field. Particle filters allowed for global localization with so-called Monte Carlo localization algorithms in particular being commonly used. These will be discussed in detail in the following section. Other, similar particle filters were also used, such as in [44]. Here, two planar, ground-based robots worked cooperatively to improve their performance. One robot was equipped with a LIDAR and the other with a distinctive shape attached to it that was easy to identify from the LIDAR data. The two robots were jointly localized using a particle filter that improved on odometry data from each robot by adjusting for their relative positions measured by the LIDAR.

The other major algorithm of probabilistic methods, Kalman filtering, was also commonly used for state tracking. Borges and Aldon [5] combined an extended Kalman filter (EFK) with feature-matching algorithms for robot localization. The planar robot began with a basic map of its environment that it then updated as new information from the LIDAR became available. The problem was simplified somewhat since the extended Kalman filter was only required to track the robot's position; a gyrometer combined with dead-reckoning estimates were able to accurately track the orientation of the planar robot. Several variations of the EKF were investigated in two experimental environments – one simple and one complex. In the simple environment, all of the algorithms successfully tracked the robot's position, however in the complex environment only algorithms that incorporated a weighted least-squares feature correspondence between the LIDAR scans and the map were successful.

An EKF was again used in [53] for planar state tracking, this time including orientation. An un-motorized cart was manually pushed down a corridor while recording data from on-board LIDAR and IMU sensors – no direct odometry data was taken. Using these data the location and orientation of the robot were tracked through the corridor. The ultimate goal was to implement such a system on an outdoor vehicle for the 2005 DARPA Grand Challenge, where the corridor would be replaced by a roadway.

Recent technological advances have enabled smaller and lighter LIDAR units to be produced. Though not as performant as the larger and heavier units, typically exhibiting reduced detection range and scanning frequency, they have become small enough to be mounted to indoor aerial vehicles. He, Prentice and Roy [22] added a LIDAR to a commercially available quadrotor helicopter. The vehicle came equipped with an IMU and a low-level stability controller, so the primary concern of the research was localization and path planning such that the selected paths would avoid areas where the robot would likely get lost. An unscented Kalman filter used the IMU and LIDAR data for 4-DOF state tracking – 3-dimensional position and yaw rotation. Though the LIDAR was mounted such that it produced a horizontal scan, a mirror was cleverly used to direct some of the LIDAR beams downward to measure altitude.

The above work was expanded on in [2], where a similar quadrotor with an attached LIDAR was used. In this case the environment was mapped simultaneously as the robot was flying. The on-board computer tracked the robot's position using an EKF while an off-board computer performed the map-building computations using the LIDAR data. Updated maps and position corrections were then periodically sent to the robot. This robot won the 19th AUVSI International Aerial Robotics Competition, which featured navigation through a confined, indoor space as the primary challenge.

4.2 Monte Carlo Localization

Monte Carlo Localization (MCL), first introduced by Dellaert et al. [12], is a form of probabilistic localization. Probabilistic robot localization relies on statistical techniques for representing a robot's pose and motion. It makes use of, and in fact relies on, uncertainty in all aspects of robot operation – sensor measurements, motion models, environments, etc. – to determine and track the robot's pose in space as a probabilistic expression.

4.2.1 Bayes Filtering for Recursive Estimation

MCL is an application of recursive Bayes filtering. Bayes filtering is a general algorithm that updates probability distributions from received data. It stems from an equation known as Bayes rule [50]

$$p(x \mid d) = \frac{p(d \mid x)\, p(x)}{p(d)} \tag{4.1}$$

where x is the value for which the probability is desired, the state variable; d is the available data; $p(x \mid d)$ is the probability of x conditioned on the knowledge of d and is known as the *posterior*; and $p(x)$ is the probability prior to incorporating the information d, so is known as the *prior*. Note also that $p(d) > 0$ is required for this equation to hold, however it is independent of x so is often excluded in favour of a normalization variable η such that

$$p(x \mid d) = \eta p(d \mid x)\, p(x) \tag{4.2}$$

Notice the "inverse" conditional probability, $p(d \mid x)$, on the right-hand sides of the above equations. It allows inference of the value of x based on the probability of data d assuming that x was indeed the case. Further, it encompasses the fact that any data d is caused by some state x, as opposed to the state being conditional on the data.

The Markov Assumption

Bayesian filtering assumes that the state at the previous time, x_{t-1} contains sufficient information from all previous data up until time $t-1$ and that earlier data, $d_{0:t-1}$, are not explicitly required to fully define the current state. Further, it is assumed that the state x_t contains all of the information on the current state of the environment, as opposed to just a robot's pose, for example. Together, these are known as the Markov assumption, or the complete state assumption. Many factors violate this assumption, such as inaccurate or unmodeled environment dynamics (e.g., moving people), approximation errors (e.g., approximating a probability by a Gaussian distribution), etc.. Despite these violations, Bayes filters are relatively robust in practice, so long as care is taken that the state variable is defined such that the effects of unmodeled variables are near random [50].

Bayes Filters for Mobile Robot Localization

In mobile robot localization, two distinct types of data are used in place of a general d: perceptual data from the sensors at time t are contained in an *observation* variable o_t, while the robot's odometry data during the time interval $(t-1:t]$ are stored in an *action* variable a_t. Further, posterior probabilities are typically called *beliefs* and are denoted

$$bel(x_t) = p(x_t \,|\, o_{0:t}, a_{0:t}) \tag{4.3}$$

while beliefs prior to incorporating measurement o_t are denoted

$$\overline{bel}(x_t) = p(x_t \,|\, o_{0:t-1}, a_{0:t}) \,. \tag{4.4}$$

Taking the above, Bayes rule of eq. (4.1) becomes (ignoring the Markov assumption for a moment)

$$bel(x_t) = \frac{p(o_t \,|\, x_t, a_{0:t}, o_{0:t-1})\, p(x_t \,|\, a_{0:t}, o_{0:t-1})}{p(o_t \,|\, a_{0:t}, o_{0:t-1})} \tag{4.5}$$

where again the denominator is independent of x and can be replaced by the normalization value η. Applying the Markov assumption further simplifies eq. (4.5) to

$$bel(x_t) = \eta p(o_t \,|\, x_t)\, \overline{bel}(x_t) \,. \tag{4.6}$$

Integrating over the state at time $t-1$ provides [50]

$$\overline{bel}(x_t) = \int p(x_t \,|\, x_{t-1}, a_t)\, p(x_{t-1} \,|\, a_{0:t-1}, o_{0:t-1}) \; dx_{t-1} \tag{4.7}$$

and incorporating eq. (4.7) into eq. (4.6) gives

$$bel(x_t) = \eta p(o_t \,|\, x_t) \int p(x_t \,|\, x_{t-1}, a_t)\, p(x_{t-1} \,|\, a_{0:t-1}, o_{0:t-1}) \; dx_{t-1} \tag{4.8}$$

which is the key recursive update equation of all Bayes filters. It allows for recursive estimation of a robot's state based on knowledge of previous states, control action data, and observed sensor data.

Bayes filtering has been used to derive several robot localization and tracking algorithms. Kalman filtering, information filtering, and their derivatives approximate the posterior as a unimodal Gaussian probability density function (PDF). This approximation allows for efficient computation and can be quite accurate so long as the system is not too far from linear, but the approximation also restricts the algorithms to only performing local pose tracking, as opposed to global localization.

The other major branch of Bayes filters are particle filters, MCL being one example. Rather than applying a preselected mathematical function, particle filters approximate the state probability distribution as a weighted sum of many individual "particles" that are each an estimate of what the true state of the robot may be. This property allows particle filters to inherently be capable of global localization, while Kalman and information filters are generally restricted to state tracking. On the other hand, particle filters require some method of evaluating the accuracy of each particle. It was primarily for the prospect of global localization that a particle filter approach was chosen, and MCL in particular because of the promising results others have shown in the literature.

4.2.2 The Monte Carlo Localization Algorithm

Being a particle filter, MCL works with a set χ of M estimates of the state vector, x. During each time-step of robot operation, χ is adjusted through a three-step process. First, each state estimate of the set χ_{t-1} is adjusted according to a_t using a motion model that attempts to apply the same relative motion as the robot underwent. Next, each particle is evaluated against o_t using a measurement model that provides a weight, or importance factor, for each particle. After all of the weights are calculated the set of particles plus weights is denoted $\overline{\chi}_t$ and the next set of particles, χ_t is drawn from $\overline{\chi}_t$ with probability proportional to the weights. This resampling is done with replacement, so there are likely to be multiple copies of the strongest particles in the resulting set. Of at least equal importance are the particles that are not drawn; those are the particles that were likely incorrect estimates of x. Thus, over time the set of particles should converge on the true state of the robot [50]. The MCL algorithm is depicted in flowchart form in Fig. 4.1.

