

Bird and Bat Interaction Vision-Based Detection System for Wind Turbines

by
William Gage Maurer

A THESIS

submitted to
Oregon State University
University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Mechanical Engineering
(Honors Scholar)

Presented March 4, 2016
Commencement June 2016

AN ABSTRACT OF THE THESIS OF

William Gage Maurer for the degree of Honors Baccalaureate of Science in Mechanical Engineering presented on March 4, 2016. Title: Bird and Bat Interaction Vision-Based Detection System for Wind Turbines .

Abstract approved: _____

Roberto Albertani

Bird and bat collisions with wind turbine blades are an occurrence which are extremely variable in frequency. With the expansion of wind farms, determining the true quantity of collisions and the species involved is imperative for preventing ecological damage. Explored in this thesis is a blade mounted camera for wirelessly transmitting a video stream to provide an optimal viewing location for capturing avian and bat strikes. An early version of computer vision software for detecting avian flybys and collisions was developed, along with initial design and testing of a blade-tip tracking program. Object recognition using a cascading classifier, and a backup tracking system provides a potential method for determining bird presence and the likelihood of collision. The ability of the program to remove repeating false-positive instances and strengthen the detection system in the process, provides a strong platform for avian detection from a blade mounted camera. Hardware validation was conducted to ensure the selected components will function as needed. A 3D printed on-blade enclosure was designed as a housing for the camera, transmitter, and power supply.

Key Words: Bird, Bat, Detection, Tracking, Wind Turbine, Wind Farm

Corresponding e-mail address: maurerw@oregonstate.edu

©Copyright by William Gage Maurer
March 4, 2016
All Rights Reserved

Bird and Bat Interaction Vision-Based Detection System for Wind Turbines

by
William Gage Maurer

A THESIS

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Mechanical Engineering
(Honors Scholar)

Presented March 4, 2016
Commencement June 2016

Honors Baccalaureate of Science in Mechanical Engineering project of William Maurer
presented on March 4, 2016.

APPROVED:

Roberto Albertani, Mentor, representing MIME

Nancy Squires, Committee Member, representing MIME

Sinisa Todorovic, Committee Member, representing EECS

Toni Doolen, Dean, University Honors College

I understand that my project will become part of the permanent collection of Oregon
State University, University Honors College. My signature below authorizes release of
my project to any reader upon request.

William Gage Maurer, Author

Acknowledgments

Thank you to my mentor, Dr. Roberto Albertani, who has provided me with meaningful research opportunities. Your guidance has helped me grow both as a researcher and engineer. Every aspect of working in the Applied Mechanics and Composites Technology Laboratory has been a fantastic experience. I would also like to thank my committee members Dr. Nancy Squires and Dr. Sinisa Todorovic; I truly appreciate your time and input on this project.

Thank you to my fellow lab members, who have offered their guidance and support through the entire research process. Thank you to my friends and family, who have been patient and supportive. Your encouragement has been immense in completing this project.

I must also thank my favorite local coffee source, Interzone, for providing the innumerable quantity of lattes required for this milestone of my undergraduate degree. Finally I must show my gratitude for MathWorks, whose products have been an integral component of my scholastic career.

Contents

Introduction.....	1
Background and Literature Review	4
Current Non-Automated Methods	4
Automated Monitoring for Birds and Bats in Wind Farms	6
Avian and Bat Vision-Based Detection and Tracking Methods	8
Blade Deflection	10
Proposed Solution and System Integration	12
Hardware and Validation	15
Camera Hardware	15
Transmitter and Receiver Systems.....	20
System Power	22
Hardware Validation.....	23
Camera Casing.....	26
Software Development: Avian Detection and Tracking	30
Overview.....	30
Overall Structure.....	32
Detection Options	35
Optical Flow.....	35
Cascade Object Detection.....	38
Blade Face Thresholding	41
Tracking.....	43
Primary Tracking	43
Secondary Tracking	44
Trajectory Classification.....	46
Supporting Structure	47
Data Storage.....	47
Filtering.....	48
Collision Sensing	49
Data Presentation	50
Detection Improvement	51
GUI for Detection and Primary Tracking Tests.....	51
Operational Testing.....	53

Trial 1.....	54
Trial 2.....	56
Trial 3.....	57
Discussion.....	58
Speed Testing.....	59
Conclusions.....	60
Blade Tracking.....	62
Introduction.....	62
Program Structure	63
Testing	64
Results.....	65
Discussion	67
Future Developments	69
Hardware.....	69
Sensor System Integration	69
Camera Casing.....	70
Avian Program.....	71
Blade Track.....	73
Conclusion	74
Work Cited.....	75
Appendix A: Turbine Size Estimation	83
MATLAB Code	83
Output	85
Source for Wind Turbine Data.....	87
Appendix B: Pixel Size Calculation Program.....	88
MATLAB Code	88
Output	90
Appendix C: Camera Case Dimension	93
Appendix D: Avian Program Logic Diagram.....	96
Appendix E: Avian Interaction Program	99
Primary Code	99
Satellite Functions (Supporting Architecture)	110
signalCheck.....	110
boundaries	111
blobCheck	111

bboxCombine	112
KLTpoints_revised	114
evalKalmanTracks	116
FinishTracking	116
reportAnalysis2	118
RepeatCheck	121
RevisedCache	124
zoneCheck	124
NegativeSave	125
Appendix F: Collision Likelihood	127
Appendix G: GUI Code	129
Primary Code	129
GUI Figure	137
Appendix H: Blade Tracking Code.....	138
Primary Code	138
Satellite Functions.....	140
pointFinder	140
pointTracker	140
Appendix I: Blade Tracking Results.....	142

Figures

Figure 3-1 Diagram of the FOV on a wind turbine.....	13
Figure 4-1 Schematic Representation of Camera FOV.....	17
Figure 4-2 Pixel area versus position along length of turbine blade.....	19
Figure 4-3 A representation of pixilation.....	20
Figure 4-4 Micro-camera circuit design and breadboard implementation.....	23
Figure 4-5 Camera rotation rig	24
Figure 4-6 The effects of motion and lighting on image quality	24
Figure 4-7 Deinterlacing frames containing significant camera motion.....	25
Figure 4-8 Target area calculations.....	26
Figure 4-9 Camera casing expanded view	28
Figure 4-10 3D printed camera casing.....	28
Figure 4-11 Warping of the camera casing base plate	29
Figure 5-1 Frame of Reference	32
Figure 5-2 General program architecture.....	33
Figure 5-3 Opening User Prompt.....	34
Figure 5-4 Example of input signal simulating vibrational sensors detecting a collision.....	34
Figure 5-5 Lucas-Kanade Optical Flow Method	37
Figure 5-6 A comparison of detection accuracy when rotating the input image.	40
Figure 5-7 Examples of cascade object detection applied to input bird images	41
Figure 5-8 Selection of the blade ROI	42
Figure 5-9 Thresholding and its results	42
Figure 5-10 Large displacement tracking via point tracking	46
Figure 5-11 Example of target motion and corresponding classification	47
Figure 5-12 Spatiotemporal filter principle	48
Figure 5-13 SSIM comparison between detections	49
Figure 5-14 Example false positive result in a window presenting the detection to the user.	50
Figure 5-15 Examples of repeating or user input FP detections	51
Figure 5-16 Example of successful tracking in the GUI.....	52
Figure 5-17 Example of unsuccessful detection and tracking within the GUI.	53
Figure 5-18 Different detected objects incorrectly tracked by the primary tracking system.....	55
Figure 5-19 Results seen by the user after program operation.....	56
Figure 5-20 False positive quantities, including total and filtered.....	58

Figure 5-21 Largest processing time for the overall program.....	60
Figure 6-1 Point tracking on balsa wood beam.....	64
Figure 6-2 Tip displacement in millimeters versus frame number	65
Figure 6-3 Measured Tip Displacement versus True Tip Displacement	67
Figure 6-4 Checkerboard pattern	68

1. Introduction

The first quarter of 2015 has shown a large increase in the growth of wind energy projects, with 13,600 megawatts of capacity under construction [1]. The implications of this near-record growth go beyond the spread of a renewable energy source. Effects of wind farms, or large groupings of wind turbines, on local and migrating birds and bats is not well understood. Due to the complexity and varying fragility of ecosystems across the globe, each new wind farm poses a different level of risk for damaging protected avian and bat populations.

Current methods for examining the collision risk a wind farm poses are generally costly, cumbersome, or inaccurate. While Chapter 2 goes into the detail of these methods, they may be summarized as follows: carcass retrieval and mortality estimation [2], collision risk modelling [3], and observation [4]. Some budding techniques automate this monitoring process, by using various sensors to continuously check for bird or bat interactions and collisions [5-8]. Automated wind farm monitoring will play an important role in understanding how oceanic wind farms affect seabirds, as the classic technique of carcass retrieval is made impossible by the setting. As a solution to this need for automation in detecting avian and bat collisions with wind turbines, Oregon State University and the University of Washington have proposed a multi-sensor solution [9].

