Laura Corvese
ME 740
Optical Flow for Autonomous Robot Control

1 Introduction

The use of robots in GPS denied, dangerous, or cluttered environments has become relevant for use in applications such as search and rescue, tracking, or surveillance. Most autonomous robots rely on GPS for positioning, and sensors for collision avoidance. However many environments which are of interest for the use of these robots, are very likely to be GPS denied, and cluttered. The clutter will cause false readings from the range sensors, and the inability to use GPS will mean the robot will not know where it is in the environment. It is in cases like this where a purely vision based autonomous robot would come in handy. It is the goal of this paper to provide the framework and necessary background for the use of optical flow in real time feedback control. This paper will also discuss the current state of the art of vision based autonomous robots using optical flow for collision avoidance, with a specific focus on the work of Seebacher in [23].

Optical flow is the quantification of the apparent motion of objects with respect to a moving observer. Optical flow is used to show how a key point, or object, of interest is moving from one frame in a video to the next. These vectors can then be used to calculate something known as Time to Transit (TTT) which describes how long it will take before the object being tracked will cross the image plane. This can be computed without knowing the actual distance to the object or the size of the object itself, making it an excellent parameter for feedback control. It has been shown in several studies that animals such as bats and bees actually use optical flow and time to transit to navigate their environments, proving the relevance of studying this topic. Research in this area will be discussed in Section 4.

Using optical flow for real time autonomous robot control presents the challenge of needing speedy and efficient implementations of the required algorithms. This rules out implementation of dense optical flow as it is extremely computationally expensive. The Lucas Kanade method for optical flow calculations lends itself nicely to a real time application, but it requires predetermined points of interest. This means that in addition to the optical flow algorithm, feature detector, descriptors, and matchers need to be used to find, describe, and optimize key points of interest within a video frame. Currently, there are many choices for feature detectors and descriptors, each with their own pros and cons. In this work, feature detectors and descriptors such as FAST, FReaK, and BRISK will be discussed and evaluated on their applicability to real time control applications.

The goal of this paper is to explore the current state of the art of the use of optical flow for autonomous robot control through the deep study of the work of Paul Seebacher in [23]. An in depth review of sparse optical flow methods and required feature detector, descriptor, and matcher components will be explored, and it will be studied how these components come together to create feedback control. Experiments are run to test the computation times of different methods of feature detectors, descriptors, and matchers in a full implementation of the optical flow control system, in open loop, on a PC.

This paper is organized as follows. Section 2 will cover the history, background, and basic ideas of optical flow. Section 3 will discuss the various feature detector, descriptor, and matcher methods, their history and background, and their relevance to computer vision applications. Section 4 will discuss the motivation behind using optical flow for autonomous robot control, and the state of the art of this area, as well as provide a comprehensive description of how to implement it based on [23]. Results of extensive feature detection and descriptor testing for real time feedback control will be discussed in Section 5, along with preliminary image segmentation testing using

Seebacher's implementation. Finally, Section 6 will discuss future work, goals, and conclusions.

2 Optical Flow

Optical Flow (or sometimes Optic Flow) is a quantification of the aparent motion of objects with respect to a moving observer. These motion vectors diverge from the image focus of expansion (which can be thought of as the horizon). Optical flow first appeared in the literature in its most basic form in the 1950s, where researchers began studying relative motion of objects and viewers. It saw much further development in the late 70s and early 80s when it was formally introduced by Horn and Schunck [9]. This concept is still a heavily researched area due to its connections with biology and its applications to autonomous robot control. This section will focus on an explanation of a few different optical flow calculation methods, as well as a discussion of their relevance to real time applications.

Two classical methods for computing optical flow via differential methods will be explored. These methods are those of Horn-Schunck [9] and Lucas-Kanade [11]. The differential method utilizes the partial derivatives of both the spatial and temporal components of image sequences. For these methods to work properly, there are several requirements, some specific to each method. The first requirement for differential methods in general is that the intensity of an image is continuously differentiable, in both space and time, to allow for the computation of the partial derivatives. Optical flow can be calculated in both sparse and dense forms. While sparse optical flow will give less information than a dense flow field, it often will be less susceptible to noise, and is certainly faster to compute. We will now explore the two aforementioned methods for calculating optical flow.

2.1 Horn-Schunck Method

The Horn-Schunck method was the first formal procedure for calculating optical flow. This method produces a global, or dense optical flow field. This method, as stated, requires the image to be continuously differentiable. It is also required that there is constant illumination in the sequence of frames and that the reflectance of objects in the sequence varies smoothly. This essentially means that an pixel that is being tracked has the same intensity in one image as it does in the next. The authors use the notation that E(x,y,t) is the image brightness of a particular point at time t. Using the constraint of constant illumination, the authors assert that:

$$\frac{dE}{dt} = 0 \tag{1}$$

Which allows them to develop the equation:

$$\frac{\delta E}{\delta x}\frac{dx}{dt} + \frac{\delta E}{\delta y}\frac{dy}{dt} + \frac{\delta E}{\delta t} = 0 \tag{2}$$

From here, the authors set

$$u = \frac{dx}{dt}, v = \frac{dy}{dt} \tag{3}$$

Which represent the optical flow velocity vectors. This is then formalized to the optical flow equation:

$$E_x u + E_y v + E_t = 0 \tag{4}$$

Where E_x, E_y, and E_t are the partial derivatives of the intensities with respect to x, y, and t respectively. The authors impose a smoothness constraint which requires objects in a neighborhood

to move similar to each other (an intuitive assumption since neighborhoods of pixels should represent an object which will move uniformly), which allows them to minimize the sum of squares of the Laplacians of x-y components of optical flow (in simple cases both Laplacians are equal to zero for both u and v). To retrieve pixels for evaluation, the image is segmented into a square grid and partial derivatives are estimated at a point in the center of a cube, which is formed by eight measurements. Then, the sum of errors in (4), the equation for rate of change of intensity, is minimized, and an estimate of the departure from smoothness in velocity flow are calculated. The Laplacian is estimated by subtracting the value at a point from a weighted average of the values at neighboring points. This leads to two equations for each pixel in the image, for the difference of flow at a point from the local average. The authors suggest an iterative solution which computes new velocity estimates from estimated derivatives and averages of previous velocity estimates (from previous frames if available). This will slightly reduce computation time. Any regions that have a zero brightness gradient (meaning the region is uniform) have their velocities set to those of their neighbors. Essentially this means that the entire neighborhood is moving uniformly.

This method develops a dense optical flow field with velocity vectors for each pixel in the image for each frame. This will give a very detailed analysis of how objects are moving in the environment, but is computationally expensive and susceptible to noise. For real time applications, it is therefore more desirable to compute optical flow vectors in a more sparse manner.