A simulated example of MCL is shown in Fig. 4.2. In it, a planar robot with 3-DOF of motion translates with constant speed along a straight line through a corridor-like environment. The robot is equipped with a range scanner and odometer, both simulated with

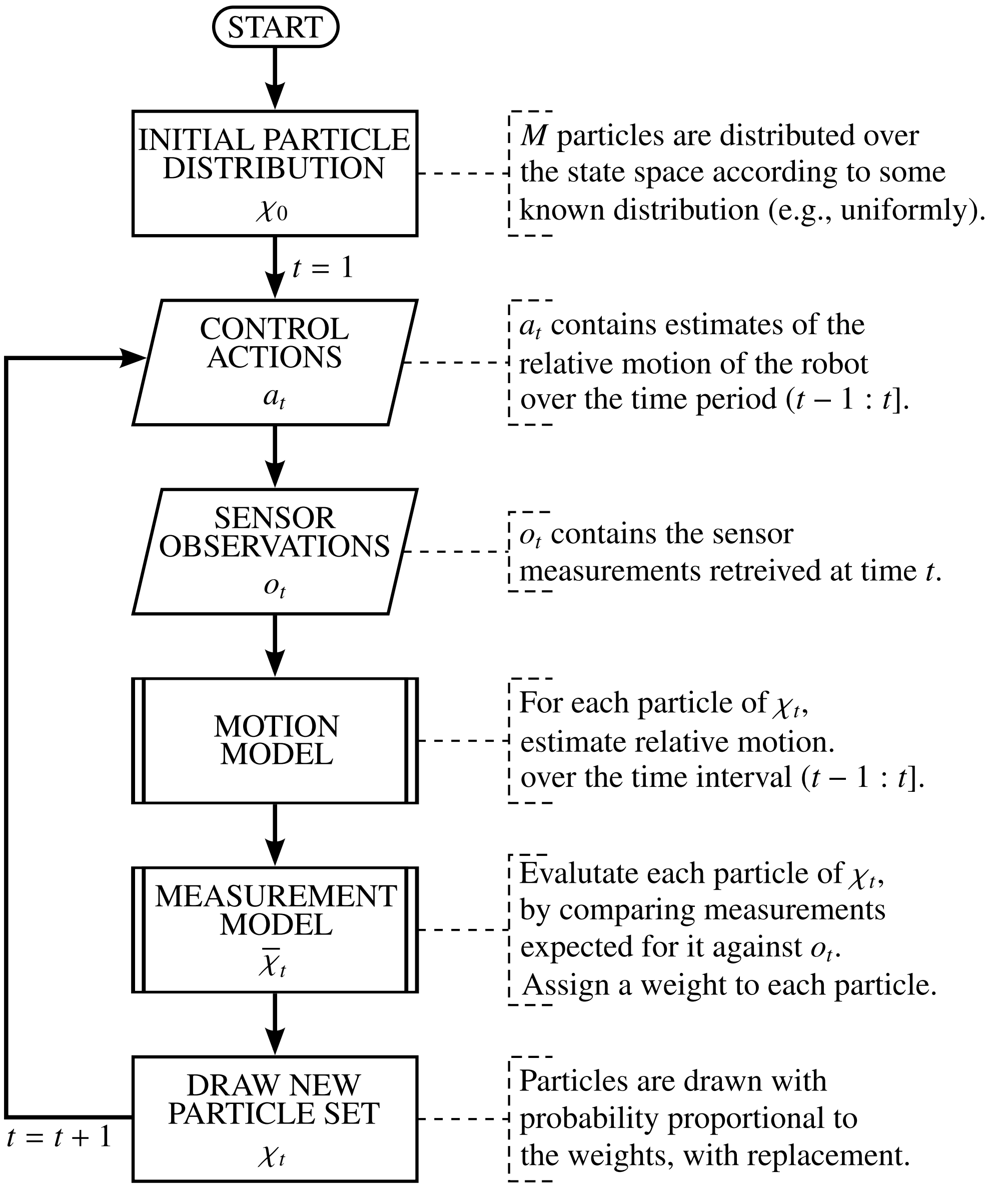
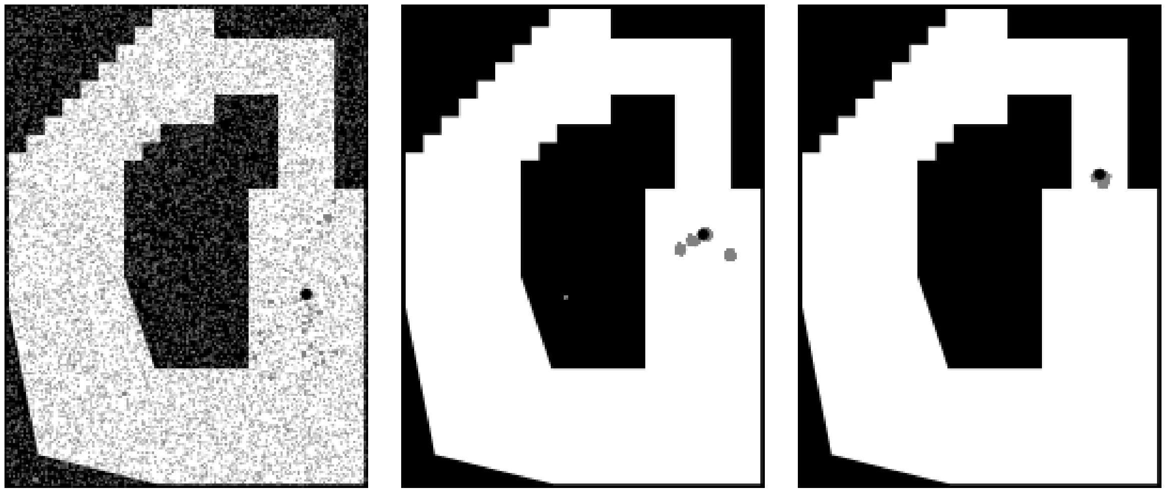

START
INITIAL PARTICLE DISTRIBUTION
$\mathcal{X}_0$
M particles are distributed over the state space according to some known distribution (e.g., uniformly).
$t = 1$
CONTROL ACTIONS
a_t
a_t contains estimates of the relative motion of the robot over the time period $(t - 1 : t]$.
SENSOR OBSERVATIONS
o_t
o_t contains the sensor measurements retreived at time t.
MOTION MODEL
For each particle of χ_t, estimate relative motion. over the time interval $(t - 1 : t]$.
MEASUREMENT MODEL
$\bar{\mathcal{X}}_t$
Evalutate each particle of χ_t, by comparing measurements expected for it against o_t. Assign a weight to each particle.
DRAW NEW PARTICLE SET
$\mathcal{X}_t$
Particles are drawn with probability proportional to the weights, with replacement.
$t = t + 1$

Figure 4.1 MCL algorithm flowchart. The specifics of the motion and measurement model functions vary according to the robot and sensors.



(a) Initial distribution **(b)** After 100 timesteps **(c)** After 200 timestesps

Figure 4.2 An example of planar Monte Carlo Localization: A robot, position indicated by the large black dot, undergoing pure translation at a constant speed along a straight line in a simulated planar environment performs MCL. Particle positions are shown as small gray dots.

sensor noise. To begin, 10^5 particles are uniformly distributed over the 3-dimensional state space (orientations are not represented in Fig. 4.2). After 100 time steps the particles have clustered into a few distinct groups near the true robot state and after 200 time steps the robot has been successfully localized with all of the particles forming a tight cluster near the true state.

The first two steps of the MCL algorithm described above relied on models of the system. The exact nature of these models depends on the type of robot being operated (e.g., ground-based or aerial), the types of sensors on-board the robot (e.g., cameras or range scanners), and the type of environment in which the robot is operating (e.g., static or dynamic).

The Motion Model

The motion model adjusts the state estimate of each particle during each time step. Assuming that the environment is static and known – that is, the robot is operating with a perfect map of the environment – the motion model only estimates the motion of the robot and applies the same relative motion to each particle. The method of determining the relative motion of the robot depends on the type of robot and the available sensors.

Most ground-based robots use wheel encoders for odometry measurements, for example [12, 36, 43, 51, 61, 63]. Though simple, odometry from wheel encoders suffers from drift due to wheel slippage or slight unevenness of the ground. These sorts of errors should be estimated by the motion model. Non-wheel-based ground robots rely on other means of motion estimation, typically gait dynamics [46, 55, 60].

Aerial robots do not have the luxury of direct contact with their environment, and hence can not rely on anything like wheel encoders. Instead, aerial robots typically use one of two approaches: flight dynamics or dead-reckoning from inertial measurement unit data. In [19], a quadrotor aircraft with 4-DOF motion used an IMU for incremental motion estimation. The altitude of the aircraft was directly measured using a LIDAR and the IMU itself provided direct orientation measurements, so only two dimensions of motion estimation were required.

Recently, an interesting approach to motion estimation from flight dynamics was presented by Müller, Gonsior and Burgard [40]. They applied MCL to a small, indoor, underactuated airship that had only four wide-angle sonic range finders as sensors. Lacking an IMU, the motion model relied on the airship's flight dynamics based on the known control inputs. Accurately modeling airship dynamics is a complex task to begin with [33], but the challenge was even greater in this case because the dynamics were subject to change during operation due to things like changing buoyancy or draining batteries reducing the available thrust. To overcome this, the parameters of the motion model were adjusted in real-time based on the sensor data.

The Measurement Model

The measurement model evaluates the importance factor of each particle during each time step. To do this, it compares what measurement data at the particle's state would be to the actual measurement data obtained by the robot. If the data closely match, then there is a high probability that the particle's state is near the true state, and a high importance factor is assigned accordingly. The challenge, of course, is determining what data would be returned by the sensors at the particle's state.

The nature of the measurement model is entirely sensor-specific. The two most commonly encountered types of sensors used with MCL are image cameras and range sensors. Measurement models for camera-based data either simulate the characteristic traits of known landmarks in the environment and compare them to the same traits extracted from the images [46, 55], or draw images from a pre-recorded data set of the environment and compare them directly to the images from the camera [36, 51].