This thesis looks to design and investigate a blade-mounted vision-based camera system for the monitoring of bird and bat interactions and collisions with the wind turbine blade. The proposed mounting location for the camera is on the root of the blade, positioned in such a way that the field of view (FOV) captures the entirety of a single face of the blade. By filming a blade face during standard wind turbine operation, the opportunity arises for monitoring both avian and bat collisions, as well as flap-wise tip deflection. A further description of integration with the sensor array may be found in Chapter 3.

Deliverables for this thesis include early iterations of MATLAB code for the avian detection/tracking and blade tracking software, and analysis and proof of concept testing for both. Also investigated, is the feasibility of selected hardware, case design for the on-blade components, and future additions for this unit.

There are numerous benefits to having a blade-mounted camera. Identification of species that collide with the wind turbine is perhaps the most important. The blade-mounted camera position provides an optimal view of the bird or bat flying near or colliding with the turbine blade, and could yield information regarding the nature of any impacts which might occur. Research has been conducted for determining ways blade deflection can be monitored during operation [10-12], as discussed in Chapter 2. Extreme situations may cause deflection beyond what the manufacturer has specified, and having awareness of such occurrences can allow operators to examine the blade early on.

The primary components of the system include a wireless micro camera, receiver, power source, and transmitter; the components are reviewed in detail in Chapter 4. The software used for video analysis will be programmed in MATLAB, and will be developed to a functional level, to demonstrate the capabilities of the selected computer vision techniques.

The development of software is covered in chapters 5 and 6, split into avian monitoring, and blade tip tracking respectively. Large wind turbines tend to rotate between 15-20 revolutions per minute, or 90-120 degrees per second [13]. The challenges associated with mounting a camera on a platform rotating at this rate includes a highly dynamic background, and extreme changes in lighting conditions- solutions presented for avian and bat interaction and collision tracking were created with these factors in mind.

Finally, chapter 7 outlines future work and considerations for applying this sub-system to real-world applications. Considerations such as power sources, on-blade mounting, and component

longevity are discussed. Also highlighted, is the further testing and design needed to bring this sub-system to a fully operational level.

2. Background and Literature Review

Understanding the risk that wind turbines pose to local and migrating bird and bat populations is a complex task with high variance between locations. There is both public and private interest in procuring accurate data pertaining to the mortality rates of birds and bats caused by wind farms. Some wind energy companies must work to remain within the bounds of their incidental take permits (ITPs) which allow for the accidental fatalities of endangered or otherwise protected species [14]. Public opinion can have a large effect on the expansion of wind energy, therefore it is imperative that accurate collision data is made available so that the impact of wind farms on ecosystems may be properly evaluated.

The values obtained from studies examining bird mortality rates at wind farms are highly site dependent. Data collected in a study by J. Everaert showed yearly avian turbine collisions in three wind turbine locations in Belgium ranging from around 0 to 125 per turbine, and for 2002, averages ranging from 18 to 35 [15]. Generally less studied than the effects of wind farms on avian populations, is the impact of wind farms on endangered and at-risk bat species. Studies have shown that the weather and migration patterns both affect bat mortality rate [16]. A concern that has also been raised with bats is barotrauma, where the pressure change from wind turbine blades can cause organ damage, however this aspect has not been well documented [16].

Current Non-Automated Methods

A common method of determining the rate of avian and bat fatalities on wind farms is the periodic collection of carcasses. To address some of the primary sources of inaccuracy for mortality estimation and provide a better comparison among estimates, K. S. Smallwood used past report data to analyze and model several factors including the mean time to removal and proportional remaining carcasses post-search [2]. Errors with this method stem primarily from scavenger removal, search accuracy, and mortality estimation equations [2]. Mortality estimation is an attempt

at removing the bias of ground search methods utilizing correction factors in a mathematical model [2], an example may be seen below:

$$M_A = \frac{\bar{c}}{\left(\frac{\bar{t} \times p}{I}\right) \left(\frac{e^{I/\bar{t}} - 1}{e^{I/\bar{t}} - 1 + p}\right)} \quad \text{Equation 2-1}$$

M_A Adjusted mortality rate

\bar{c} Average number of carcasses observed per year

\bar{t} Mean number of days until carcass removal

p Searcher efficiency rate

I Search interval in days

This equation is a revision, the predecessor of which tended to provide mortality rates that were approximately 23% too low [2]. With the large number of factors which affect the accuracy of mortality estimation, biases tend to form favoring either high or low estimates [2]. These biases can stem from variance in searcher efficiency, which is dependent on factors such as ground vegetation; a tilled field will allow for easier searching in comparison to tall grass [2]. Scavenger removal of carcasses is also variant, and may be dependent on time of year and the species of the carcass; during autumn an increase in removal may occur as scavengers work to store extra fat for winter [2].

Collision risk modelling is a predictive version of mortality estimation. Geometric factors such as wind farm width, airspace volume, rotor size, and bird wingspan are combined to determine the likelihood of a bird species coming in contact with a wind turbine [3]. Assumptions must be made for this type of risk assessment, and one of the most impactful is the incorporation or exclusion of avoidance behavior [3]. A lack of avoidance behavior models bird flight within the wind farm as

though the bird does not detect any presence of turbine structures [3]. To correctly model avoidance behavior, observation is required, and such information is species and site specific.

Observation provides an alternative to carcass searches and estimation. This alternative can provide more intimate details regarding avian and bat behavior in the presence of operational, non-operational, and pre-construction wind farms. For example, Cindy L. Hull and Stuart C. Muir performed a study which spanned across 875 days to examine the avoidance behavior of two eagle species in Australian wind farms [4]. The detail learned from this type of monitoring is much greater than the aforementioned estimation method. For instance, it was found that the Tasmanian wedge-tailed eagle and white-bellied sea-eagle prefer to fly between 1.5 and 3 rotor diameters through wind farms [4]. This type of information can lead to improved mortality estimation, however it is costly in time and does not guarantee accuracy between differing species and wind farm locations.

Automated Monitoring for Birds and Bats in Wind Farms

Non-automated monitoring methods generally are affected by uncertainty (mortality estimation) or personnel hours required (observation). Automation of the observation and monitoring process for wind farms provides an alternative which allows for both greater accuracy in collision statistics, and lowered time and cost. Wind turbines on off-shore locations create the issue of difficult to impossible carcass retrieval, and poor location for long term observation.

There are numerous methods for automating the monitoring process on wind farms. There are several companies which have commercialized approaches to this issue, some examples including the DeTect Merlin Avian Radar System [5], and DTBird [6]. Other systems include WT-Bird which was developed by the Energy research Centre of the Netherlands (ECN) [7], and a system being investigated by the California Energy Commission [8]. Each of these solutions will be summarized in order to provide insight to current methods.

The Merlin Avian Radar System by DeTect is based off of technology developed for use by the United States Air Force and NASA, in order to increase the safety of operations by ensuring minimal avian activity near mission sites [5]. The equipment necessary for this radar technology can be placed on-site and controlled remotely using a variety of techniques [5]. The number of birds passing through the radar swept region may be used in order to improve collision risk assessment, and provide an early warning system for operators for the approaching of flocks [5].

DTBird is a system of hardware and software dedicated to detecting birds in wind turbine airspace, and optionally taking preventative measures to avoid collision [6]. Four to eight high definition cameras create a 360 degree field of view for the detection system, which is setup to include the rotor sweep [6]. Preventative measures includes two possibilities, the Collision Avoidance Module, and the Stop Control Module [6]. The Collision Avoidance Module emits warning tones when birds enter high risk flight paths near wind turbines, with the purpose of dissuading collision [6]. The Stop Control Module brings the wind turbine to a complete stop between 20-40 seconds after the system has been triggered, to allow safe passage for the bird setting off the module [6]. Finally a secondary monitoring system, referred to as the Collision Control Module, provides vision and auditory based monitoring around the rotor sweep to record any collisions which may occur [6].

The Energy Research Centre of the Netherlands has developed a wind turbine detection system referred to as WT-Bird, which is comprised of video cameras and contact microphones [7]. Contact microphones placed on the inside of the tower continuously check for vibrations which do not follow the usual patterns caused by typical operations [7]. Cameras are placed in multiple locations: one looking up from the mid-section of the tower towards the rotor sweep, and another looking from the nacelle towards the rotor sweep [7]. Abnormal sounds cause analysis of video recorded around a given time-span of the instance, which allows for bird recognition during interactions or impacts with the wind turbine [7].