2.2 Lucas-Kanade Method

Unlike the Horn-Schunck method, the goal Lucas Kanade method is to determine the motion vectors of particular pixels of interest in successive images rather than every pixel. In order to use the Lucas Kanade method, several assumptions are required. It is assumed that successive video frames are separated only by a short time, and therefore objects in those frames are not displaced by large amounts. This method also requires that changes are made in a 'smooth' manner, and like the Horn-Schunck method, there must be constant illumination. The overall goal of the algorithm is to predict how a pixel of interest has moved from one frame to the next by explaining local changes in pixel intensity. The optical flow vector through this method is calculated as follows. Suppose we are tracking a pixel located at (x,y) in image coordinates in the first frame of a sequence. In the next frame, the intensity of that pixel location has changed, such that the increase in brightness in the x direction is represented by $I_x(x,y)$ and $I_y(x,y)$ for the y direction. Now it is clear that the total increase in pixel intensity can be explained by moving u pixels in the x direction and v pixels in the y direction giving:

$$I_x(x,y)u + I_y(x,y)v = -I_t(x,y) \tag{5}$$

Where $I_t(x,y)$ represents the difference in intensity of the pixel from the second frame to the first. This equation is identical to the Horn-Schunck equation for optical flow, with the difference in intensity term moved to the other side of the equal sign.

If we only look at one pixel, we have an underdetermined system with one equation and two unknowns; we are essentially trying to figure out how something has moved without considering it's surroundings, which is a nearly impossible task. It is therefore beneficial, and necessary, to track the motion of a pixel based on its neighborhood. If we look at a 3x3 neighborhood of pixels around the pixel of interest, we have 9 equations, one for each pixel, of the form:

$$I_x(x+\Delta x, y+\Delta y)u + I_y(x+\Delta x, y+\Delta y)v = -I_t(x+\Delta x, y+\Delta y) \tag{6}$$

$$\Delta x, \Delta y = -1, 0, 1$$

Which is summarized in vector form as:

$$S[u,v]^T = \vec{t} \tag{7}$$

This equation can be solved using least squares and gives a better estimate of the pixel motion. To use the Lucas Kanade method, however, it is required that key points of interest have been pre-selected ahead of time. These key points must be robust meaning that they need to be found in both the first and second video frames in question. They should also describe an area of interest such as an object. This brings the need for feature detectors, descriptors, and matchers, which will be further discussed in Section 3. Once optical flow has been calculated, we are able to find a critical value called the time-to-transit (TTT).

2.3 Time to Transit

The time-to-transit represents the amount of time before a particular keypoint of interest crosses the image plane. This value can be easily calculated once the optical flow is known, and can be computed without knowledge of the size or distance to the keypoint of interest, making it invaluable for autonomous robot control. Time to transit can also be calculated without knowledge of the observer velocity, however this value must be constant. Research in the area of time-to-transit began in the field of psychology and studying how humans perceive oncoming obstacles or landmarks. In [10], the authors discuss the perception of an observer approaching objects that are not on a collision course, and express how time-to-passage may be appropriately calculated using the ideas of Local and Global Tau. They note that Local Tau is more useful for objects on a collision course with the observer, whereas Global Tau may be better for objects on the peripheral and is therefore more useful for navigating through environments. Studies were also done using time-to-transit and human reactions to needing to break when driving in a car, such as in [14]. Baillieul et. al [22] then use the idea of time-to-contact and relate it to the flight patterns of bats and explain how this can be used for autonomous robot control.

Calculating the time-to-transit for a moving observer and an object can be thought of as a similar triangles problem. In Figure 1, we see a camera with an object approaching it (assuming relative motion where the camera is moving and the object is stationary). This object is 'seen' through the camera as an image created on the focal plane. The distance from the camera to the object is $x(t)$, the focal length of the camera f, d is the diameter of the object, and d_i is the diameter of the object as it appears on the image plane.

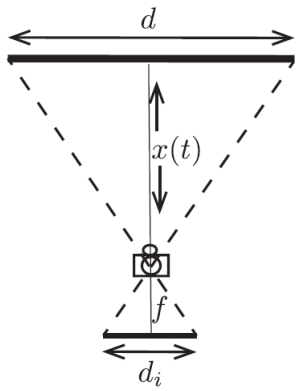

Figure 1: An pinhole camera is approaching an object of diameter d at a constant velocity [22]

The variables in the figure, by similar triangles, are related by the following equation

$$\frac{d}{x} = \frac{d_i}{f} \tag{8}$$

If we suppose the observer is moving at a constant velocity $\dot{x}(t) = v$ we can derive a differential equation to describe the rate of change of the size of the object as it appears on the image plane. If you imagine moving toward an object, it will appear to get larger and larger as you get closer. This is the idea behind the following equation:

$$\dot{d}_i(t) = \frac{v}{d \cdot f} d_i(t)^2 \tag{9}$$

Since we have assumed that the observer is moving at a constant velocity v, the solution to the differential equation is:

$$d_i(t) = \frac{d \cdot f}{x_0 - vt} \tag{10}$$

Where x_0 is $x(0)$, initial distance from the obstacle. The most interesting result of these equations is that time-to-contact (τ) can be calculated without knowing a priori the values of d, x_0 or v (however we still must assume v is constant). This can be done my manipulating the original similar triangles equation (8) to get:

$$d \cdot f = d_i(t) \cdot x(t) \tag{11}$$

And differentiating both sides along with the assumption of constant velocity, we see that:

$$\frac{d_i}{\dot{d}_i} = \frac{x_0 - vt}{v} \tag{12}$$

When $t = x_0/v$ we see that this equation is zero, which represents the 'collision' or the fact that the object has passed through the image plane. At $t = 0$ we have $d_i(t)/\dot{d}_i(t) = x_0/v = \tau$ which is the time-to-contact parameter we sought. This proves that the time to contact can be calculated with only knowledge of the time derivative of the perceived change in 'size' of the object, not needing information on the size or distance to the object. We realize this in terms of optical flow by noting that $d_i(t)$ represents the distance between an object and the focus of expansion (where the optical flow vectors eminate from), and the associated optical flow component along this direction $\dot{d}_i(t)$. We can note that if τ is negative, this means that the objects are retreating, and if it is positive the object is approaching. This lends itself very nicely for feedback control for optical flow based systems as we can navigate an environment without the need for additional sensors. In a practical application we will have information on how a pixel has moved from one frame to the next, in terms of x-y image coordinates, and how much time passed between frames. Using this data alone, we are able to calculate τ. Applications of this will be discussed later, in Section 4.