Range sensors, the focus here being on laser range scanners though sonic range finders are similar, take a very different approach to the measurement model. The typical approach is to simulate a range scan within a known map for each particle using a beam model [19, 43, 51]. The beams could be calculated on-line, but beam calculations are computationally expensive and calculating several beams for each of several thousand particles, multiple times per second, in even a moderately complex map quickly hits the limits of modern CPUs. To improve efficiency, beams could instead be pre-calculated off-line for a subset of points in the state-space. During operation, the pre-calculated beams for the point closest to the particle are taken as those for the particle. Finally, the simulated beams for each particle are compared to the real measurements through aggregate values such as the mean distance of all of the beams, or the position of the beam centroid relative to the sensor [50, 51].

In [63] this idea is taken one step further. A grid of precalculated measurements is created, but another grid of the measurement "energy" is also created. The "energy" was a simple combination of all of the sensor measurements for that grid point. Since the robot was assumed round with uniform sensor distributions around its perimeter, the energy of a point was constant regardless of orientation and the second grid's dimensionality was lower than that of a standard beam grid, allowing easy segregation of particles based on this "energy" value. Of course, this is just another aggregate value of the beam distances. To get the added utility described, the sensors must be uniformly arranged around the entire perimeter, so its general applicability is minimal.

Either of the above measurement model approaches – known images/landmarks or a known map – are a regression from the ultimate goals for the airship laid out in the introductory chapter; specifically, the ability to operate in a completely unknown environment without previous preparation of the environment. Using a known map in this instance was deemed an acceptable compromise, while preparing the operating space with known landmarks for the work of Chapter 3 was not, with the following reasoning. The airship is restricted to indoor operation, thus would be operating in a human-constructed and generally known environment. A preliminary map could be quickly generated using building plans, for example. Populating the space with known landmarks, however, could not be done without much greater operator intervention.

4.2.3 Advanced Monte Carlo Localization Algorithms

The basic MCL algorithm as described above has several shortcomings. Primarily one known as the particle deprivation problem, in which incorrect localization occurs due to insufficient particles near the true robot state. It would theoretically affect any particle filter using a finite number of particles and allowed to run an arbitrarily long time, simply due to the variance in random sampling [50]. In practice, particle deprivation primarily occurs if the number of particles is too small. To test against particle deprivation the "kidnapped robot" challenge has been developed, where the robot is moved instantaneously from one location to which it has correctly localized to another, unknown location. The challenge is to re-localize to the new location. Basic MCL would fail.

A simple method of combating the particle deprivation problem is to introduce randomly selected particles into the set χ_t. A naïve approach would be to add a fixed number of random particles in each time step, but how many should be added? Further, if the robot was operating well, arbitrarily adding random particles would reduce the performance. A slightly better approach would be to add random particles based on a heuristic, such as the mean particle weight. There are, however, some scenarios where even if the weights are low, new particles should not be added: if a single measurement is unusually noisy, or if the particles are still spread out during global localization.

An even better approach was suggested by Gutmann and Fox [20]. Known as the Augmented MCL algorithm, it tracks two running averages of the particle weights, short-term (w_{fast}) and long-term (w_{slow}). These two averages determine if a new sample should be added. Since the particles should be converging on the true position, the mean of the particle weights should constantly be increasing, so during typical operation $w_{\text{fast}} > w_{\text{slow}}$. If that relationship does not hold, the particle set has become a worse approximation of the robot state. In this case new, random particles should be added to the set to attempt to re-localize the robot. Moreover, by keeping the long-term average as an exponential smoothing term, momentary sensor noise spikes that temporarily reduce the mean particle weight are not mistaken as poor localization.

Related to the particle deprivation problem is the question of how many particles to use. If too few are used the particle deprivation problem is encountered. Conversely, if correct localization has been achieved and the algorithm is simply performing state tracking, far fewer particles are required since they are all focused near the correct state. Computing too many particles would waste resources that could be put toward other tasks. Optimally, the number of particles would vary based on the requirements at the time, which is precisely what the KLD-Sampling (short for *Kullback-Leibler distance sampling*) variant of MCL attempts to do [17]. In essence, KLD-Sampling monitors the distribution of particles. If they are being widely distributed, then many particles are required to sufficiently cover the space of possible robot states, but if they are highly concentrated then fewer particles are required.

One of the advantages of MCL over other probabilistic localization algorithms is the ability to track multiple hypotheses of the robot's state in the form of multiple clusters of particles through the state space. The basic MCL algorithm makes no explicit point of keeping separate hypotheses and as a result the particles tend to converge on a single state before any data to support that state over others has been obtained. Milstein, Sánchez and Williamson [37] proposed taking advantage of the MCL's clustering abilities by explicitly keeping multiple independent clusters of data. Each cluster was used to maintain a separate state hypotheses and the clusters were evaluated against one-another to determine the best overall state hypothesis. Periodically, weak clusters were removed from the set and new clusters generated that performed global localization, which helped overcome the kidnapped robot problem. The obvious problem with Cluster-MCL, as it was dubbed, is the number of particles required. Since each cluster of particles is expected to perform global localization, each set requires enough particles to adequately cover the state space. Using a single set of particles and dynamically drawing clusters from that set could potentially improve efficiency.

Finally, and most bizarrely, if the robot's sensors are too accurate or, in the extreme, able to perfectly localize the robot, basic MCL would fail. Within the measurement model that simulates sensor measurements is a sensor noise model. Nominally, the sensor model would perfectly capture the distribution of sensor measurements. This distribution provides each particle an area of influence (AOI) within the state space. If the sensor is noise-free, then each particle's area of influence reduces to a single point, so the probability of a particle's AOI containing the true robot state is exactly zero and the particle set would never converge to the correct state [43, 51]. One obvious method to avoid this problem is to artificially increase the sensor noise in the measurement model such that the cumulative AOI of all of the particles has a high probability of containing the true robot state. An intelligent approach to this was presented in [43], where the noise model was adjusted such that the AOI of each particle was expanded until it encountered its nearest neighbouring particle.

Another, more rigorous solution to this problem was proposed by Thrun et al. [51] with the Mixture MCL algorithm. The basic MCL algorithm predicts states based on the belief during the prior time step and assigns weights based on sensor measurements. The authors proposed a "dual" of the MCL algorithm that does the opposite – predicts states based on sensor measurements and assigns weights according to the prior belief. The dual of MCL would localize perfectly with a perfect measurement sensor, but would conversely fail with a perfect motion model. Thus, to increase robustness, Mixture MCL generates predictions with probability ϕ using standard MCL, and with probability $1 - \phi$ using the dual of MCL, where ϕ is pre-determined. While conceptually simple, implementing the dual of MCL is not necessarily straight-forward. For example, predicting the robot's state directly from range scanner measurements in a complex environment is far from trivial. There is one type of sensor for which predicting states directly from measurements may be feasible: cameras tracking known landmarks in the environment, which is precisely what was done in [51].

4.3 Implementation of Monte Carlo Localization

Due to difficulties stabilizing the airship with the LIDAR mounted, it was decided that airship's motion would be restricted to 4-DOF – three translational and rotation about its vertical axis. The motion was restricted by arranging the components on the airship such that the center of mass was directly below the center of buoyancy and instructing the controller to actively maintain zero roll and pitch rotations. Then, based on the above descriptions of the various MCL algorithms, Augmented MCL was selected for implementation. It solves the particle deprivation problem and is straight-forward to implement. Mixture MCL is not well suited for use with laser range scanners, so was ruled out of possible implementation. KLD-Sampling has very attractive properties from a computational perspective, but it was not selected for implementation due to limitations in Simulink. Though it may be possible to implement a KLD-sampling-like algorithm, Simulink requires that arrays and loop sizes be specified prior to running the code, so dealing with an indeterminate number of particles would not be a straight-forward exercise. The details of the implementation, including the sample flight data, the motion and measurement models for Augmented MCL, and the selection of the airship's state from the particle set are presented in the following sections.

4.3.1 Flight Data

Evaluation of the Augmented MCL algorithm was performed off-line using data recorded from airship flight. The only data available for testing was a single, 75 s test flight within the AML with the LIDAR mounted to the airship. Further flights could not be performed due to building renovations forcing the shutdown of the laboratory. The test flight was restricted to 4-DOF motion, though roll and pitch motions could not be completely eliminated they were minimized to a few degrees and will be assumed as zero.

The flight was tracked using the VICON system, whose data was be treated as ground-truth. The LIDAR and the IMU were also both mounted to the airship and their data streams recorded – LIDAR scans at a period of 0.12 s and the IMU data at 0.01 s. The commands sent to the airship's thrusters were also recorded at 0.01 s intervals.

4.3.2 Dynamics Based Motion Model

The greatest challenge for implementing a probabilistic localization algorithm for the airship was developing a motion model. Obviously, tools like wheel encoders that directly measure odometry are impossible to use. Other aerial vehicles commonly use IMUs to measure accelerations and estimate motion based on those, for example [1, 2, 19]. As was shown in §2.3.1, the weak thrusters and large inertia of the airship restrict its accelerations to very small values, while the vibrations induced by the thrusters completely drown out the signal to be measured by the IMU. Thus, this approach was deemed infeasible.

Since the most commonly tools used for measuring robot motion could not be used, a basic dynamics model of the airship was used instead. The dynamics of traditionally shaped airships have been studied in the past, for example [33], but they cannot be directly applied to the spherical airship and due to the unique nature of the spherical airship, dynamics models of such systems do not exist in the literature. Investigating and developing a detailed dynamics model of the airship was beyond the scope of the present research. Further, probabilistic localization algorithms ostensibly require only approximate motion models and the IMU was capable of providing accurate orientation information, so it was decided to use a basic, constant acceleration model.