The Public Interest Energy Research Program, or PIER, is an organization under the management of the California Energy Commission. In 2007, PIER released a report outlining an automated wind farm bird and bat collision detection system [8]. The purpose of this particular study was to investigate the feasibility of an array of sensors (to be retrofitted on current wind turbines or incorporated into new turbines), including accelerometers, fiber-optic sensors, non-contact sensors, radar and infrared, and acoustic emission sensors [8]. In terms of ease of implementation and overall cost, acoustic emission sensors were found to be the optimal choice, since there is no need for on blade installment of the microphones, and microphone cost tends to be low [8]. The use of radar or vision based techniques was found to be less than ideal for collision detection when utilized on their own, due to the difficulty of deciphering between a bird or bat fly-by versus collision [8].

Avian Vision-Based Detection and Tracking Methods

Vision based avian detection systems are created for a wide range of applications, including ecological purposes and airport safety. There is more computer vision research pertaining to ornithological applications, however many of the techniques may translate well to bats.

In Hierarchical Incorporation of Shape and Shape Dynamics for Flying Bird Detection, Z. Jun, X. Qunyu, C. Xianbin, Y. Pingkun, and L. Xuelong present the design and testing of a novel bird detection system [17]. Detection is two-fold, first performing shape based confirmation of the bird, followed by an analysis of bird movement [17]. Bird flight is categorized in four states, which comprise the entire flapping process [17]. The hierarchical structure maintains computational efficiency by removing obvious non-bird targets via shape analysis before moving on to more process-heavy dynamic shape analysis [17]. This method demonstrated a high detection rate and low false positive quantities, including when used with videos that incorporate noise and low contrast [17].

Webcams are a technology that can be found in a myriad of locations. By adapting computer vision techniques to be used with these affordable cameras, these devices can be used to collect data pertaining to bird's migration patterns [18]. W. W. Verstraeten et al performed experiments to investigate the validity of using webcams for the detection and three dimensional tracking of birds (for determining flight velocity and altitude) [18]. One of the experiments conducted involved the use of a pendulum to test the detectability of varying sizes and contrasts between targets and the background [18]. The results of this experiment showed that using background subtraction, as the velocity of the target increased, it became harder for lighter targets to be detected (against a white background) [18]. This relationship is key for more than background subtraction methods- as the contrast between a target and the surrounding environment is lowered, many detection methods may become faulty [18]. Accounting for lens distortion, managing multiple moving objects, calculating distance, and analyzing error were also performed in this report to develop a bird tracking system [18].

In “Automatic Bird Species Detection from Crowd Sourced Videos”, W. Li and D. Song designed two algorithms, one to extract avian inter-wing tip distance across time, and another to determine the wingbeat frequency and estimate the species of the avian target [19]. An important feature of the work outlined in this paper is the use of optical flow to decipher between background and foreground [19]; a large benefit of this method over background subtraction is the allowance of camera motion from crowd sourced videos [19]. Fast Fourier Transform applied to the inter wing tip distance time series provided the wingbeat frequency, which is used to estimate the species [19]. Experimentation showed successful extraction of inter-wing tip distance and wing beat frequency [19]. Robustness to error caused by foreground extraction, species prediction accuracy, and behavior when the most current inter-wing tip distance is not available was also tested [19].

A wind turbine blade mounted camera brings about challenges that are uncommon to many bird detection and tracking computer vision applications. The rotational velocity of the wind turbine

blade can create large displacements between frames, which depending on the camera framerate and other factors, may cause certain methods to not be a viable option. A limited number of frames containing the target leaves little room for analyzing flight.

Blade Deflection

Mounting a camera on the root of a wind turbine blade has the potential for monitoring blade deflection. From a recent report, the annual blade failures were estimated to be around 0.54% of the current number of blades in operation (approximately 3,800 out of 700,000) [20]. There are a number of sources which could contribute to the failure of a blade, such as operational failures which lead to loading which exceeds the rated amount [21]. One way of monitoring the stresses within the blade is to examine the deflection of the tip (the point of greatest deflection). To make use of the on-blade camera for measuring blade tip deflection, the system serves an additional purpose and bolsters its appeal for mass wind farm use.

The National Renewable Energy Laboratory in Golden, Colorado developed and tested an optical blade position tracking system, with the purpose of determining a low cost method for monitoring the loads seen by the wind turbine blade [10]. A standard webcam with infrared LED lighting was mounted at the blade root with a light filter on the lens, while reflective strips were placed along the wind turbine blade, allowing the lighting from the webcam to bounce back to the lens [10]. A bend test of a wind turbine blade provided a comparison between measurements using the optical method and string potentiometers [10]. Testing provided a low mean error between the measurement types, and demonstrated the viability of the system [10].

The work of X. Fu, L. He, and H. Qiu shows promise for the utilization of MEMS gyroscopic sensors in combination with an artificial neural network for measuring blade deflection on wind turbines [11]. For testing, each blade on a 100m diameter wind turbine was outfitted with a MEMS gyroscopic sensor 20m from the blade root [11]. Data collected from these sensors during operation

was compared with data obtained from a laser measurement device attached at the tower (making measurements for each blade once per rotation) [11]. The accuracy of the MEMS sensors came out to $\pm 0.4\text{m}$, when comparing the data to that of the laser measurement device, the accuracy of which is approximately 1 centimeter [11].

P. Giri, and J. R. Lee explored a real-time deflection monitoring system for use with wind turbines [12]. The proposed system projects and receives the reflection of a laser emanating from the tower of the turbine [12]. With this laser displacement sensor (LDS), experiments including bolt loosening, nacelle tilting, and blade mass removal were performed to examine the corresponding blade deflection [12]. The experiments suggested that this method is an effective way to both measure and infer the cause of blade tip displacement [12].

The use of an on-blade camera for blade tip deflection measurement provides a constant monitoring scheme for providing real-time feedback. Preventing turbine failure is one of the foremost reasons for implementing this software, however it also has the added value of examining blade deflection across large periods of time. Understanding the deflection of large composite wind turbine blades will provide key insight to the cyclical loading seen during operation for a range of conditions. This information may ultimately reduce the number of wind turbine blade failures, thus cutting costs of blade replacement, and most importantly increasing the safety of the system.

3. Proposed Solution and System Integration

The large size, and dynamic state of operational wind turbines makes them a particularly difficult structure to monitor. A solution posed by Oregon State University, and the University of Washington, aims to use an array of sensors to continuously monitor wind turbines for avian and bat interactions and collisions [9]. The application of the sensor array is designed for use with off-shore wind turbines, however the system has the potential to be used with land based turbines without modification [9]. The array is comprised of vibrational sensors, vision based cameras, and IR cameras, and bioacoustic microphones [9].

The sensor array utilizes wireless accelerometers and contact microphones, which may be placed on the tower, nacelle, or blades of the turbine [9]. Using batteries and wireless transmitters, the vibrational sensors require minimal support structure to operate [9]. The vibrations caused by an impact on a wind turbine will vary based on the size and speed of the object colliding, as well as the location of the sensors on the wind turbine; in order to decipher between operational and collision caused vibrations, wavelet analysis is performed on the digitized signals [9]. An important aspect of this method is its ability to be used in real-time. When a collision does occur, the vibrational sensors can act as a trigger mechanism for vision based sensors, and for data storage purposes [9].

Vision and infrared based cameras were explored for the purpose of taxonomic classification, detection of flybys and collisions, and capturing bird or bat interactions with the wind turbine for later review [9]. There were five camera locations suggested including [9]:

- Nacelle with a FOV intersecting the rotor plane
- Nacelle with a FOV above the rotor plane
- Tower with the FOV facing an upwards direction

- Adjacent tower with the FOV intersecting the rotor plane
- Root of the blade with a FOV that covers most of one face of the blade

The latter option, with the camera mounted on the root of the blade is what this thesis will develop. The reasoning behind this mounting location lies mostly in its ability to capture the critical footage of impacts, which provides the most solidifying evidence of bird or bad collisions with the wind turbine. Another attractive feature of the on-blade camera location is its ability to track the blade tip position, which when used with real-time processing, allows for an alert system when severe deflection occurs.