3 Feature Detectors, Descriptors, and Matchers

As mentioned previously, we are interested in exploring the Lucas Kanade optical flow method. However, in order to perform this method, predetermined keypoints, or features, are required. An image feature is often described as a corner, which is defined as a region in an image which has maximum variation when moved by a small amount. In computer vision applications, it is often important to find features of interest, be it animals, people, or cars for example, for tracking. Finding these features requires something called a feature detector. Most detectors find corners, and objects can be determined through clustering. Once a feature is found, it needs to be described through a feature descriptor to be able to track its motion. Describing a feature is essential to making use of it. Feature descriptors typically use intensity gradients in neighborhoods around the detected features. Finally, once features are detected and described, we need to find a

way to match them between two or more images. Matching methods include using L1, L2 norm or Hamming distance, depending on the type of descriptor. The choice of descriptor heavily determines the amount of time that the matching process will take.

There are many types of feature detector, descriptor, and matching methods in the literature, but care must be taken in choosing them for real time applications. As stated, computation time varies greatly among the different methods. Some methods result in superior accuracy in correctly matching features, invariant to rotations and scale, but suffer from long computation times. In this section some different feature detector, descriptor, and matching methods will be briefly discussed with an emphasis on both accuracy and computation time. For detailed explanations of each type, references will be provided.

3.1 Feature Detectors

There are many feature detectors in the literature, starting from the early Harris Corner detector, to those inspired by biology like the Fast Retina Keypoint algorithm. In the following sections some of the most popular and historically important feature detectors will be discussed along with their applicability to the task of real time computation.

3.1.1 Harris Corner Detector

Harris and Stephens developed a corner detector, later coined the Harris detector, in 1988 [8]. This method finds the maximum difference in intensity for a displacement of (u,v) of a window of interest, in all directions by maximizing:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) = I(x,y)]^2 \tag{13}$$

Where $w(x,y)$ is the window function, $I(x+u,y+v)$ is the shifted intensity, and $I(x,y)$ is the initial intensity. Essentially this means that if an image patch contains a corner, moving it in any direction should yield a large change in appearance. To maximize this, the second term must be maximized by applying Taylor expansion. Once the maximization is solved, the authors developed a scoring metric which will determine if the window may contain a corner. This metric requires computing a determinant and trace of the original window, multiplied by the partial derivatives of image intensity in x and y directions with respect to the displaced window. The eigenvalues of this matrix determine if a region is a corner, an edge, or flat. A flat region will not have much or any change when the image patch is moved in all directions. An edge will have no change in appearance along the edge direction, but changes in other directions, and a corner will have substantial changes in all directions. The Harris corner detector is the most fundamental corner detector and is not typically used in current research. It is introduced here as the groundwork for the feature detector literature.

3.1.2 SIFT Detector

Later, Lowe created the Scale-Invariant Feature Transform (SIFT) detector [16] which improved upon the Harris detector by ensuring the same features would be found if the image was scaled (the Harris detector is rotation-invariant but not scale-invariant). Lowe's work is a combination of feature detector and descriptor. Lowe claims the SIFT method is scale and rotation-invariant and partially invariant to illumination and 3D camera viewpoint. The method uses a cascade filtering approach and can be broken down into four main stages. Stage 1 is the scale-space extrema detection, which uses the difference of Gaussians (a close approximation to the Laplacian of Gaussians) to identify potential areas of interest. The scale-space is necessary to detect corners of many different sizes. Stage 2 is keypoint localization, where at each candidate

location, a detailed model determines location and scale. Keypoints are selected on the measure of their stability. Stage 3 is orientation assignment where local image gradient directions determine the orientation. Finally stage 4 is the keypoint descriptor where local image gradients are measured at selected scale regions around each keypoint. These are then transformed into a representation allowing for levels of local shape distortion and change in illumination. While SIFT features and descriptors are known to give some of the best results in accuracy, they unfortunately take too much time to compute (and later match) for real time operations.

3.1.3 SURF Detector

Another classic feature detector is the SURF detector. To speed up SIFT features, Herbert Bay et. al developed Speeded Up Robust Features (SURF) in 2008 [2]. This method utilizes a fast Hessian detector that relies on integral images for faster computation. An integral image is a description of the intensities of pixels in an image where each entry in the integral image represents the sum of pixel intensities above and to the left of that pixel in the original image. This is useful for speeding up computation time as it reduces calculations required to find the sum of intensities in a region to several short additions. Lowe uses an approximation of the determinant of a Hessian for selecting the location and scale of a keypoint. Box filtering is used to approximate second order Gaussian derivatives, as shown in Figure 2.

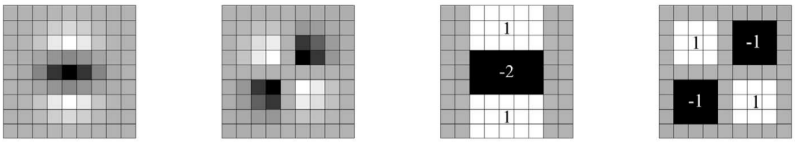

Figure 2: Left to right: the (discretised and cropped) Gaussian second order partial derivatives in the y-direction and the xy-direction, and the approximations thereof using box filters. The grey regions are equal to zero [2]

The filter responses are normalized with respect to the mask size. The scale space is analyzed by up-scaling the filter size rather than iteratively reducing the image size. Like the SIFT method, SURF comes with its own descriptor which describes the distribution of Haar-wavelet responses within a neighborhood of the keypoint. Once again, SURF is known as a very good detector and descriptor, and out performs SIFT in accuracy and speed, but it is still too slow for real time applications like those of interest in this research.

3.1.4 FAST Detector

In 2006, before SURF features were introduced, Rosten and Drummond introduced their FAST detector with specific focus on its use in real time computer vision applications [20]. FAST stands for Features from Accelerated Segment Test. This method considers a circle of 16 pixels centered around a corner candidate 'p' as shown in Figure 3.