The airship's controller was specified to run at a frequency of 100 Hz, since that was the rate at which IMU data was obtainable. At each controller time step, desired motor forces were calculated by the controller. These forces were based on thruster characterizations carried out in the laboratory. The net force of the thrusters in the fixed global frame was calculated by using the airship orientation data provided by the IMU:

$$\mathbf{F} = R_t^A \sum_{p=1}^{6} \mathbf{f}^p \tag{4.9}$$

where $\mathbf{f}^p$ is the vectored thrust generated by thruster p relative to the airship's coordinate frame, and R^A is the 3×3 rotation matrix expressing the orientation of the airship's coordinate frame relative to the global coordinate frame, as measured by the IMU. Note that the orientation was measured in 3-dimensions, despite the roll and pitch rotations of the airship being constricted. Also, uncertainty was added to the thruster forces in the form of a zero-mean Gaussian distribution with standard deviation of 0.05 N, so the net thruster force for the motion model was calculated independently for every particle.

The force generated by the thrusters was assumed to be the net force acting on the airship – the neutral buoyancy of the airship counteracted gravity, and accurate drag estimates were not attainable. Thus, the acceleration of the airship during each time step was simply calculated as

$$\mathbf{a} = \frac{\mathbf{F}}{m_A} \tag{4.10}$$

where m_A is the mass of the airship. This acceleration was assumed constant for the duration of the time step, so the velocity and position of the airship were updated using the equations of motion for constant acceleration:

$$\mathbf{v}(t) = \mathbf{v}(t-1) + \mathbf{a}(t)\,\Delta t \tag{4.11}$$

$$\mathbf{p}(t) = \mathbf{p}(t-1) + \mathbf{v}(t-1)\,\Delta t + \frac{\mathbf{a}_t\,(\Delta t)^2}{2} \tag{4.12}$$

where Δt was the time step length, 0.01 s. The relative change in position was then applied to each particle. Finally, the orientation of the particles was taken as the final yaw rotation measured by the IMU plus zero-mean Gaussian uncertainty with a standard deviation of 0.008 rad.

4.3.3 LIDAR Measurement Model

A beam model was used to calculate the simulated LIDAR distances for each particle within the environment. Each measurement was calculated as the intersection point of the line along which the beam travels and the nearest obstacle face within a predefined map. The uncertainty model described in §2.3.3 was applied to the simulated scans. Since the uncertainty model assumes independence of each LIDAR beam, not all of the beams were used. The uncertainty of adjacent beams is not independent, for example an unmodeled obstacle in the environment would affect several LIDAR beams similarly. To ensure beam independence, only 11 uniformly spaced beams were used.[1]

Map Definition

Constructing environment maps using mobile robots is an active field of research, but beyond the scope of this work. For the MCL implementation, a known, detailed map of the operating environment was assumed. As mentioned previously, the basis of such a map could be taken from building plans. In this case, the operating environment was relatively confined and easily measurable, thus the map was created manually.

The map was defined in a simple, custom format that explicitly defined points and faces within the environment. Points were defined by their location relative to an origin while faces were defined as triangular areas with vertices at the above-defined points. Further, faces were given directions – a LIDAR beam could only intersect with one side of the face. If, during beam calculations, the beam struck the back side of a the nearest face, then the associated particle state was known to be invalid. For example, the LIDAR could not be located in the space within a wall, so if a beam struck the back side of a face on a wall, then the particle was located within the wall and was invalid.

The map of the flight space contained a volume slightly larger than that visible by the motion tracking system – 7.0 m $\times$ 5.0 m $\times$ 3.6 m. A total of 107 points and 108 faces made up the map, which included all permanent fixtures of the volume such as walls as well as semi-permanent objects, such as large pieces of furniture. Not included in the map were easily movable objects such as chairs or cardboard boxes that happened to be

[1]Beam counts from 8 to 60 have been suggested in the literature [43, 50, 51]. After selecting 15 beams for use here, it was discovered that two at each end of the arc were obstructed by the airship's hull, so only the 11 middle beams were ultimately used.

located in the operating space. These objects were omitted since their positions could easily change between flights, which would require re-measuring the locations of the objects and re-defining the map if they were included. Such obstacles were to be accounted for by the LIDAR uncertainty model discussed earlier in §2.3.3.

Beam Computation

Two approaches to beam computation were investigated: on-line scans and pre-computed scans. Ideally, each distance for each particle would be calculated on-line for the exact particle position for each time step. In complex 3-dimensional environments with thousands of particles, however, this very quickly becomes computationally impossible to do in real-time on a standard CPU. Alternatively, distances for a subset of points in the state space could be pre-calculated off-line. During operation the distances for each particle would be approximated as those of the nearest point for which distances were calculated. This approach has been used in the literature, for example in [12, 50, 51].

Though pre-computing values is less computationally expensive, it has several drawbacks. The most obvious of course being that the simulated LIDAR measurements are no longer perfectly accurate. If the environment does not contain many small components, then these inaccuracies would be minimal. The other major disadvantage of pre-computing the LIDAR scans is that the scans must be recalculated for any change in the environment or reconfiguration of the LIDAR mounting on the airship.

Pre-computing a grid of scans is not free of the restrictions of computing resources, either. Taking the operating environment as the map described in the previous section, descretizing the space with a 0.1 m spatial resolution and 5° angular resolution with 11 LIDAR beams calculated for every point, the 4-DOF state space of the airship would require 9.4×10^7 distance values. Storing each value as a 16-bit integer would require 179 MB of computer memory – not unreasonable by modern standards and this discretization was implemented for testing. However, increasing the state space to 6-DOF within the same environment would require 2.4×10^{11} distance values, or 463 GB of memory – not even remotely feasible by current computing standards if attempting to store the full set in memory.

Particle Evaluation

Ultimately, the measurement model must provide an evaluation of each particle such that good particles may be identified and poor particles dropped. Two approaches were briefly mentioned in §4.2.2 – comparing the average and the centroid of each particle's measurements against the actual measurements – and these two were also selected for implementation.

Mathematically, the difference in means is very straight forward:

$$d_p^m = \left| \frac{\sum_{k=1}^{K} r_t^{k,a} - r_t^{k,p}}{K} \right| \tag{4.13}$$

where K is the number of distances returned by the LIDAR per scan, r_a^k is the actual distance measured by the LIDAR along beam k, and r_p^k is the simulated distance along beam k of particle p.

The centroids are calculated as

$$\mathbf{c} = \left(c_x, c_y\right) = \left(\frac{\sum_{k=1}^{K} r^k \cos\theta_k}{K}, \frac{\sum_{k=1}^{K} r^k \sin\theta_k}{K} \right) \tag{4.14}$$

where θ_k is the angle the k beam makes with the x-axis of the LIDAR's local coordinate frame. The distance between the centroids is simply

$$d_p^c = \sqrt{\left(c_{x,a} - c_{x,p}\right)^2 + \left(c_{y,a} - c_{y,p}\right)^2}. \tag{4.15}$$

The two measures need to be combined in some manner to form a single evaluation. Since neither the difference between the means nor the distance between the centroids naturally present themselves as superior measures of particle fitness and both are measures of distance, they are simply summed to provide an overall distance measure,

$$d_p = d_p^m + d_p^c. \tag{4.16}$$

The value of d_p is inversely related to the desired weights – a smaller value should receive a larger weight. Moreover, taking the weight to be the direct inverse of d_t^p would result in the weights sharply increasing toward infinty as d_p goes to zero. As a result, poor particles that happen to have good distance matches could be assigned such high importance factors as to completely wipe out all other particles during the re-drawing step, leading to incorrect localization. To keep the magnitudes of the weights from reaching these levels, the weight assignment was conditional:

$$w_p = \begin{cases} \frac{1}{d_p} & \text{if } d_p > 1 \\ w_{\max}\left(1 - d_p\right) & \text{otherwise} \end{cases} \tag{4.17}$$

where $w_{\max}$ was the maximum possible weight. After some brief preliminary evaluation, a value of $w_{\max} = 2$ was settled on.

4.3.4 Robot State Selection

One practical concern of MCL that has not been thoroughly expanded on in the literature is the actual selection of the robot's state from the set of particles. The matter is either ignored outright, merely observing the clustering of the particles in graphical representations as proof of localization, or the state is simply calculated as the mean of all of the particles, as in [12], for example. This approach assumes that all of the particles converge around a single state, however one of the advantages of MCL over other probabilistic localization methods is the intrinsic ability to track multiple hypotheses through distinct particle clusters. Taking the mean state of particles forming multiple clusters would of course lead to very incorrect localization by selecting a state near none of the clusters.

Dealing with multiple particle clusters, and in fact taking advantage of them, was the goal in [37]. In that research, a set number of completely distinct particle clusters were each tracked with their own MCL computations. The mean states of the clusters were then used as the set of particles to another MCL instance that evaluated the states and selected the best as the true robot state. The problem with this approach is that it effectively requires running multiple MCL instances in parallel, each individually capable of localizing the robot.

For evaluation of the algorithm using off-line computations, it was decided to perform dynamic clustering of the particles and take the mean state of the most populous cluster as the predicted robot state. At each time step, the particles are segregated into clusters as follows:

1. Calculate the infinity norm distance from the particle, p, to the mean position of each cluster. The set of distances is denoted $\left\{d_p\right\}$.
2. If the shortest distance, $min\left\{d_p\right\}$, is less than a threshold, $d_{\max}$, add particle p to the corresponding cluster, C. Proceed to the next particle.
3. If $min\left\{d_p\right\} > d_{\max}$ and the number of clusters already formed, n_c, is less than the maximum number of allowable clusters, n_c^{max}, create a new cluster whose only particle is p. Proceed to the next particle
4. Finally, if $n_c = n_c^{\max}$, add p to cluster c, regardless that $min\left\{d_p\right\} > d_{\max}$.