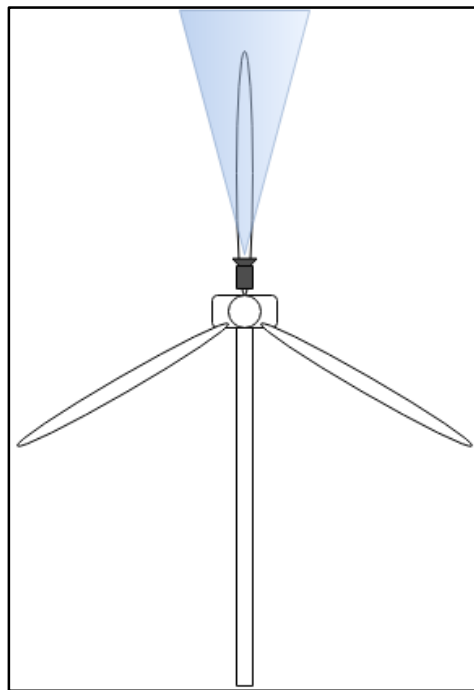


Figure 3-1 *Diagram of the FOV on a wind turbine.*

All data will be relayed to a central computing unit inside the nacelle [9]. The purpose of this computer is to apply algorithms and computational techniques to the raw incoming data from the transducers and cameras [9]. A central computing unit allows for easy data storage and analysis at a single location, which can then be relayed to external locations for evaluation [9]. A single camera

node (containing both visual and IR cameras) can output data at a rate near 1 Gbps [9]; applying vision-based algorithms to such large quantities of data is a resource intensive process, and storage for the videos becomes near-impossible with one hour of footage requiring around 3.6 Tb. A possible solution for this problem is the use of ring buffering, where data is stored before and after an event, creating a temporal “buffering” zone [9]. Infrared cameras are generally associated with simpler computer vision techniques, and may be able to run in real-time as an additional triggering mechanism for the ring buffering architecture [9]. Vision based data can be stored around the time of an event detected by vibrational sensors, and analyzed sub-real-time [9].

The blade mounted camera will provide a constant transmission of video data to the central computing unit. All video processing will be performed in the central computer, limiting the amount of components and stored power needed within the camera housing. There are two possible methods of processing for this on-blade camera, including sensor triggered ring buffering, or real-time vision based triggering for event storage. The benefit of real-time processing is the ability of the on-blade camera to check for flybys and impacts without input from the other sensors.

Automated taxonomic classification, which is a future addition for this system, will be performed using vision-based data, and recordings from bioacoustic microphones [9]. As a possible future addition, the use of two cameras at any of the camera mounting location can provide binocular vision, which allows for determination of the size of the bird or bat. Size is generally measured by wingspan or body length, and is a key factor in determining species. The importance of taxonomic classification lies in its ability to further automate the turbine monitoring process, and alert wind farm operators to the presence of collisions or interactions between turbines and endangered or at risk species.

4. Hardware and Validation

In this chapter, hardware is selected and validated. Housing for the on-blade camera is designed and manufactured.

Camera Hardware

The camera selected needs to have a number of requirements for on-blade installation. Automatic adjustability of shutter speed for brightness changes, low power consumption, small profile, and sufficient resolution are desirable. The Sentech STC-N632 is an NTSC board type CCD micro-camera which provided the needed for this application (see table 4-1). Additionally, this camera supports an assortment lenses for varying focal length and thus FOV. This camera is designed to endure a range of humidity, temperatures, and a high level of vibrations- important factors for wind turbine applications [22].

Table 4-1 *Sentech STC-N632 Camera Specifications.* Obtained from Sentech mfg. specifications [22].

Sentech STC-N632 CCD Camera Specifications		
Electronic Specifications	Imager	1/3" Interline NTSC CCD: ICX638AK (Sony)
	Active Picture Elements	768 (H) x 494 (V)
	Chip Size	5.59 (H) x 4.68 (V) mm
	Cell Size	6.35 (H) x 7.4 (V) μm
	Scanning System	2:1 Interlace
	Vertical Frequency (Frame Rate)	59.94 Hz
	Horizontal Frequency	15.734 kHz
	Resolution	480 TV lines
	S/N Ratio	More than 48 dB (Gain 0 dB)
	Minimum Scene Illumination	0.53 lx at F1.2

	Sync. System	Internal
	Video Output	1.0Vp-p with 75Ω
	Shutter Speed	Electronic Iris/Fixed Shutter (Software Selectable)
	Gain	AGC/Fixed Gain (Software Selectable)
	Gamma	1/0.45 (Software Selectable)
	Input Voltage	DC +7V to +13V
	Consumption	Less than 1.0W
Mechanical Specifications	Dimensions	26 (W) x 26 (H) x ** (D) mm
	Optical Filter	IR cut filter
	Weight	12g (without lens)
Environmental Conditions	Operational Temperature/Humidity	-10 to 45°C; 0 to 85% (relative humidity with no condensation)
	Storage Temperature/Humidity	-30 to 65°C; 0 to 90% (relative humidity with no condensation)
	RoHS	RoHS Compliant

NTSC cameras, with a frequency of 59.94 Hz, display around 60 fields per second [23]. Fields are horizontal lines that alternate between blank space and information from the frame, in such a way that two consecutive fields combine to create a whole image [23]. Due to this alternating characteristic, digitized NTSC footage translates to 30 frames per second (FPS) [23]. The advantage of interlacing is a reduction in bandwidth [24], an important factor for wirelessly transmitted video.

A 25 mm focal length M12 lens was selected to apply to the CCD camera. Before making this decision, the FOV needed to be calculated for the lens and camera combination, to verify that it would be sufficient to capture most of the blade face. The FOV should encompass as much of the blade face and its surroundings as possible, while not being so large as to severely degrade the resolution of targets at a distance. The FOV is comprised of two angles, which will be referred to as the horizontal and vertical FOV. The calculation for these angles requires values for the focal length and sensor dimensions [25]. This FOV calculation may be seen below in equation 4-1 and figure 4-1.

$$HorizontalFOV = 2 \times \tan^{-1}\left(\frac{\frac{1}{2} \times SensorWidth}{FocalLength}\right) \quad \text{Equation 4-1}$$

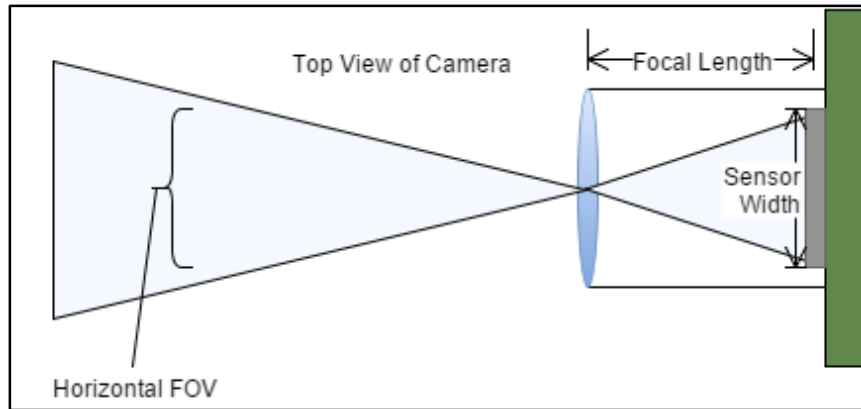


Figure 4-1 Schematic Representation of Camera FOV.

The vertical FOV is calculated in a similar method. The sensor width is determined using the pixel size, along with the number of pixels in the corresponding direction. Pixel size is interchangeable with cell size, so 6.35 (H) x 7.4 (V) μm is used. The cell is the light sensing diode which is used in digital cameras. The horizontal and vertical FOV were found to be 11.14° and 8.36° respectively.

For taxonomic identification purposes, the bird or bat being observed must have a sufficient number of pixels so that critical features may be visible. In order to ensure the selected hardware provides enough resolution, an avian target and observation distance were selected, and the pixel density for

the target was determined. The chosen target was a marbled murrelet, a smaller endangered oceanic bird species [9].

It is important to consider the largest possible blade length to obtain the most conservative pixel density estimate. General Electric is looking to construct a 10 MW turbine testing facility, indicating that in the near future, offshore wind turbines may have even larger blade lengths than those currently used [26]. The blade length of the GE 10 MW turbine is unknown, so an estimation was created based on current production turbines. The data and MATLAB code for this estimation may be found in Appendix A.

The estimate of the GE 10 MW blade length provided a turbine blade length of 98.2 m. This blade length was used as the observation distance for target pixel density calculations. The marbled murrelet was modelled as a 23 by 23 cm box for the sake of simplicity. A second MATLAB script was written to calculate the pixel density of the target at a specified distance, and provide visuals of the pixel size across the turbine blade. Figure 4-2 shows the outputs of the program, and supporting code can be found in Appendix B. The program was designed to allow for vertical camera tilt, as adjustments to this angle may ultimately be needed for optical FOV placement. For the purposes of this initial analysis, the direction of projection (DOP) for the camera was offset from the turbine blade face by 4° , and the camera was assumed to be at the root of the blade. Calculations for pixel size were made by rotating the DOP, while holding the view plane at a specified distance, and normal to the length axis of the blade.