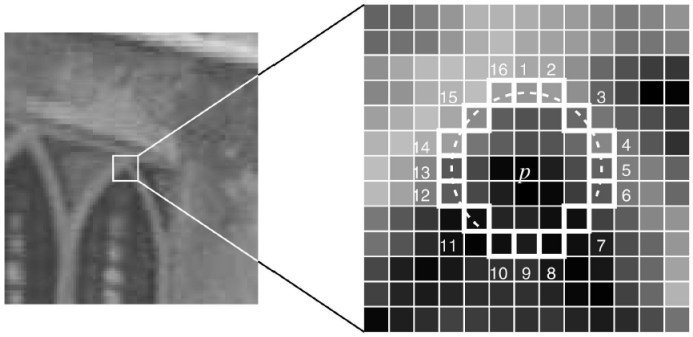

Figure 3: The FAST corner detector [20]

A value of n is selected such that if there exists n pixels in the circle which are brighter than the intensity of the pixel p (I_p), plus some threshold value t, or all darker than I_p minus the threshold t, then p is a corner. The idea behind the method is that all 16 pixels should not need to be tested individually, but only the compass directions. If three of the four satisfy the requirements stated above, then a full segment test can be applied to the remaining pixels. This will quickly eliminate many tests, however there are issues with the model accuracy. Instead, Rosten and Drummond suggest implementing machine learning which learns a corner detector using the 'slow' implementation of checking all 16 pixels in the circle. The machine learning technique separates all pixels in the training image into three subsets (one for darker, lighter, and same intensity than the center pixel), then attempts to extract the pixels with the most information by calculating entropy. This is iterated recursively until the entropy is zero and the corner detector has learned. A decision tree is then converted into nested if statements for corner detection. This strategy of nested if statements make the learned corner detector very efficient at identifying corners, making it a very good candidate for real time computer vision applications.

3.1.5 BRISK Detector

Leutenegger et al. developed the Binary Robust Invariant Scalable Keypoints (BRISK) method in 2012 which was proven to perform well [15]. The BRISK detector, much like the SIFT method, detects keypoints in octave scale space layers, however this method also utilizes the spaces between layers. BRISK searches for a maxima in the image plane and scale space using FAST scores as a measure. Estimates of the true scale of each keypoint in the continuous scale space are then calculated. A 9-16 mask (9 consecutive pixels in a 16 pixel circle) FAST detector is used on each octave and intra-octave. For a point to be considered a corner, the 8 neighboring FAST scores must fulfill the maximum condition. The FAST scores in the layers above and below are also considered. The BRISK method also has its own descriptor, which is based on binary strings, which will be described in the next section. This method is more comprehensive than FAST in that the keypoints are scale invariant, but it is based on the same basic framework. For these reasons BRISK may be useful for real time applications.

3.2 Feature Descriptors

Once features have been detected, they need to be described in order to be found again in subsequent frames. The descriptor is typically a vector that describes the gradient of the pixel intensities around the key point of interest. Varying in dimension, the choice of descriptor plays a large role in the amount of time it takes to match these features from frame to frame, so the choice cannot be taken lightly.

3.2.1 BRIEF Descriptor

The Binary Robst Independent Elementary Features method for description was introduced by Calonder et. al [5]. This method does away with classifier trees and instead creates a bit vector out of test responses. These bit vectors can be 128, 256, or 512 dimensions. 9x9 Gaussian smoothing kernels are chosen to smooth the images. Drawbacks of the BRIEF method are that it does not account for orientation, so it does not perform well when rotations are over 15 degrees. Testing on this descriptor was done by computing descriptors for keypoints of interest, and attempting to match key points to a second image by finding the nearest neighbor. The recognition rates were comparable to other descriptors, where the other descriptors were up to four times larger than the BRISK descriptor. The authors conclude that the method is much faster at matching using the hamming distance than existing descriptors and is a good method as long as invariance to large in-plane rotations is not a requirement.

3.2.2 BRISK Descriptor

As mentioned in the previous section, the BRISK detector also has a descriptor. The BRISK descriptor is composed as a binary string of simple brightness comparisons. The characteristic direction of each keypoint is identified by selecting brightness comparisons with a focus on maximizing descriptiveness. Rotation invariance is achieved through using a sampling pattern of N circles concentric with the keypoint and applying Gaussian smoothing with variance proportional to the distance between points on the respective circle. Smoothed pairs are used to estimate the gradient using short and long distance pairs, and the bit vector descriptor is created by performing short distance comparisons between sampling patterns. The BRISK descriptor is a bit string of 512 bits. Like with the BRIEF method, keypoints described by BRISK descriptors can be matched using hamming distance to reduce computation times.

3.2.3 FReak Descriptor

The Fast Retina Keypoint (FREAK) descriptor was developed by Alahi et al. in [1]. As suggested in the title, the descriptor uses a retinal sampling patttern inspired by the natural vision system, where the pattern consists of circles with increasing point density as you get closer to the center, as shown in Figure 4.

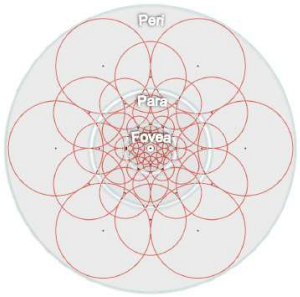

Figure 4: The FReaK sampling pattern as it relates to retinal ganglion cells distribution [1]

These receptive fields are also overlapped to add redundancy, allowing for the use of less receptive fields. A binary descriptor is constructed by thresholding the difference between pairs of receptive fields with their corresponding Gaussian kernels. The binary string is formatted by a sequence of 1-bit difference of Gaussians. The descriptor is parsed in a cascade coarse-to-fine

strategy such that 90% of candidates are discarded in the first step. An object can be determined by a single FreaK descriptor by a region inside it's bounding circle. Rotation invariance is acheived similarly to the BRISK method, however only 45 receptive fields are chosen, considerably reducing the memory requirements. For this method, the descriptor is binary, which allows hamming distance to be used in stead of L1 or L2 norms, which can decrease computation time.

3.2.4 Synthetic Basis Function Descriptor

In 2016, Desai et. al [7] developed an efficient feature descriptor based on synthetic basis functions (SYBA). The principle idea behind using SYBA stems from work in compressed sensing, which is used to encode and decode a signal efficiently and reduce bandwidth and storage requirements. This work uses a binary image patch with randomly selected patterns and compares this image patch to a neighborhood around a keypoint of interest. An example of a SYBA image patch can be seen in Figure 5.

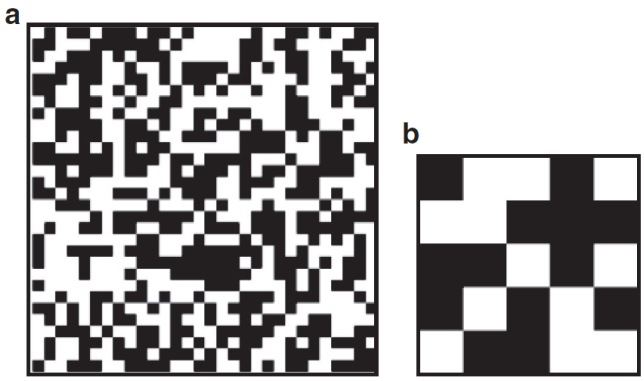

Figure 5: An example of a synthetic basis function image patch where (a) is a 30x30 patch and (b) is a 5x5 patch [7]

The image patch will have approximately 50% black pixels and 50% white pixels and with a 5x5 image patch, 9 different SYBA patterns are required. The authors relate their synthetic basis function patches to 'guesses' in a game of battleship where the patch is trying to guess what the actual distribution is in the image. The number if 'hits' for a region in each of the SYBA patches (the number of times that the pixel in the SYBA matched the pixel in the original image) is saved as the descriptor, such that for the 5x5 SYBA patches, each descriptor will have 9 numbers. Matching between two images is done through a simple hamming distance test. In a comparison study of the algorithm, it was found that SYBA uses less average memory, and has higher accuracy than the methods of SIFT, SURF, and BRIEF.