Using this approach, as opposed to other standard clustering algorithms such as k-means clustering, has the advantage of not requiring a specified number of clusters; key aspects worth investigating with MCL and Augmented MCL in particular are the clustering trends over time, including the number of clusters. The algorithm is not particularly efficient and is only suitable for off-line evaluations, not for on-line robot localization.

Two values are required for the above clustering algorithm: $d_{\max}$ and $n_c^{\max}$. Since the test environment displayed only slight symmetry – primarily where walls and the floor met – it

Table 4.1 The times required to compute one time step of the Monte Carlo localization algorithm using on-line and pre-computed LIDAR beam simulation.

	Computation time [s]	
Number of particles	On-line simulation	Pre-computed simulation
2500	0.118	0.036
10000	0.275	0.084

was not expected to get many genuine clusters that could each be equally likely candidates of the robot state, so the maximum number of clusters to track was set as $n_c^{\max} = 50$. That would provide enough resolution to view clustering over time during global localization. If the particles are forming more than 50 clusters, they are effectively still spread out across the state space. The value of the maximum distance between a particle and a cluster's mean for that particle to be considered part of the cluster was set to $d_{\max} = 500$ mm. This value was rather arbitrarily chosen as coarse enough to group genuinely clustered particles together but fine enough to allow for multiple distinct clusters relatively near one-another.

4.4 Augmented MCL Results

To evaluate the Augmented MCL algorithm, several aspects must be considered: computation time, localization accuracy and the ability to recover from incorrect localization. Discussions of all three will be presented in order.

4.4.1 Computation Time

The time required for computing the motion and measurement models for the full set of particles is the primary deciding factor when selecting the number of particles to use. The calculation times for on-line versus pre-computed simulated LIDAR scans for the measurement model were calculated for two particle set sizes. Unfortunately, the Simulink model could not be compiled for real-time calculation with particle sets larger than 10000 particles, due to an "unknown error", so only sizes of 2500 and 10000 particles were used. The calculation times are presented in Table 4.1.

Clearly pre-computing the LIDAR simulations greatly reduces computation time, by approximately 70%. Since the LIDAR is able to communicate data wirelessly at 0.12 s periods, the goal would be to operate at this rate. Using on-line computation, then, would require that only 2500 particles be used. Pre-computing the scans, on the other, hand could use more than 10000 particles if the program could be successfully compiled. Extrapolating from the times in Table 4.1, potentially 15000 particles could be used.

4.4.2 Localization Accuracy

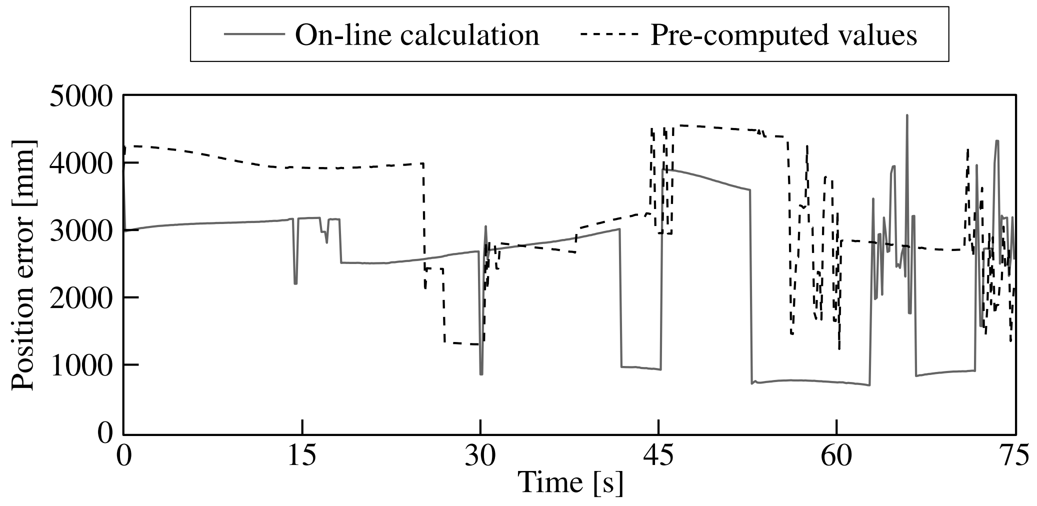
Naturally, localization accuracy is the most important aspect to be evaluated. Though the localization program could only be compiled for real-time operation with particle counts up to 10000, it could be run under Simulink's "Normal"[2] mode with larger particle counts. Despite being unable to operate in real-time, which would be required for in-flight operation, for the of sake evaluation the algorithm was run with 50000 particles for both on-line and pre-computed scenarios, as well as with the 2500 and 10000 particle scenarios above. The position of the airship as calculated by MCL was selected as the mean position of the particles comprising the largest particle cluster. The errors from the position determined by the VICON system, which was treated as ground-truth, for all six scenarios are plotted in Fig. 4.3.

Several features are immediately apparent from the plots. First, there are sharp discontinuities in the errors for all scenarios. These are times at which the cluster sizes changed such that a different cluster became the largest, and thus the error from the true position suddenly changed. These cluster changes will be expanded on in the following section.

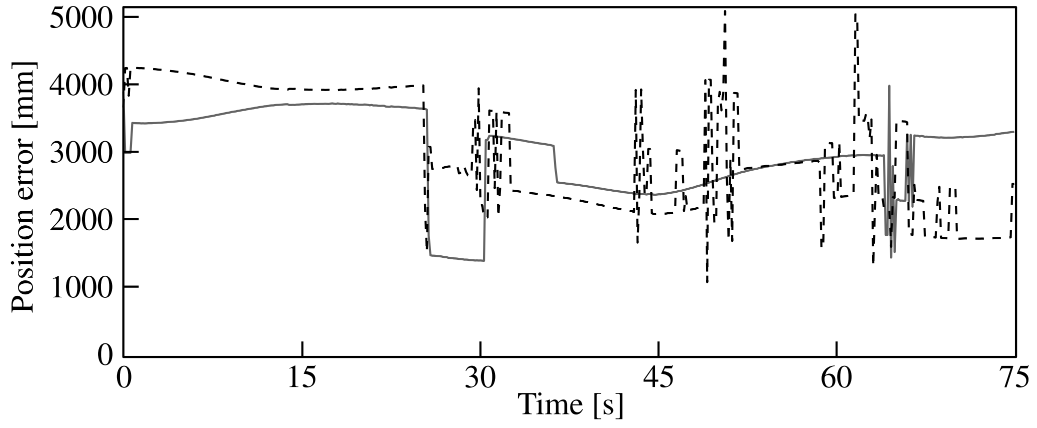
The second observation to make is that no clear superior option between on-line simulation of the LIDAR beams and the pre-computed value scenario presents itself. Though this may indicate that simulated LIDAR beams need not be very accurate, it may more likely be related to the third observation, that the localization in all scenarios effectively failed, barely achieving errors of less than 1 m. With such large localization errors, it is impossible to say whether the pre-computed LIDAR simulations really do perform as well as on-line LIDAR simulations.

The final observation to make is that the localization does show improvements with greater particle numbers. With 50000 particles, the localization errors reach lows of approximately 0.5 m, temporarily at least, for both on-line and off-line calculations. That the results did show improvement with increased particle counts suggests that the particle deprivation problem played a role in the poor results. This is largely attributable to the increased state space dimensionality of the airship compared to the planar vehicles used in most applications of MCL. The increased dimensionality requires an exponentially larger number of particles to achieve a comparable particle density.

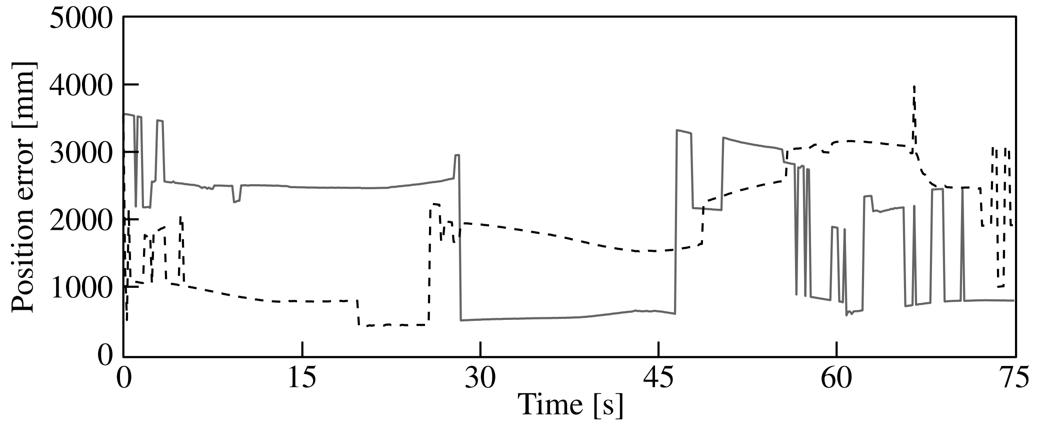
[2]For real-time computations, Simulink's "External" mode was used along with a real-time kernel.