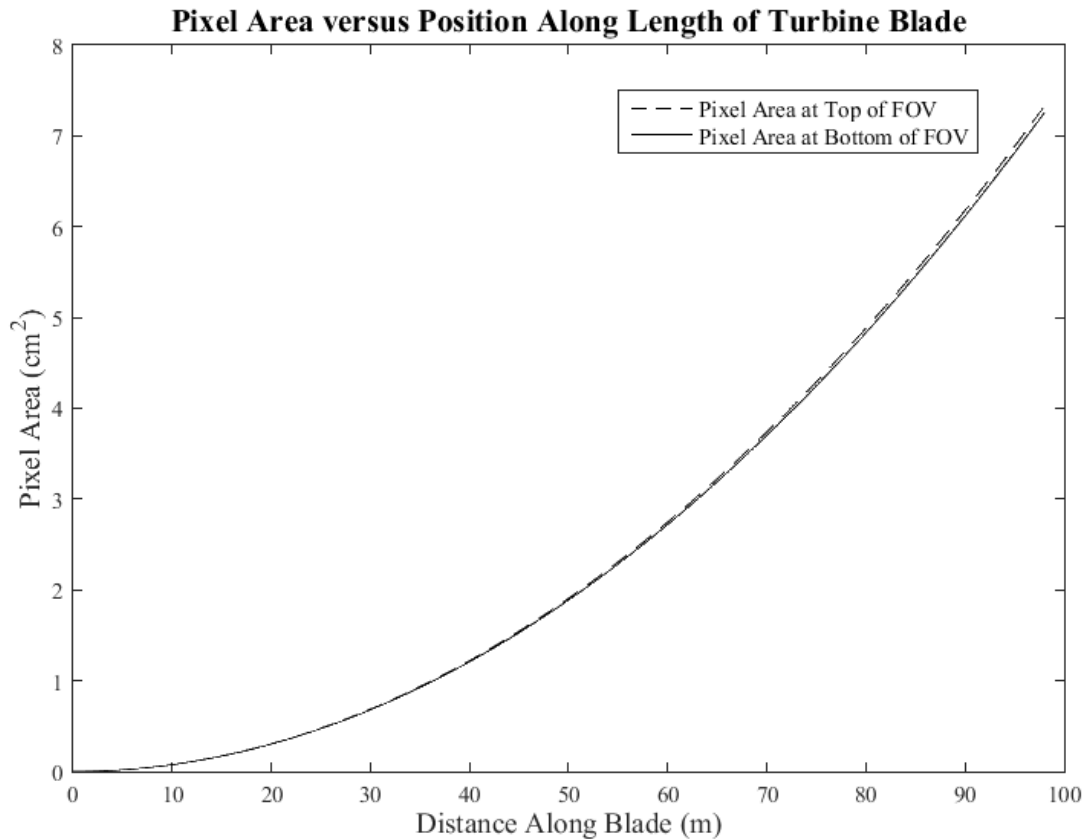


Figure 4-2 Pixel area versus position along length of turbine blade.

The estimated number of pixels in the 23 by 23 cm region of the marbelled murrelet is 73. As demonstrated in figure 4-3, the pixilation is severe enough to make identification difficult based on discerning features, however the general shape and coloration are still present. A comparison of pixel and target areas may be found in Appendix B. From this analysis, the Sentech STC-N632 CCD camera and 25mm focal length lens should be sufficient for this on-blade system.



Figure 4-3 A representation of pixilation, similar to a 23 by 23cm target at 100m from the Sentech STC-N632 camera. The image on the right approximates the pixilation estimated by the pixel size calculations.

Transmitter and Receiver Systems

A wireless transmitter will stream video from the micro-camera to the central computer. The transmitter will be connected with the camera and mounted on-blade, while the receiver will either be mounted in the nacelle with the central computer, or in a line of sight (LOS) location from the transmitter. The transmitter need a small profile and low power consumption. The SDX-26 2.4 GHz audio/video transmitter, whose specifications from [27] may be found in table 4-2, provides the qualities needed for this application.

Table 4-2 *SDX-26 2.4 GHz audio/video transmitter specifications.* Obtained from RF-Links specifications [27].

SDX-26 2.4 GHz Audio/Video Transmitter Specifications	
Smallest Size	0.5 x 0.5 x 0.12 in
Voltage Range	4.5 – 6.5 V
RF Power	25 mW/ 5.5 V
Current Consumption	60 mA/ 9 V
Picture quality	Broadcast
TV Systems	NTSC, PAL, or SECAM

Video Signal	1 V, 75 Ω
Frequency	2300 – 2500 MHz (Single Channel)
Channels	4 Channels

To communicate with this transmitter, the multi-channel VRX-24L Audio/Video Receiver was chosen. The size and power consumption of the receiver are less important than those of the camera and transmitter, due to its mounting location. Strong reception and low cost are the foremost reasons for selecting this receiver. Table 4-3 outlines the specifications from [28] of the VRX-24L.

Table 4-3 *VRX-24L Audio/Video Receiver Specifications.* Table Obtained from RF-Links specifications page for VRX-24L receiver [28].

VRX-24L Audio/Video Receiver Specifications	
Operating Frequencies	2300-2500 MHz
Channel	1-8
DC Voltage	9-12 V
RF Power	N/A
Minimum Required Voltage	9 V
Battery Power	12 V/ 300 mA
Video Distortion	3%
Sensitivity	-92 dBm
Video Format	PAL, NTSC
Current Consumption	280 mA/ 9 V
Antenna	Recommended High Gain, Omni Directional Ant.
Antenna Connector	SMA
Impedance	50 Ω
Video Output Connector	RCA F
Video Impedance	75 Ω

Two Audio Outputs	300 mV per channel
Carrier Frequencies for Audio Channels	6 MHz and 6.5 MHz
Temperature Range	-25 to 65° C
Dimensions	2.5 x 5.7 x 0.8 in
Weight	200 grams
Demodulation	WFM

System Power

To power the micro-camera and wireless transmitter, a single 120 volt AC to 12 volt DC wall power supply was initially used. When mobility was required during testing or otherwise, a 3 cell in-series lithium polymer battery pack was used, providing 12 volts at nominal.

While the camera operates at the voltages provided by the wall power supply and battery, the transmitter needs a lower voltage. The solution was to wire the camera, transmitter and resistor, and power source in parallel. The current consumption of the transmitter is known from the manufacturer specifications (table 4-2), and the voltage drop before the transmitter needed to be between 5.5 and 7.5 volts. For testing and early design, a simple resistor (100 Ω) in series was utilized to drop the voltage. The current consumption of the transmitter is relatively constant, so heat dissipation is the primary concern of using this voltage drop method. Figure 4-4 shows the wiring diagram for the camera system and breadboard implementation used to verify the circuit design.

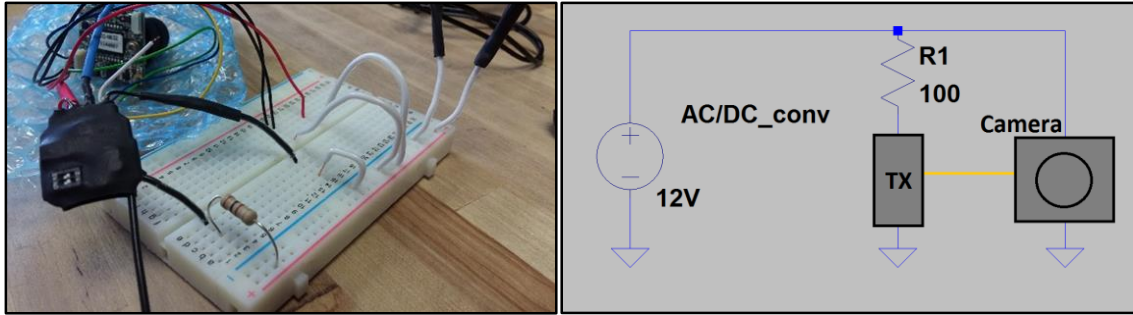


Figure 4-4 *Micro-camera circuit design and breadboard implementation.* The yellow wire represents the video signal distributed between the camera and transmitter.

Power distribution for the final system, discussed in Chapter 7, will require circuitry beyond what is presented here.

Hardware Validation

Fast target-camera relative motion and large contrast changes can compromise video quality. The electronic iris of the Sentech STC-N632 allows for a change in shutter speed, or exposure time, based on lighting conditions in order to achieve the correct amount of contrast. If large changes in lighting conditions between frames are present, the electronic iris may not be able to adjust fast enough, causing the resulting frame to appear either too dark or light. The interlacing of NTSC cameras does not perform well with fast motion, since each field is captured at different points in time. Rapid displacements of an object can cause faulty interlacing, with motion artifacts. These defects in the frames could generate issues for computer vision software, and thus should be mitigated.

To explore the extent of these issues with the selected camera, a rotation rig was designed and built (Figure 4-5). A high torque 12V 15 RPM motor is mounted on a Manfrotto video tripod. The battery and camera/test casing are spaced so the moments generated about the motor shaft are balanced. Using the built in spirit level, the rotating platform can be angled as desired.