3.3 Feature Matchers

Feature matching is typically done by brute force using metrics such as L1 or L2 norms. Library matching may also be done which uses a stored library of image features. If a binary descriptor is used such as BRIEF, BRISK, or FReaK, the hamming distance can be used rather than the L1 or L2 norm, which has shown a marked improvement in computation time. A discussion of experiments run on different combinations of feature detector, descriptors, and matchers to find the best set will be detailed in Section 5.

4 Optical Flow and Robot Control

We will now discuss the relevance of using optical flow for autonomous robot control, discuss the current state of the art for the research, and explain, in detail, how to use the Lucas Kanade method to provide real time visual feedback control.

4.1 Motivation and Literature Review

The use of optical flow in control of autonomous robots stems from biological studies of animals such as bats and bees. Baillieul et. al [22] study flying animals such as pigeons, and relate their natural guidance and collision avoidance systems to that of optical flow. The focus is on high speed flights through cluttered environments such as forests. The authors discuss the simple computation of time to contact through optical flow information and its pivotal role in autonomous robot control. In [13], the authors explore the flight trajectories of bats and study the relation to optical flow. This paper introduces the concept of time-to-transit and how it may be used by bats for visual collision avoidance. The work focuses on side facing cameras, mimicking that of the geometry of bat eyes. The authors use their modified version of time to transit to study distance maintenance using optical flow and a combination of control laws.

Milde et. al [18] study the idea of saccades and intersaccades (times of high angular velocity about the yaw axis and periods where the insect's body and head orientation remain relatively constant respectively) inspired by insects to develop an optical flow based collision avoidance system. The authors use optical flow (dynamic vision) sensors to determine 'relative nearness' vectors. The opposite direction of the average of the relative nearness vector is used as the obstacle avoidance direction. This research was tested in open loop only and complete autonomous control was not achieved. Bertrand et. al [3] also draw inspiration from insects such as bees for their optical flow based collision avoidance system. In this paper, the authors discuss how they used a relative nearness map to determine a collision avoidance direction and necessity value (how important is the collision avoidance direction based on 'threat'). The collision avoidance necessity is used to tell the robot how much to turn toward the collision avoidance vector, or to tell it to continue straight. The results of this study are purely computational but simulations show successful navigation of 2D fabricated environments between a start and goal location.

Serres et. al [24] develop a simulation of autonomous control of a 3 DoF hovercraft using optical flow, inspired by the flight of insects. This research uses a wall following strategy such that the robot is able to traverse U and S-shaped corridors. The angle of incidence between the robot and a frontal obstacle is found using to local optical flow measurements and the appropriate collision-free path is followed. As stated this work was a simulation and was not implemented on a real robot. Clearly animals are a key area of motivation for the use of optical flow in autonomous robot control as it has been shown that they use a form of it for their own natural collision avoidance systems. This means that if we can mimic these systems, sophisticated autonomous robot controls can be developed.

Many different platforms have been explored for using optical flow for collision avoidance including ground vehicles, fixed wing air vehicles, quadrotors, and micro UAVs. Early work includes that of Convertino et. al [6] who formulates a region based optical flow estimation for applications in collision avoidance in a general sense. In more recent studies, optical flow has been used for velocity estimation of aerial vehicles such as the studies in [4] [12][17] and [19]. These studies use downward facing cameras and calculate optical flow vectors, from which they extract velocity information for the aerial vehicle. These studies, however, do not include collision avoidance, as the camera is downward facing. Garcia et. al [21] use a ground based robot that performs optical flow calculations across the entire image on a specified grid of pixels. The objects are recognized as areas where the magnitudes of the optical flow vectors are higher. The image is segmented into five columns and time to contact is averaged across each segment. The segment that has the lowest time to contact is decided to have an object and the robot is

told to move away from that segment. Watanabe et. al [25] also utilize a ground robot with four wheels to develop their optical flow obstacle avoidance method. This method recognizes objects through optical flow, then converts their lengths into potential fields. A safe movable area is then extracted and a path is planned accordingly. Collision avoidance with optical flow for aerial vehicles has also been studied. Yoo et. all [26] use Lucas-Kanade optical flow balancing to turn the uav right or left in a simulated environment, with a flow balancing strategy similar to that of Seebacher in [23].

4.2 Seebacher's Implementation

In this research, it is of interest to study the use of optical flow for autonomous quadrotor flight as in [23]. Seebacher built a system which computes sparse optical flow using the Lucas Kanade method, on key points chosen through feature detection, description, and matching, and then calculates the time to transit to use for feedback control of a quadrotor. The process used in Seebacher's method will be detailed below.

The first step to the program is initializing parameters such as feature detector, descriptor, and matching methods, in addition to setting up the desired number of keypoints to track and the parameters for the Lucas Kanade optical flow. In his work, Seebacher used 2000 keypoints for tracking, and set up a back calculation threshold of 25 pixels for the Lucas Kanade back calculation threshold. Next, the processing of frames begins. For the first call, there is only one frame so feature detection and description is run, but no matching or optical flow calculations can be made, because there is only one reference frame. It is therefore not until the second frame where the system begins its full implementation. Once more than one frame has been taken, it is checked to see if the number of keypoints is equal to the desired number, initialized in the beginning of the program. Since the vehicle is moving, key points will often drop out of the image frame, which necessitates adding new points to keep the total number at the desired level. Points may also be dropped in the filtering step when keypoints are filtered based on their ability to be matched between frames (a process that occurs later). If the number of points has dropped below the threshold, the program will detect and describe new keypoints. An attempt is made to match these points to the previous frame, however this might not immediately be able to happen. Next the key points from the previous image are extracted and points that are able to be matched between the previous and current frame are considered for optical flow calculations. The matching technique finds the closest two key points in the second frame and employs Lowe filtering to filter out bad matches based on their ratio distance. This ensures that keypoints that drop out or are not matched in the next frame will not be used, and those that are used are 'good' points.