(a) 2500 particles



(b) 10000 particles



(c) 50000 particles

Figure 4.3 Monte Carlo localization errors for on-line LIDAR simulation and pre-computed scans. Errors were calculated as the distance from the mean position of the largest particle cluster to the position measured by the VICON system.

4.4.3 Particle Clustering and Error Recovery

The error plots in Fig. 4.3 showed sharp discontinuities in the error values. To investigate these further, Fig. 4.4 overlays the cluster counts onto the error measures for the on-line simulation case. The pre-computed value scenario showed the same patterns, so the plots have been omitted for the sake of brevity.

As is clearly visible, nearly all of the discontinuities in the localization error occur together with large cluster counts. The large cluster count spikes are caused by the Augmented MCL algorithm determining that the localization is becoming worse and new random particles should be added. These incorrect localizations are effectively unintentional occurrences of the kidnapped robot problem, from which Augmented MCL should be able to recover, based on the results.

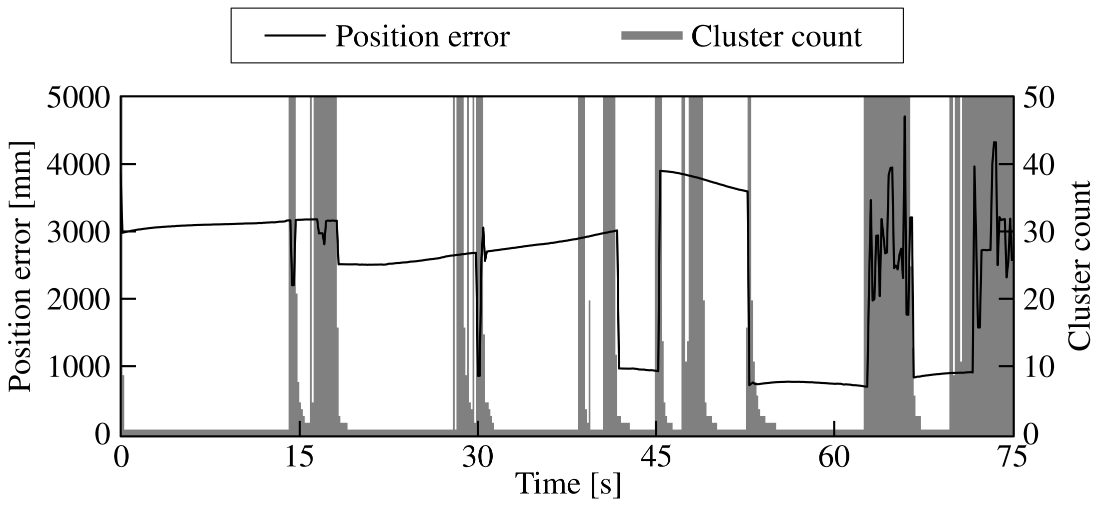
What's more, after almost every cluster spike the localization improved, suggesting that MCL is selecting the best available particles. The problem lies in the fact that the localization offered by the cluster does not improve with time – the errors either stay constant or become worse. The weak motion model is likely to blame here; with an accurate motion model the localization should gradually improve with time. The poor motion model does not track the airship's motion accurately, so particles that were relatively near the airship at the time of the cluster spike gradually travel away from the airship, leading to ever-worse localization and eventually to Augmented MCL instigating another cluster spike.

4.5 Conclusions on Monte Carlo Localization

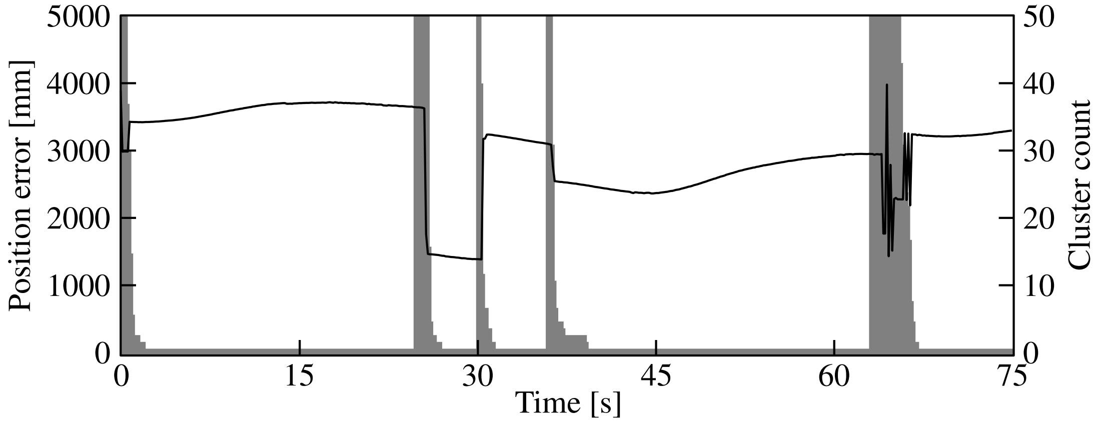
The Augmented MCL algorithm was implemented to perform off-line global localization of the airship. Orientation was tracked by an inertial measurement unit while a laser range scanner was the sole exteroceptive sensor used. Two approaches to LIDAR simulation for the particles were investigated: on-line calculation based on the exact pose of the particle and pre-calculating the LIDAR beams for a discretized subset of the state space.

The localization was not successful, for several reasons. The particle deprivation problem played a major role, since the number of particles to use was restricted by the real-time calculation requirements to 2500 with on-line calculation and potentially 15000 if using pre-calculated values, though the algorithm failed to compile for real-time calculation with more than 10000 particles. If efficiency could be improved, or more powerful computing hardware available, localization accuracy could be improved, as shown by the improved accuracy with 50000 particles.

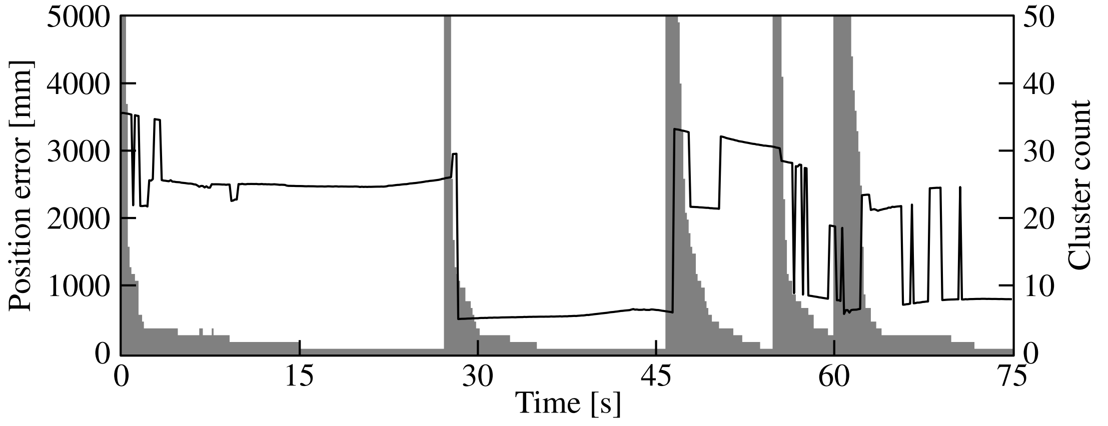
The other major reason for localization failure was the motion model. Rather than



(a) 2500 particles



(b) 10000 particles



(c) 50000 particles

Figure 4.4 Monte Carlo particle clustering for on-line LIDAR simulation. Errors are the same as those shown in Fig. 4.3. Cluster counts were determined as described in §4.3.4.

tracking the true motion of the airship well enough for localization to improve over successive iterations of estimating the motion and then evaluating the particles states, the motion model was at best good enough for the localization errors to stagnate, but generally the errors increased with time.

Though the localization failed, the algorithm itself showed signs that it would be capable of localizing the airship with improved implementation. The Augmented MCL algorithm reliably added new particles to the system if the localization accuracy degraded, as shown by the clustering of the particles. This suggests the ability to recover from incorrect localization and the kidnapped robot problem. Moreover, after adding new random particles to the system the localization generally improved, suggesting that with an improved motion model and greater particle density MCL would be able to localize the airship.

Chapter 5

Conclusions and Future Work

Two approaches to localization and navigation of the spherical holonomic indoor airship of the Aerospace Mechatronics Laboratory have been investigated. Vision-based navigation using a wireless camera mounted to the airship and optical flow algorithms, and probabilistic Monte Carlo localization using a laser range scanner mounted to the airship. Though neither approach ultimately proved truly successful, several valuable insights were gained for furthering work in this area.

5.1 Conclusions

5.1.1 Optical Flow and Vision-Based Navigation

Both optical flow algorithms implemented and investigated in Chapter 3 proved to be incapable of independently capturing the motion of the airship. The relatively poor image quality provided by the wireless camera and the large amount of very noisy images it produced could not be overcome with simple image filtering methods. The unsteady motion of the airship, including considerable rotational motion proved too much for the basic optical flow algorithms.

Ultimately, the only conclusions that can be drawn from the research are that the algorithms investigated failed and achieving vision-based navigation for the airship is a challenging problem that will require a significantly more advanced approach.

5.1.2 Probabilistic Localization

The airship localization research of Chapter 4 showed more promising results, however it was not yet suitable for real-time integration with airship's controller. The algorithm produced results that suggest it was operating as intended, but the limitations of the computational power and the poor motion model could not be overcome to achieve truly accurate localization.