Figure 4-5 *Camera rotation rig.* On the left is the overall rotation rig, which is balanced about the motor shaft. On the right is the 3D printed tripod/motor-mount connector.

Using indoor fluorescent lighting, the Sentech camera was used with the rotation rig to qualitatively observe the effects of rotational motion on video quality. The left image in Figure 4-6 displays a resulting frame from the recorded video, and the right image shows what the target with the camera motionless. The severe motion blur and poor interlacing can likely be attributed to the low lighting conditions, which causes the electronic iris to set longer shutter speeds. The shutter speed can be locked at a single rate, however the electronic iris will be required for outdoor applications where lighting conditions are variable within each rotation.



Figure 4-6 *The effects of motion and lighting on image quality.*

In order to test the video quality in a more realistic lighting setting, the rotation rig and camera components were taken to an outdoor setting. A light meter application was used to measure the lighting conditions, which came out to 31500 lux, and 6650K. The sun overhead and tree foliage

provided changes in lighting within each rotation. The target was distorted during several passes due to poor interlacing, however overall the video quality was significantly better than that from the indoor test. The linear interpolation deinterlacing MATLAB function was applied to the video (this method produced the best results with this footage), the results of which may be seen in Figure 4-7. The deinterlacing MATLAB function is an algorithm which uses a selected method to correctly align the fields in a video to remove motion artifacts [29].



Figure 4-7 *Deinterlacing frames containing significant camera motion.* On the left is the original footage with significant motion artifacts. On the right, it can be seen that by deinterlacing a frame the motion artifacts can be significantly reduced.

To verify the pixel calculations made earlier, field testing the camera was set at a distance of one blade length of a Vestas V90 wind turbine from a 23 by 23 centimeter target (see figure 4-8). The composite to USB video capture device crops the video to 720 (H) by 480 (V) pixels, so the pixel calculator was modified to account for this and the new distance. 345 pixels were estimated to be within the 23 by 23 centimeter target. The perimeter surrounding the target on a single frame was determined, and the area was calculated. The area was calculated to be 320 pixels, which provides a percent difference of around 7%. This error may be attributed to incorrect positioning of the target, and rounding between pixels.



Figure 4-8 *Target area calculations.* A 23 by 23 centimeter target was set at 45 meters (simulating a Vestas V90 blade length) from the camera. The corresponding image was analyzed by calculating the number of pixels within the target.

Camera Casing

The on-blade camera and corresponding components will require housing to protect the electronics from the elements and provide secure mounting to the blade. As discussed in Chapter 3, the housing may also contain vibrational sensors such as a contact microphone or accelerometer. For the purposes of a test enclosure, and demonstration of early design, a rapid prototyped casing was designed and printed. The casing had to be able to house a battery, micro-camera, and transmitter—all while being as compact as possible. Future iterations of the design are discussed in Chapter 7.

The proposed mounting location for the camera is on the root of the blade, placed so that the FOV encompasses the entirety of the face of the blade. Generally, for large wind turbines such as the GE 1.5MW, the turbine blade transitions from an airfoil style cross section from the tip and across the

face of the blade, to a circular cross section near the root. This circular cross section is large in diameter, so a small camera mounted tangentially should have a nearly flat surface on which to be placed. If placed correctly, the camera should be able to observe any avian or bat interactions and collisions occurring near the face of the blade on which it is mounted.

Mounting of the casing may be tricky, requiring either a scissor lift or access from the nacelle depending on the turbine. Adhesive strips along the bottom of the case, and pull tabs for easy removal, should provide sufficient mounting of this lightweight design. Moisture, and the acidity in rain water may degrade the adhesive bond over time. Vibrations, aerodynamic forces, and the seemingly ever present pull of gravity comprise the three elements which will ultimately lead to the failure of this joint.

3D printing is an excellent manufacturing choice for the on-blade camera casing. The quantity needed would be relatively small, which fits the longer lead time of printing a large part. Printing allows for easy manufacturing of complex shapes and contours, while requiring a minimal amount of hands-on work. This manufacturing method also opens the door for on-site production of the on-blade system, which could reduce the costs of implementation for wind farms. Another benefit is the ease of modification of the casing, should a wind farm require a differing bolt pattern, shape, or other change in features.

For testing, a three cell (11.1V nominal) lithium polymer battery was selected to support the camera and transmitter. The battery chosen was high capacity to support multiple tests across a span of several days, with the battery connection being unplugged between uses. The primary tradeoff of this high capacity battery is the need for larger housing. Maintaining a low profile in the casing design is imperative for a marketable unit, which will be sleek in appearance and create a negligible drop in efficiency in the turbine blade.

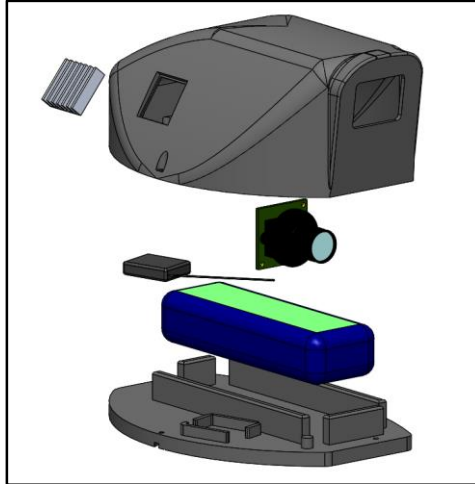


Figure 4-9 *Camera casing expanded view.*

The camera casing was designed to be as compact as possible, while maintaining an aerodynamic shape in the direction of airflow. Support structure for holding components in place was designed to be robust but minimal to reduce the amount of material needed for printing. The final drawings specifying key dimensions may be found in Appendix C. Bolt holes and hexagonal cut regions for nuts were sized with additional space, to account for the tolerance of the 3D printer selected. On the trailing side of the camera casing, a heat sink was added for a voltage drop unit before the transmitter: maintaining a low temperature within the case is important for extruded materials.



Figure 4-10 *3D printed camera casing.*

The first iteration of the camera case (figure 4-10) was printed using PLA plastic, whose brittleness and low melting temperature make it less than ideal for a final case. ABS plastic is a common 3D printed (extruded) material, which should be easily used for this casing if the proper 3D printer is utilized. A heated bed and closed printing area are especially important for larger and detailed print jobs. Some warping of the base plate was noticed after the printing process, as seen in figure 4-11. The bolt holes and hexagonal sunk regions were correctly sized, and the dimensions provided adequate spacing while not being loose. Assembly time was low, requiring around a half hour for bolting the casing together, applying an aluminum heat sink, and sanding the outside of the case.



Figure 4-11 *Warping of the camera casing base plate.*

5. Software Development: Avian Detection and Tracking

In this chapter, program architecture for avian detection and tracking is determined. A preliminary version of vision-based avian interaction and collision sensing program is developed in MATLAB and tested. The performance of the program is examined, which dictates whether this program will run continuously during the day, or be triggered by other sensors.

Overview

For the purposes of this program, daylight is assumed for operation. Additionally, although bat flybys and collisions could occur during the day, only birds will be evaluated in this version of the program. Any daytime monitoring techniques investigated here should apply similarly to bats. There are two types of interactions this program is concerned with: a flyby, where the subject enters the blade-mounted camera FOV, but does not collide with the blade, or a collision, where the subject has direct impact with the turbine blade. Collisions are of primary concern for wind farm operators, so being able to rate the likelihood of a collision versus a flyby is imperative.

In future versions of the program, images of avian interactions and the likelihood of collision will be coordinated with data from the other vision-based, vibrational, and bioacoustic sensors. The program developed in this chapter outputs graphics for user review and evaluation of program performance.

Having a camera mounted on a wind turbine blade poses challenges for vision-based algorithms including a dynamic background and vastly varying lighting conditions. Shadows, cars, light blooms, and other wind turbines are just a few of the factors which may present themselves within the FOV during operation. Cameras mounted on offshore wind turbines will see waves, whose constantly changing structure creates textures that may generate a large number of FP instances.

Despite these detracting factors, there are a number of aspects owed to the nature of this camera placement which can be taken advantage of in the program. Assuming constant blade pitch and turbine yaw:

- Objects which are stationary, or nearly stationary, such as buildings, clouds, and hills, appear to move in the same direction from the perspective of the rotating camera. (Figure 5-1)
- Stationary objects will follow the same path across the image plane during each rotation, while moving objects may deviate or disappear depending on their trajectory. (Figure 5-1)
- The face of the blade for large wind turbines is generally white and free of markings, which provides an excellent background for thresholding.

In actuality the turbine may change pitch or yaw based on wind conditions, however adapting to these changing conditions is a feature to be addressed in later versions of the program.