Once the points to use are determined, the Lucas Kanade optical flow is calculated using the OpenCV implementation. The first pass uses forward optical flow, that is the flow from the previous frame to the current frame using the 'good' keypoints from the previous frame. The LK flow returns where these keypoints have moved to in terms of their x-y positions. Each of the newly calculated points are checked to ensure that they did not pass beyond a desired threshold edge margin. This means that if between two frames, the LK flow predicts that a keypoint moves very far, the prediction is most likely flawed (especially since LK flow only works for small motions). Now the LK flow is performed backwards using the points found in the forward flow for the current image, to see if it correctly matches the points in the previous image that were fed to the forward optical flow calculation. The difference in position between the key points that were fed to the forward optical flow and those that are returned from the backward optical flow is calculated, and the only points that are accepted are those that stay within the acceptable pixel deviation. At this point the number of keypoints may have dropped, once again, below the desired number due to filtering, so new points may be detected an described here.

Now that the optical flow calculations are complete, the feedback control calculations must be done. First the focus of expansion (foe) must be found, since this is where the optical flow vectors emanate from. In Seebacher's work, he attempts to finding the foe for each frame seperately,

however the technique is not robust and sometimes may revert to using the center of the image. This is an area where there is potential for further work. The distance between old and new points, where the new points were determined through optical flow, is calculated and subtracted from the focus of expansion to give x-y position from the foe. This value is divided by the time between frames to give $\frac{dx_dy}{dt}$. This is called the loom. Next the coordinates are transformed from Cartesian to cylindrical for calculation of time to transit. A lot of thought was put into the decision to use cylindrical coordinates. A quadrotor is an underactuated device, such that it needs to use pitch and roll motions to move forwards and backwards, and left and right. This presents a particular challenge when dealing with optical flow since rotational flow will be introduced as the quadrotor makes side to side motions down the corridor. Converting to cylindrical coordinates (radial) will determine the rate at which keypoints travel toward the observer without consideration of the angular components, which would muddle the control calculations. Once dr, the difference between the key points in the two frames in cylindrical coordinates, is found, then $\frac{dr}{dt}$ is calculated. Finally, the time to transit is calculated as $\frac{r}{\frac{dr}{dt}}$.

Next the image segmentation is set up such that the image is separated into m rows and n columns. Histogram binning is used to separate the tau calculations into these grid cells. A threshold for time to transit is set such that if TTT is above this threshold, the 'object' or key point is ignored for control calculations. This means that objects that are far away that may have an extremely high TTT do not factor in, allowing for looming objects to play a larger role in control calculations. The average TTT is calculated for each of the image segments, and these values are used for determining controls. In Seebacher's implementation, the image was segmented into one row and three columns. The middle column was not used for controls and the right and left columns were used for left-right flow balancing. The sign of the average tau values for the left and right columns are checked to see if the flow is uniform or divergent (representing forward or backward motion respectively). If the flow is uniform, the current motion is determined by the sign of tau (if the left tau is negative, the vehicle is moving leftward). A left moving vehicle is assigned a *motion* value of -1, and a right moving vehicle a *motion* of 1. The vehicle control is then calculated as:

$$control = motion * (\frac{1}{left\_tau\_average} + \frac{1}{right\_tau\_average}) \tag{14}$$

If the flow is divergent, there is no *motion* assigned and the control is calculated as:

$$control = motion * (\frac{1}{left\_tau\_average} - \frac{1}{right\_tau\_average}) \tag{15}$$

This essentially tells the robot to move away from the direction which has a higher TTT. The sign of the control determines whether the quadrotor moves left or right. This results in a left and right balancing control which is able to navigate a quadrotor down a straight corridor without colliding with either wall or objects along the walls in the hallway. However since the middle column is not used, and the motion control is purely right-left balancing, the quadrotor cannot avoid collisions with objects that are directly in front of it, such as the wall at the end of a corridor. This consideration and others will be discussed in Section 6.

5 Testing and Experiments

In this section the time testing and image segmentation tests will be described. In order to run the tests, Seebacher's optical flow software needed to be modified and updated to run on my lap top. To get this working I needed to learn the python language (since I did not have any prior experience with it). Once modifications were made, libraries were installed, and updates were put in place, the system was running smoothly and as expected.

5.1 Time Testing

In order for an optical flow method to be usable in autonomous feedback control, all computations must be efficient. It was therefore of interest to test several combinations of detectors, descriptors, and matchers for computation time. This testing was done in the same environment that Seebacher tested the quadrotor. To complete this testing, the optical flow program was updated and modified to run on a Dell XPS 9560 with an Intel quad-core i7-7700HQ processor at 2.8GHz and 16GB ram, and the video stream was taken from the webcam. The laptop was placed on a rolling cart and video was taken as it was pushed down the hallway for 30 seconds at a slow speed. A video still from the time testing can be seen in Figure 6.