One major challenge for the Monte Carlo localization algorithm was computational. Using standard computer hardware for real-time MCL in three-dimensional environments requires too many compromises to be truly feasible. As implemented, the localization clearly suffered from the particle deprivation problem. The other major failing was the motion model, which was unable to accurately track the airship's motion, so the ideal scenario of the motion and measurement models complementing each-other to achieve localization was never achieved. Even as MCL selected the best available particles, their states relative to the true airship state did not improve over time because of the poor motion model.

5.2 Recommendations for Future Work

Vision certainly presents enticing possibilities for the navigation of an aerial platform. Beyond the relationship to human navigation, the amount of information potentially available in the small and lightweight package of a small wireless camera cannot be matched by other sensors. Extracting that information from the video data, however, is the primary challenge. To simplify the problem, placing known landmarks in the environment and navigating based on their appearance in the video data may be a more immediately feasible goal. Though this would be a regression from the ability to fly in any environment as is our final goal, the algorithm could potentially be generalized to identifying, locating and tracking the locations of previously unknown objects in the environment. More general still, performing simultaneous localization and mapping using the camera is another option. To achieve such a task, though, the airship's in-flight stability must first be improved.

Another alternative approach would be to use a pair of cameras for stereo vision. Importantly, distances to obstacles could be calculated – a key piece of data missing from monocular vision without known landmarks. As an example of the utility of distance measures beyond simply knowing distances to obstacles, an improved optical flow algorithm together with a distance metric could be used to measure the airship's velocity.

Monte Carlo localization of the airship using a laser range scanner needed improvements in two key areas to be useful: the motion model and the computational efficiency of the measurement model. An improved and verified dynamics model of the airship would likely lead to significant improvements immediately. Incorporating adjustable parameters to the dynamics model as in [40] could lead to even greater improvements. Other approaches to estimating the airship's motion should also be investigated, such as using the LIDAR's data to measure incremental position changes [19].

To tackle the computational efficiency of the measurement model, the most promising way forward would be on-line calculation of simulated LIDAR scans using highly-parallel processing. Calculating hundreds-of-thousands of beam distances per time step naturally presents itself as a problem for graphics processing units (GPUs), which are architecturally designed for highly-parallel vector calculations. Implementation on such hardware would not necessarily be straight-forward, however. Currently the Simulink environment has only limited support for using GPUs, and no support for real-time functions incorporating GPUs.

The alternative of pre-computing LIDAR scans and storing them in memory is simply impossible for 6-DOF operation in even modestly sized environments. In theory the data could be stored on a hard drive and only loaded into memory as necessary, but during global localization the whole data set would be required, while not loading any data into memory and instead reading values directly from a hard drive would introduce too much latency to be performed hundreds-of-thousands of times per second.

Finally, an exciting area worth investigating is so-called sensor fusion: intelligently combining data from multiple different sensors to complement each other. Examples of such work have already been presented, such as using data from an IMU to de-rotate images for optical flow [30], or combining IMU data with LIDAR data to form an improved motion model [19]. Other possible combinations would be combining the LIDAR with the wireless camera to again assign distance values to image, or going even further and combining all three sensors, the camera, the LIDAR and the IMU, to all complement each other. Of course, the challenge with all of these sensor fusion approaches is efficiently combining the data such that they do complement each other.

On a methodological note, due to time constraints, the work of Chapter 4 proceeded from performing planar localization on purely simulated data directly to attempting 4-DOF localization using real flight data. Further investigation should be performed to evaluate each step of the process, such as performing 4-DOF localization using purely simulated data. To evaluate the effects of the motion model, the algorithm should also be run using real flight data, but with the airship's true relative motion as measured by the VICON system used in place of the dynamics-based motion model. Performing such tests would help identify potential weaknesses and verify the feasibility of the approach.

The localization and navigation of the holonomic indoor airship proved to be a challenging problem. The unique design of the airship presents very interesting possibilities for research, but also poses unique challenges when implementing navigation and localization techniques developed primarily for ground-based robots operating in a plane. Unsteady motion with large rotational components and severe drift of the airship, all within three-dimensional space, are examples of such challenges. Overcoming those challenges would require significant work, but is required to achieve the full potential of indoor airship as a stand-alone robot.

References

[1] Ahrens, S., Levine, D., Andrews, G., and How, J. P., 2009, "Vision-Based Guidance and Control of a Hovering Vehicle in Unknown, GPS-denied Environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Kobe, Japan, pp. 2643–2648.

[2] Bachrach, A., de Winter, A., He, R., Hemann, G., Prentice, S., and Roy, N., 2010, "RANGE – Robust Autonomous Navigation in GPS-denied Environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, USA, pp. 1096–1097.

[3] Barron, J. L., Fleet, D. J., and Beauchemin, S. S., 1994, "Performance of Optical Flow Techniques," *International Journal of Computer Vision*, **12**(1), pp. 43–77.

[4] Beyeler, A., Mattiussi, C., Zufferey, J.-C., and Floreano, D., 2006, "Vision-based Altitude and Pitch Estimation for Ultra-light Indoor Microflyers," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Orlando, Florida, USA, pp. 2836–2841.

[5] Borges, G. A. and Aldon, M.-J., 2003, "Robustified Estimation Algorithms for Mobile Robot Localization Based on Geometrical Environment Maps," *Robotics and Autonomous Systems*, **45**(3), pp. 131–159.

[6] Bourquardez, O., Mahony, R., Guenard, N., Chaumette, F., Hamel, T., and Eck, L., 2009, "Image-Based Visual Servo Control of the Translation Kinematics of a Quadrotor Aerial Vehicle," *IEEE Transactions on Robotics*, **25**(3), pp. 743–749.

[7] Campos, M. F. M. and Coelho, L. d. S., 1999, "Autonomous Dirigible Navigation Using Visual Tracking and Pose Estimation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Detroit, Michigan, USA, **4**, pp. 2584–2589.

[8] Çelik, K., Chung, S.-J., Clausman, M., and Somani, A. K., 2009, "Monocular Vision SLAM for Indoor Aerial Vehicles," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, St. Louis, Missouri, USA, pp. 1566–1573.

[9] Courbon, J., Mezouar, Y., Guenard, N., and Martinet, P., 2009, "Visual Navigation of a Quadrotor Aerial Vehicle," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, St. Louis, Missouri, USA, pp. 5315–5320.

[10] Cuperlier, N., Quoy, M., and Gaussier, P., 2006, "Navigation and Planning in an Unknown Environment Using Vision and a Cognitive Map," in *Proceedings of the European Robotics Symposium*, Palermo, Italy, **22**, pp. 129–142.

[11] Davison, A. J., Reid, I. D., Molton, N. D., and Strasse, O., 2007, "MonoSLAM: Real-Time Single Camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **29**(6), pp. 1052–1067.

[12] Dellaert, F., Fox, D., Burgard, W., and Thrun, S., 1999, "Monte Carlo Localization for Mobile Robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Detroit, Michigan, USA, **2**, pp. 1322–1328.

[13] DeSouza, G. N. and Kak, A. C., 2002, "Vision for Mobile Robot Navigation: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **24**(2), pp. 237–267.

[14] Eade, E. and Drummond, T., 2006, "Scalable Monocular SLAM," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Patter Recognition*, New York, New York, USA, **1**, pp. 469–476.

[15] Einsele, T. and Färber, G., 1997, "Real-Time Self-Localization in Unknown Indoor Environments lasing a Panorama Laser Range Finder," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Grenoble, France, **2**, pp. 697–702.

[16] Fleet, D. J. and Langley, K., 1993, "Toward Real-Time Optical Flow," in *Proceedings of Vision Interface*, Toronto, Ontario, Canada, pp. 116–124.

[17] Fox, D., 2001, "KLD-Sampling: Adaptive Particle Filters and Mobile Robot Localization," Tech. rep., Department of Computer Science & Engineering, University of Washington, Seattle, Washington, USA.

[18] González, P., Burgard, W., Sanz, R., and Fernández, J. L., 2009, "Developing a Low-Cost Autonomous Indoor Blimp," *Journal of Physical Agents*, **3**(1), pp. 43–51.

[19] Grzonka, S., Grisetti, G., and Burgard, W., 2009, "Towards a Navigation System for Autonomous Indoor Flying," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Kobe, Japan, pp. 2878–2883.

[20] Gutmann, J.-S. and Fox, D., 2002, "An Experimental Comparison of Localization Methods Continued," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Lausanne, Switzerland, **1**, pp. 454–459.

[21] Gutmann, J.-S., Weigel, T., and Nebel, B., 1999, "Fast, Accurate, and Robust Self-Localization in Polygonal Environments," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Kyongju, South Korea, **3**, pp. 1412–1419.

[22] He, R., Prentice, S., and Roy, N., 2008, "Planning in Information Space for a Quadrotor Helicopter in a GPS-denied Environment," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Pasadena, California, USA, pp. 1814–1820.

[23] Herisse, B., Oustrieres, S., Hamel, T., Mahony, R., and Russotto, F.-X., 2010, "A General Optical Flow Based Terrain-Following Strategy for a VTOL UAV Using Multiple Views," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, USA, pp. 3341–3348.

[24] Hyslop, A. M. and Humbert, J. S., 2010, "Autonomous Navigation in Three-Dimensional Urban Environments Using Wide-Field Integration of Optic Flow," *Journal of Guidance, Control, and Dynamics*, **33**(1), pp. 147–159.

[25] Iida, F., 2001, "Goal-Directed Navigation of an Autonomous Flying Robot Using Biologically Inspired Cheap Vision," in *Proceedings of the 32nd International Symposium on Robotics*, Seoul, Korea, pp. 1404–1409.