As data from [15] suggests, even regions containing high bird mortality rates present a low quantity of collisions when considering the amount of time captured on camera versus the quantity of events. Due to this low frequency, it is key that the program have a true positive (TP) rate approaching 100%. A TP refers to the correct labeling of a target by the program. It is far more important to capture a bird flyby or collision and have a high FP rate, than to have zero FP instances and not record a collision. Filtering and the use of data from other sensors may provide a means for eliminating FPs. Birds may also appear in flocks instead of a single target, requiring the ability of the program to handle several targets in the FOV at any given time.

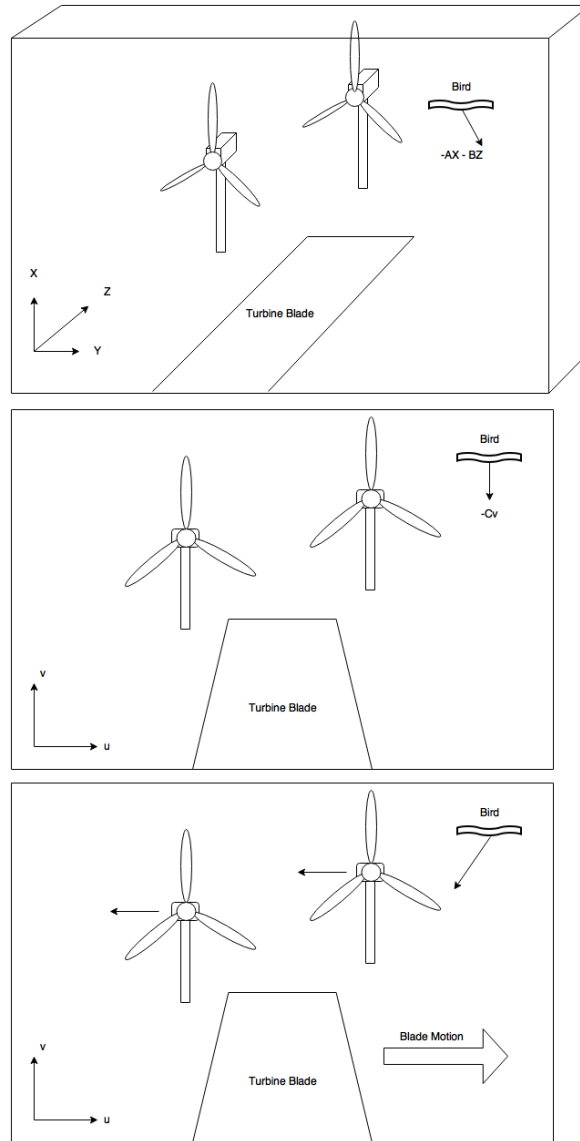


Figure 5-1 *Frame of Reference.* The upper image represents the 3D world coordinate system. The middle image shows the perspective of the blade mounted camera when the turbine is stationary. The bottom image shows the apparent motion of all objects in the view plane during turbine rotation.

Overall Structure

The overall structure of the program is designed to take an input video (.mp4, .avi, etc.), and using a for-loop, obtain one frame at a time for analysis. At the conclusion of the video, the data obtained is analyzed and presented to the user. Avian detection is the forefront of the program after video

input, and the chosen method has a large effect on the overall performance of the program for successfully classifying instances. Figure 5-2 outlines the general program architecture that is be used.

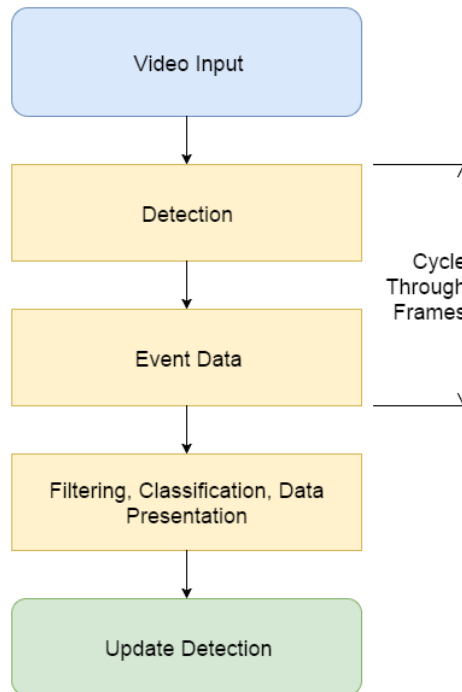


Figure 5-2 *General program architecture.* The input video is loaded into the program. Detections and event data are extracted for each frame. Event data refers to any additional information pertaining to the detection or otherwise which is used for later analysis. At the conclusion of the video analysis, the final two sections occur: filtering, classification, data presentation, and updating the detection system. This format does not reflect a real-time format.

The structure used for this version of the program is conducive to a saved video input, however it will need modification to be paired with a live video stream. The speed of operation provides insight to determine the feasibility of real-time avian and bat interaction detection. To demonstrate the operation of the program when operating as triggered by other sensors, an option is incorporated into the MATLAB program allowing for an input signal to trigger video analysis.

```

Continuous [1] or Triggered [2]? 1
Would you like to record this session? [1] yes [2] no 2
Should the detection algorithm be updated after processing? [1] yes [2] no |

```

Figure 5-3 *Opening User Prompt.* To facilitate the use of this program, the user is asked for their input pertaining to various settings pertaining to operation.

Instead of analyzing the entire video, the triggering setting analyzes a frame buffer surrounding the time of the event as determined by a peak in the signal. The units of time for the input signal must be converted to frames, which correspond to frames within the video (figure 5-4).

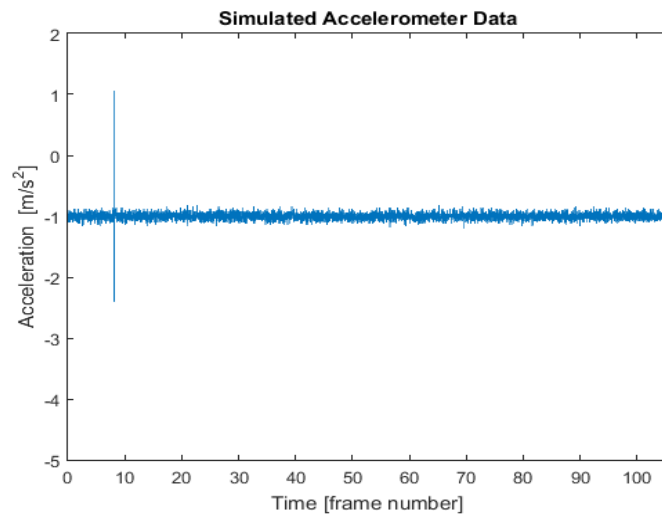


Figure 5-4 *Example of input signal simulating vibrational sensor data including a collision.* This signal would have a frame buffer surrounding the spike in acceleration around the 10th frame. That frame buffer would then be analyzed by the program, as opposed to the entire video.

Filtering, classification, data presentation, and updating the detection system have a post-processing format, where they operate after the video is done being analyzed. In a real-time setting these items would need to be reconfigured to support frame-by-frame analysis. The overall program architecture may be seen in detail in Appendix D, and the MATLAB code along with all supporting functions may be found in Appendix E.

Detection Options

As a starting point for the program, a detection method was selected. The purpose of detection in this program is to determine the presence of birds in the frame. Two methods of avian detection were examined, including optical flow and cascade classification. Both of these methods are supported by MATLAB Computer Vision System Toolbox, which is a collection of image and video processing algorithms. A commonly used technique for determining the presence of a bird, or other moving target, is frame differencing. Frame differencing compares two video frames, and looks for changes between the two in order to differentiate between the background and foreground [30]. A binary image represents the difference between the frames, by highlighting foreground with high intensity pixels. Background subtraction uses a learned “background” image, which is removed from the current frame, yielding the foreground [30]. The basic differencing techniques lose their viability with camera motion and changing lighting conditions.

Optical Flow

As explained earlier, a bird flying through the FOV will likely not follow the motion flow field induced by the rotating camera. In order to take advantage of this potential deviation in motion, optical flow was investigated as a detection method. Two methods provided with Computer Vision System Toolbox includes Horn-Schunck and Lucas-Kanade. These optical flow methods originate from the constraint of brightness consistency. Brightness consistency dictates that the intensity of a point on an image should remain the same after a small change in time and position [31]. With an image coordinate system x and y , the following equation is derived [31]:

$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t) \quad \text{Equation 5-1}$$

Expanding the right side via Taylor series, and truncating to the first order terms [31]:

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \delta x \frac{\delta I}{\delta x} + \delta y \frac{\delta I}{\delta y} + \delta t \frac{\delta I}{\delta t} \quad \text{Equation 5-2}$$

Combining equations 5-1 and 5-2 produces the following [31]:

$$\delta x \frac{\delta I}{\delta x} + \delta y \frac{\delta I}{\delta y} + \delta t \frac{\delta I}{\delta t} = 0 \quad \text{Equation 5-3}$$

Or, by dividing by the partial derivative with respect to time [31]:

$$I_x u + I_y v + I_t = 0 \quad \text{Equation 5-4}$$

The movement of pixels across the image can now be obtained using equation 5-4. With the two unknowns, u and v , additional constraining equations are needed. Horn-Schunck utilizes a global technique, meaning that the entirety of the image is analyzed at once [32]. The error of the optical flow and deviation from global smoothness is combined, and the resulting equation is minimized by setting the derivative (with respect to u and v separately) to zero [32]. Iterative solving then allows for u and v to be determined [32].