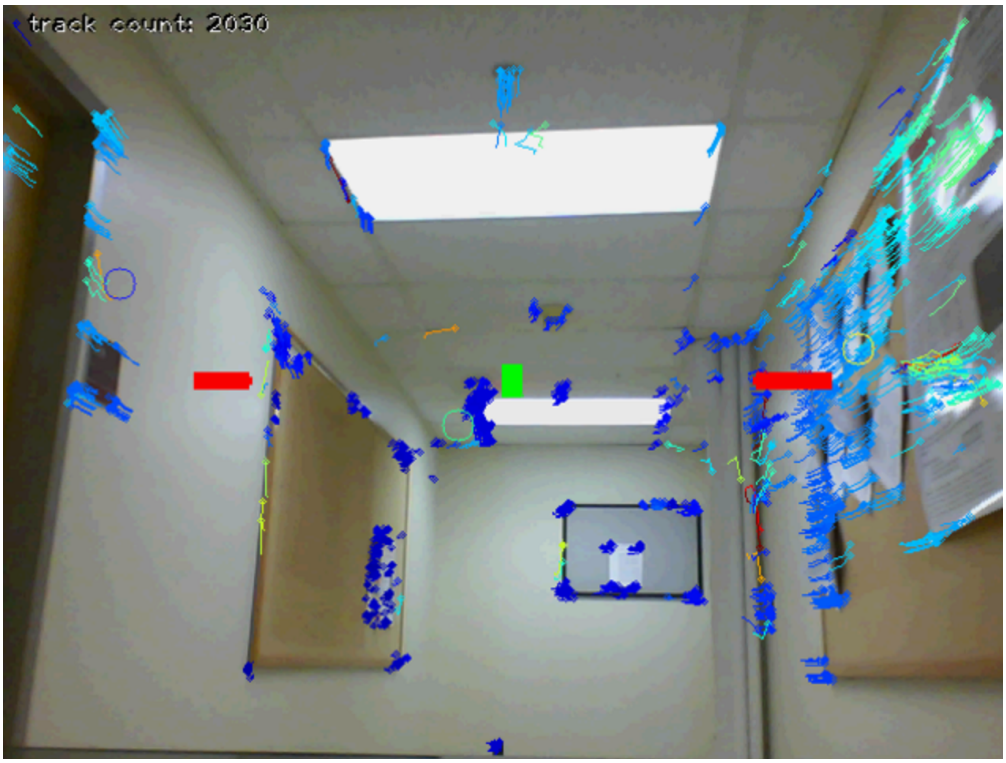

Figure 6: A still frame from the time testing experiments

Tests were run for different combinations of feature detectors, descriptors, and matchers. The feature detectors that were tested were FAST, GRIDFAST (a grid based implementation of FAST), BRISK, and OpenCV's ORB (a non-patented version of SIFT and SURF). The feature descriptors that were tested were FReaK, BRISK, ORB, and BRIEF. Unfortunately an implementation of the SYBA descriptor could not be found, however in the future it is of interest to potentially study this method since it is the newest of them all and shows promising results, especially in its small memory requirements. Matching was done through brute force and used L2 norm for all descriptors, and separate tests were run using hamming distance for BRISK, BRIEF, and FReaK descsriptors. For each test run, the average computation time for each step (detecting, describing, and matching) was computed. It should be noted that these tests simply tested computation times of the detectors, descriptors, and matching methods and did not consider the accuracy. The accuracy of each method is explored in their respective papers. The results for the combined computation times of detection, description, and matching is shown in Table 1 for hamming distance. Where the rows represent the descriptors and the columns the detectors.

Descriptor/Detector	BRISK	FAST	GRID FAST	ORB
BRIEF	0.0067	0.0057	0.0039	0.0075
BRISK	0.0064	0.0064	0.0042	0.0073
FReaK	0.0062	0.0060	0.0038	0.0081

Table 1: Combined times of detecting, describing, and matching keypoints with Hamming distance matching

The results for the combined times of detection, description, and matching using the L2 norm matching method are shown in Table 2.

Descriptor/Detector	BRISK	FAST	GRID FAST	ORB
BRIEF	0.0062	0.0056	0.0038	0.0067
BRISK	0.0059	0.0068	0.0042	0.0072
FReaK	0.0060	0.0058	0.0038	0.0070
ORB	0.0106	0.0055	0.0039	0.0073

Table 2: Combined times of detecting, describing, and matching keypoints with L2 matching

Clearly we can see that with Hamming distance matching, the best combination for speed is the GridFast and FReaK combination. However it is interesting to note that the GridFast/BRIEF combination had the same average time when using the L2 distance. However it should be noted that each time test was only run once for each detector/descriptor/matching method. Future testing could result in a greater gap between the methods, but for now it is sufficient for understanding which combinations to choose when dealing with a system that must make these calculations in real time.

5.2 Image Segmentation Testing

The next area for testing and experimentation was with the image segmentation strategy. As mentioned, Seebacher segmented the image into three columns, where the left and right columns were used for time-to-contact balancing and the middle column was disregarded due to the errors and confusing control commands that it may create. It was of interest to attempt to segment the image plane into three rows and three columns such that left and right balancing could be achieved at the same time as vertical balancing. To do this, the segmentation was changed and additional calculations were added to compute average TTT across the entire left and right columns (all three segments), since we still wanted to do left and right balancing using these columns. The same controls were calculated for the left and right balancing as before, and a new set of controls was added for vertical balancing. Here the loom signal and control were computed the same way as for the horizontal balancing. I then updated the visualization to add bars for the vertical loom and control signals.

Testing this segmentation proved to be a challenge however. With the current testing platform being a PC on a cart, I was unable to truly test the vertical balancing controls as the computer could not be moved in the z direction without introducing a large amount of extra vibrations. The test was run in the same way as the previous tests, by simply pushing the cart down the hallway. A still frame for this testing can be seen in Figure 7.

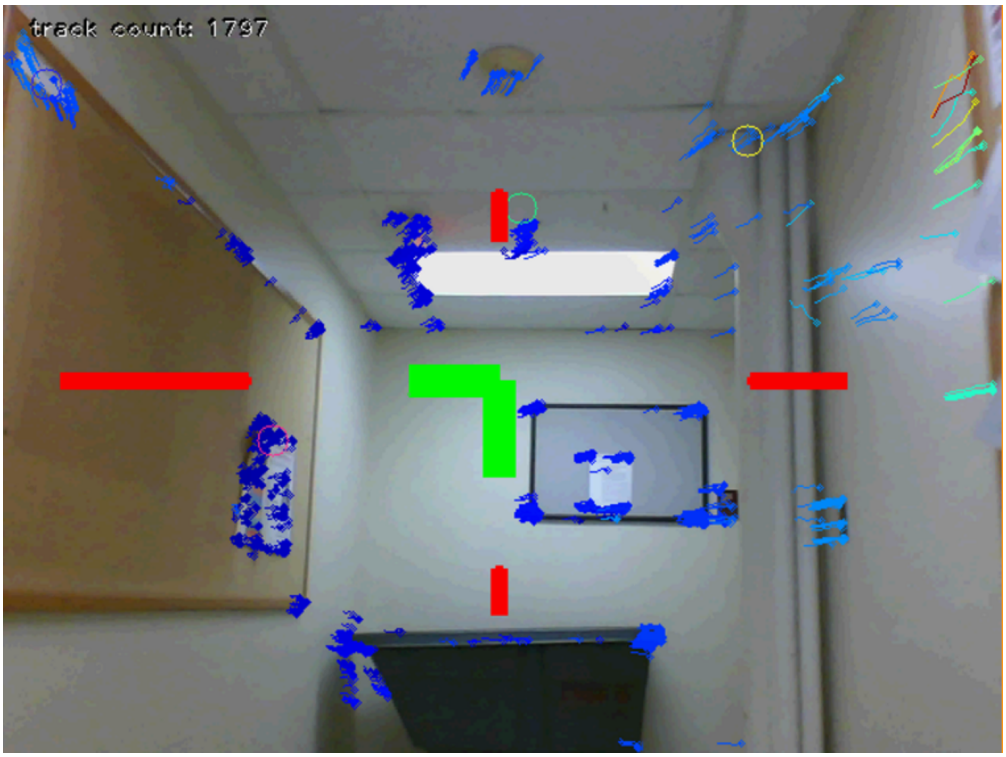

Figure 7: A video still from the image segmentation testing

Even though extensive testing could not be run, the results from the test appear to make sense. For the most part there are more points of interest on the ceiling than on the floor, so the control signal was telling the device to move in the -y (down) direction for most of the run. This could also pose a challenge if the environment has a monotone floor pattern where there are not any keypoints, as the control will simply keep telling the robot to move downwards and it will eventually hit the floor rather than balancing between the floor and ceiling. This control strategy may be better suited for environments with obstacles above and below in all directions (for example a search and rescue operation in a fallen building). It would also be interesting to only implement the vertical controls under certain cases, for example if there is an object approaching directly in the y-direction, then move to avoid it, otherwise ignore the small signals from above and below. This could be done by thresholding the values that trigger a control signal for the vertical case, or separating the x-y components of the flow and only responding to large y-direction flows.