[26] Iida, F., 2003, "Biologically Inspired Visual Odometer for Navigation of a Flying Robot," *Robotics and Autonomous Systems*, **44**(3), pp. 201–208.

[27] Ishida, H., 2009, "Blimp Robot for Three-Dimensional Gas Distribution Mapping in Indoor Environment," in *Proceedings of the 13th International Symposium on Olfaction and Electronic Nose*, Brescia, Italy, **1137**, pp. 61–64.

[28] Kang, S., Nam, M., Kim, B., Tsubouchi, T., and Yuta, S., 2003, "A Novel Design and Control of Robotic Wheeled Blimp for Tele-guidance," in *Proceedings of the International Symposium on Computational Intelligence in Robotics and Automation*, Kobe, Japan, **1**, pp. 67–72.

[29] Kawamura, H., Mashito, Y., Toshihiko, T., and Azuma, O., 2008, "Learning Landing Control of an Indoor Blimp Robot for Self-Energy Recharging," *Artificial Life and Robotics*, **12**, pp. 116–121.

[30] Kendoul, F., Fantoni, I., and Nonami, K., 2009, "Optic Flow-Based Vision System for Autonomous 3D Localization and Control of Small Aerial Vehicles," *Robotics and Autonomous Systems*, **57**(6), pp. 591–602.

[31] Kim, J., Bok, Y., and Kweon, I. S., 2008, "Robust Vision-based Autonomous Navigation Against Environment Changes," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nice, France, pp. 696–701.

[32] Ko, J., Klein, D. J., Fox, D., and Haehnel, D., 2007, "Gaussian Processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Roma, Italy, pp. 742–747.

[33] Li, Y. and Nahon, M., 2007, "Modeling and Simulation of Airship Dynamics," *Journal of Guidance, Control, and Dynamics*, **30**(6), pp. 1691–1700.

[34] Liu, C., Yuen, J., Torralba, J. S., and Freeman, W. T., 2008, "SIFT Flow: Dense Correspondence Across Different Scenes," in *Proceedings of the 10th European Converence on Computer Vision*, Marseille, France, **3**, pp. 28–42.

[35] Lucas, B. D. and Kanade, T., 1981, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *Proceedings of the DARPA Imaging Understanding Workshop*, pp. 121–130.

[36] Menegatti, E., Zoccarato, M., Pagello, E., and Ishiguro, H., 2004, "Image-Based Monte Carlo Localization with Omnidirectional Images," *Robotics and Autonomous Systems*, **48**(1), pp. 17–30.

[37] Milstein, A., Sánches, J. N., and Williamson, E. T., 2002, "Robust Global Localization Using Clustered Particle Filtering," in *Proceedings of the Eighteenth AAAI National Conference on Artificial Intelligence*, Edmonton, Alberta, Canada, pp. 581–586.

[38] Montemerlo, M. and Thrun, S., 2003, "Simultaneous Localization and Mapping with Unknown Data Association Using FastSLAM," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, **2**, pp. 1985–1991.

[39] Motoyama, K., Suzuki, K., Yamamoto, M., and Ohuchi, A., 2000, "Evolutionary Design of Learning State Space for Small Airship Control," in *Proceedings of the Sixth International Conference on Simulation of Adaptive Behaviour*, Paris, France, pp. 307–216.

[40] Müller, J., Gonsior, C., and Burgard, W., 2010, "Improved Monte Carlo Localization of Autonomous Robots through Simultaneous Estimation of Motion Model Parameters," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, USA, pp. 2604–2609.

[41] Oh, P. Y., Green, W. E., and Barrows, G., 2004, "Neural Nets and Optic Flow for Autonomous Micro-Air-Vehicle Navigation," in *Proceedings of the ASME International Mechanical Engineering Congress and Exposition*, Anaheim, California, USA, pp. 1–7.

[42] Ohata, Y., Ushijima, S., and Nenchev, D. N., 2007, "Development of an Indoor Blimp Robot with Internet-based Teleoperation Capability," in *Proceedings of the IASTED International Conference on Robotics and Applications*, Würzburg, Germany, pp. 186–191.

[43] Pfaff, P., Burgard, W., and Fox, D., 2006, "Robust Monte-Carlo Localization Using Adaptive Likelihood Models," in *Proceedings of the European Robotics Symposium*, Palermo, Italy, **22**, pp. 181–194.

[44] Rekleitis, I., Dudek, G., and Milios, E., 2003, "Experiments in Free-Space Triangulation Using Cooperative Localization," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, USA, **3**, pp. 1777–1782.

[45] Robert, J., 2008, "Autonomous Capture of a Free-Floating Object Using a Predictive Approach," M.Eng. thesis, McGill University, Montreal, Quebec, Canada.

[46] Röfer, T. and Jüngel, M., 2003, "Vision-Based Fast and Reactive Monte Carlo Localization," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, **1**, pp. 856–861.

[47] Royer, E., Lhuillier, M., Dhome, M., and Lavest, J.-M., 2007, "Monocular Vision for Mobile Robot Localization and Autonomous Navigation," *International Journal of Computer Vision*, **74**(3), pp. 237–260.

[48] Sharf, I., Laumonier, B., Persson, M., and Robert, J., 2008, "Control of a Fully-actuated Airship for Satellite Emulation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Pasadena, California, USA, pp. 2218–2219.

[49] Takaya, T., Kawamura, H., Minagawa, Y., Yamamoto, M., and Ohuchi, A., 2006, "PID Landing Orbit Motion Controller for an Indoor Blimp Robot," *Arificial Life and Robotics*, **10**(2), pp. 177–184.

[50] Thrun, S., Burgard, W., and Fox, D., 2005, *Probabilistic Robotics*, MIT Press, Cambridge, Massachusetts, USA.

[51] Thrun, S., Fox, D., Burgard, W., and Dellaert, F., 2001, "Robust Monte Carlo Localization for Mobile Robots," *Artificial Intelligence*, **128**(1), pp. 99–141.

[52] Tournier, G. P., Valenit, M., How, J. P., and Feron, E., 2006, "Estimation and Control of a Quadrotor Vehicle Using Monocular Vision and Moiré Patterns," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Keystone, Colorado, USA.

[53] Travis, W., Simmons, A. T., and Bevly, D. M., 2005, "Corridor Navigation with a LiDAR/INS Kalman Filter Solution," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, Las Vegas, Nevada, USA, pp. 343–348.

[54] Trucco, E. and Verri, A., 1998, *Introductory Techniques for 3-D Computer Vision*, Prentice Hall, Upper Saddle River, New Jersey, USA.

[55] Ueda, R., Fukase, T., Kobayashi, Y., Arai, T., Yuasa, H., and Ota, J., 2002, "Uniform Monte Carlo Localization – Fast and Robust Self-Localization Method for Mobile Robots," in *Proceedings of the IEEE Conference on Robotics and Automation*, Washington, District of Columbia, USA, **2**, pp. 1353–1358.

[56] Valette, F., Ruffier, F., Viollet, S., and Seidl, T., 2010, "Biomimetic Optic Flow Sensing Applied to a Lunar Landing Scenario," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, USA, pp. 2253–2260.

[57] van der Zwaan, S., Bernardino, A., and Santos-Victor, J., 2000, "Vision Based Station Keeping and Docking for an Aerial Blimp," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Takamatsu, Japan, **1**, pp. 614–619.

[58] Verma, S., Sharf, I., and Dudek, G., 2005, “Kinematic Variables Estimation Using Eye-in-Hand Robot Camera System,” in *Proceedings of the 2nd Canadian Conference on Computer and Robot Vision*, Victoria, British Columbia, Canada, pp. 550–557.

[59] Verri, A. and Trucco, E., 1999, “Finding the Epipole from Uncalibrated Optical Flow,” *Image and Vision Computing*, **17**(8), pp. 605–609.

[60] Wooden, D., Malchano, M., Blankespoor, K., Howard, A., Rizzi, A. A., and Raibert, M., 2010, “Autonomous Navigation for BigDog,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, USA, pp. 1050–4729.

[61] Wulf, O., Wagner, B., and Khalaf-Allah, M., 2005, “Using 3D Data for Monte Carlo Localization in Complex Indoor Environments,” in *Proceedings of the European Conference on Mobile Robots*, Ancona, Italy.

[62] Zhang, H. and Ostrowski, J. P., 1999, “Visual Servoing with Dynamics: Control of an Unmanned Blimp,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, Detroit, Michigan, USA, **1**, pp. 618–623.

[63] Zhang, L., Zapata, R., and Lépinay, P., 2009, “Self-Adaptive Monte Carlo Localization for Mobile Robots Using Range Sensors,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, St. Louis, Missouri, USA, pp. 1541–1546.

[64] Zingg, S., Scaramuzza, D., Weiss, S., and Siegwart, R., 2010, “MAV Navigation through Indoor Corridors Using Optical Flow,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, USA.

[65] Zufferey, J.-C. and Floreano, D., 2005, “Toward 30-gram Autonomous Indoor Aircraft: Vision-Based Obstacle Avoidance and Altitude Control,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, Barcelona, Spain, pp. 2594–2599.

[66] Zufferey, J.-C., Floreano, D., van Leeuwen, M., and Merenda, T., 2002, “Evolving Vision-Based Flying Robots,” in *Biologically Motivated Computer Vision*, H. H. Bülthoff, S.-W. Lee, T. A. Poggio, and C. Wallrave, eds., no. 2525 in Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany, chap. 59, pp. 592–600.