6 Conclusions and Future Work

The long term goal is to continue this work as a thesis project in which controls will be developed for visual servoing, to allow for the quadrotor, or robot, to turn corners and navigate more densely populated environments. The current method only accounts for left and right balancing and needs to be modified to allow for turning corners. Methods for implementing this include attempting different image segmentation techniques, adding side facing cameras, and attempting to decouple rotation from translation in the optical flow vectors, which continues to be a struggle in the literature. One intuitive way to consider a corner in a hallway is that when the area is reached, the optical flow on one side will drop out, because of the space of the joining hallway. It may be possible to utilize this information to signal the quadrotor has reached a corner and should turn toward where the flow has dropped off. Additionally, it is of interest to potentially

implement adaptive control in the system by utilizing the history of saved optical flow data. This could allow for a more sophisticated system for collision avoidance. The original system by Seebacher also had issues with latency in communicating between the PC and the quadrotor. Because of this latency, the quadrotor could only fly at relatively low speeds. It is of interest to reduce this latency either by means of updating the platform or the software system used. The results of this paper are an in depth discussion of optical flow techniques, the idea of time to transit, and different feature detector and descriptor methods. It also includes a literature review of the current state of the art for optical flow and autonomous robot control, in addition to preliminary background research on understanding work that has been done so far in this area, with specific focus on understanding the code developed in [23]. Basic testing was also performed to compare real time computation times of different feature detector, descriptor, and matcher methods and combinations for use in optical flow computations. This research stands as the groundwork for thesis research in this area.

References

[1] A. Alahhi, R. Ortiz, and P. Vandergheynst. Freak: Fast retina keypoint. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 510–517, 2012.

[2] H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. *Computer Vision - EECV 2006*, pages 404–417, 2006.

[3] O. J. Bertrand, J. P. Lindenmann, and M. Egelhaaf. A bio-inspired collision avoidance model based on spatial information derived from motion detectors leads to common routes. *PLOS Computational Biology*, 11(11), 2015.

[4] A. Broid, J.-C. Zufferey, and D. Floreanno. A method for ego-motion estimation in micro-hovering platforms flying in very cluttered environments. *Autonomous Robots*, 40(5):789–803, 2016.

[5] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. *Computer Vision-ECCV 2010*, pages 778–792, 2010.

[6] G. Convertino. Region-based optical flow estimation technique for collision avoidance. *Intelligent Robots and Computer Vision XVI:Algorithms, Techniques, Active Vision and Materials Handeling*, 3208(1):118–125, 1997.

[7] A. Desai, D.-J. Lee, and D. Ventura. An efficient feature descriptor based on synthetic basis functions and uniqueness matching strategy. *Computer Vision and Image Understanding*, 142:37–49, 2016.

[8] C. Harris and M. Stephens. A combined corner and edge detector. *Alvey Vision Conference*, 15:147–152, 1988.

[9] B. Horn and B. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 2014.

[10] M. K. Kaiser and L. Mowafy. Optical specification of time to passage: Observers' sensitivity to global tau. *Journal of Experimental Psychology: Human Perception and Performance*, 19(5):1028–1040, 1993.

[11] B. L. T. Kanade. An iterative image registration technique with an application to stereo vision, 1981.

[12] F. Kendoul, I. Fantoni, and K. Nonami. Optic flow-based vision system for autonomous 3d localization and control of small aerial vehicles. *Robotics and Autonomous Systems*, 57:591–602, 2009.

[13] Z. Kong, K. Ozcimder, N. Fuller, A. Greco, D. Theriault, Z. Wu, T. Kunz, M. Betke, and J. Baillieul. Optical flow sensing and the inverse perception problem for flying bats, 2013.

[14] D. N. Lee. A theory of visual control of breaking based on information about time-to-collision. *Perception*, 5(4):437–459, 1976.

[15] S. Leutenegger, M. Chli, and R. Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. *2011 IEEE International Conference in Computer Vision*, pages 2548–2555, 2011.

[16] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[17] K. MacGuire, G. de Croon, C. de Wagter, B. Remes, K. Tuyls, and H. Kappen. Local histogram matching for efficient optical flow computation applied to velocity estimation on pocket drones. *2016 IEEE International Conference on Robotics and Automation*, pages 3255–3260, 2016.

[18] M. B. Milde, O. J. Bertrand, R. Benosman, M. Egelhaaf, and E. Chicca. Bioinspired event-driven collision avoidance algorithm based on optic flow. *2015 International Conference on Event-based Control, Communication, and Signal Processing*, pages 1–7, 2015.

[19] M. B. Rhudy, Y. Gu, H. Chao, and J. N. Gross. Unmanned aerial vehicle navigation using wide-field optical flow and inertial sensors. *Journal of Robotics*, 2015.

[20] E. Rosten, R. Porter, and T. Drummond. Faster and better: A machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):105–119, 2010.

[21] A. J. Sanchez-Garcia, H. V. Rios-Figueroa, A. Marin-Hernandez, and G. Contreras-Vega. Decision making for obstacle avoidance in autonomous mobile robots by time to contact and optical flow. *International Conference on Electronics, Communications and Computers*, pages 130–134, 2015.

[22] K. Sebesta and J. Baillieul. Animal-inspired agile flight using optical flow sensing, 2012.

[23] P. Seebacher. Motion control using optical flow of sparse image features, 2015.

[24] J. R. Serres and F. Ruffier. Biomimetic autopilot based on minimalistic motion vision for navigating along corridors comprising u-shaed and s-shaped turns. *Journal of Bionic Engineering*, 12(1):47–60, 2015.

[25] K. Watanabe, T. Kaegeyu, S. Maeyama, and I. Nagai. An obstacle avoidance method by combining image-based visual servoing and optical flow. *SICE Annual Conference*, pages 677–682, 2015.

[26] D.-W. Yoo, D.-Y. Won, and M.-J. Tahk. Optical flow based collision avoidance of multi-rotor uavs in urban environments. *International Journal of Aeronautical and Space Science*, 12(3):252–259, 2011.