Lucas-Kanade developed a local method for optical flow, which uses small pixel neighborhoods [33]. For example, considering a 3x3 pixel cluster, applying the brightness consistency equation (equation 2) to each pixel would yield 9 linear equations [33]. The motion (u, v) is then assumed to be constant within this neighborhood [33]. By putting these equations into matrix form, Least Squares may be used to find a single approximate solution to this over-constrained system [33].

Both of the aforementioned methods have their benefits and drawbacks. The global smoothness constraint of the Horn and Schunck method limits extreme deviation from the motion field [34]. Lucas-Kanade has difficulty determining optical flow in regions of similarity, or when the gradient is completely random [33], such as speckle. Due possible large deviations from the motion field during blade rotation, and lack of large homogeneous surfaces (barring the blade face, which does not need to be tracked), Lucas-Kanade was determined as the primary method to be tested.

The footage obtain in Chapter 4 during field testing with the Sentech camera and rotation rig is suitable for testing the reliability of Lucas-Kanade optical flow for estimating motion flow fields with on-blade applications. Brightness changes seen during rotation are similar to that obtained from an on-blade camera. By applying the Lucas-Kanade method to this footage, observations can be made about the ability of the algorithm to determine the motion flow field during fast camera rotation.

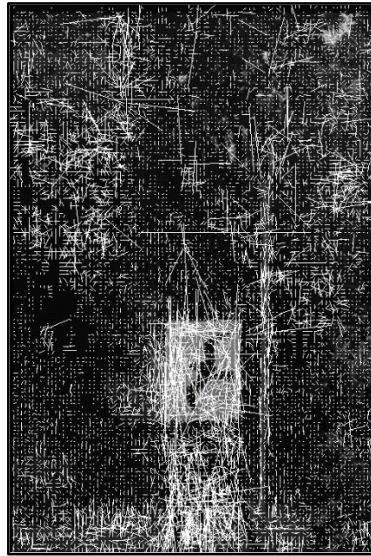


Figure 5-5 *Lucas-Kanade Optical Flow Method.* An example of optical flow applied to rotating camera footage, with low and high level noise filtered. The lines are motion vectors between frames, and are scaled to represent the actual distance travelled by a pixel.

The fast rotation rate of the camera (15 rpm) in combination with varying lighting conditions caused significant noise with optical flow. A low and high pass filter were applied in order to reduce noise, however the results were not significant enough to mitigate the severe noise present in regions containing large shifts in illumination (figure 5-5). Additionally, if a bird flies within the blade sweep plane, it may not deviate enough from the motion flow field to stand out. Due to the apparent limitations of the basic optical flow algorithms offered on MATLAB, the solution of optical flow was dropped in favor of another technique.

Cascade Object Detection

The use of objection recognition is a wide area of research within computer vision. The MATLAB Computer Vision System Toolbox offers a trainable cascade object detector, for the recognition of objects which do not vary greatly in terms of aspect ratio [35]. This decision tree detector is trained using supplied negative and positive image datasets, the latter of which may be labelled using a built in image labeler [35]. There are three choices of feature types integrated into the MATLAB cascade classifier, including Haar, local binary patterns (LBP), and histogram of oriented gradients (HOG) [35].

HOG features are found by examining the distribution of intensity in a small, generally square, region in an image in order to approximate the direction of the gradient using a histogram [36]. Each pixel in the region represents a data point in a histogram to find the best fit orientation [36]. This feature type is often used in human and vehicle detection, where the shape of the object is generally invariant. Haar features are comprised of rectangular regions where the intensities of one region are differenced from another [37]. The value obtained from the aforementioned differencing provides a data point which may be compared to a threshold to determine if the criteria of the desired feature is met. Local binary patterns (LBP) is a method that is robust to changes in illumination, as it focuses on texture [38]. For a grayscale image, the LBP is created by comparing the intensity value of a pixel to its neighboring pixels [38]. Neighboring pixels are assigned a 1 or 0, the latter of which is assigned if the intensity value is less than the center pixel [38]. The values found in this manner are entered into a histogram, which is normalized to obtain the LBP [38].

Testing performed in “A Comparison of Image Processing Techniques for Bird Detection” by Elsa Reyes demonstrated much higher accuracy using Haar than HOG features, and generally higher accuracy using Haar as opposed to LBP features for bird detection [39]. The tuned accuracy using Haar features was determined to be 87% [39]. Also confirming the use of Haar-like features for avian detection applications, is a publication by R. Yoshihashi, R. Kawakami, M. Iiada, and T.

Naemura, who compared HOG and Haar-like features using a collected image dataset of over 32,000 TP bird images, and over 4,900 negative images [40]. It was found that Haar-like features outperformed HOG features for lower resolution avian detection [40]. Through both of these articles, large data sets were examined, and the use of a cascade classifier using Haar-like features has been shown to be a potentially reliable choice for avian detection.

The cascade object detector is based on the Viola-Jones algorithm, where detection involves stages to decrease processing time [37]. A sliding window is moved across the image, and the contents within the window are tested in stages. Failing to pass a stage leads to labeling as a non-target, and the window moves to the next section of the image [37]. The advantage of this method is saved time, by moving past non-targets faster than checking each stage before moving to the next region.

The cascade classifier in the MATLAB environment trains each stage individually via a process called supervised learning, where labelled images and negative images are provided [35]. Each stage is comprised of an ensemble of weak learners [35]. The weak learners may be comprised of the features mentioned earlier, or some other feature type depending on the application. Generally, the number of possible weak learners is much greater than what is needed to detect an object, so some method of selecting the best weak learners is required [37]. Adaboosting is used to weight weak learners and ultimately determine which will be used in each stage [37].

The effectiveness of cascade object detection is entirely dependent on how the detector is trained. Tuning of the positive image sets, negative image sets, number of stages, and false positive rate, and false alarm rate are a few of the variables which must be managed to create a reliable detector. The initial cascade classifier training set involved the use of around 80 labeled bird images that varied in perspective and point in the flapping cycle, and utilized HOG features (the default setting for detection training). For nearly every orientation and point in the flapping cycle, the beak or wingtips are generally visible on the view plane, which helps in maintaining consistent features for detection.

The detection system was comprised of around 90% albatross images, along with approximately 10% murrelet images. Albatross species tend to have distinct sharp wings and beaks, which are exaggerated in comparison to other avian species. Additionally, variants such as the Short-tailed Albatross, or *Phoebastria albatrus*, are listed as endangered by the U.S. Fish & Wildlife Service [41]. The false alarm rate and number of cascade stages were varied until detection was consistent. HOG feature based detection was used during testing and program development, however [39] and [40] demonstrated that Haar features provide improved results over HOG features for bird detection.

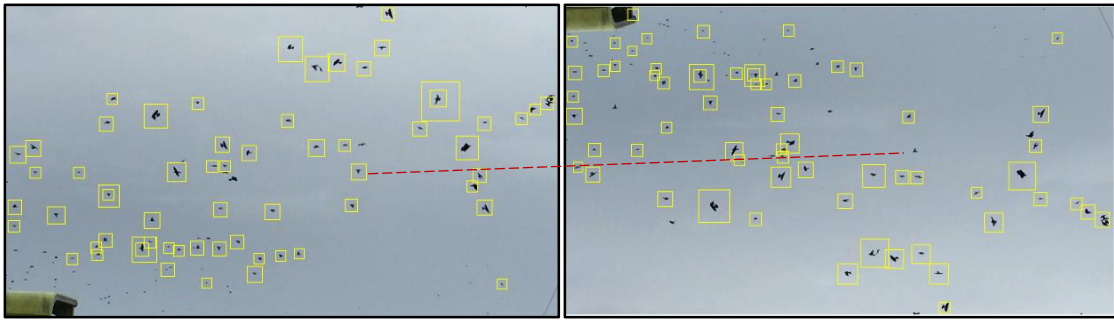


Figure 5-6 A comparison of detection accuracy when rotating the input image. By rotating the input image, it can be seen that certain targets are no longer detected. The dashed line demonstrates a loss of detection between the upright and rotated image. The same detector and merge threshold settings were used for detection in these images.

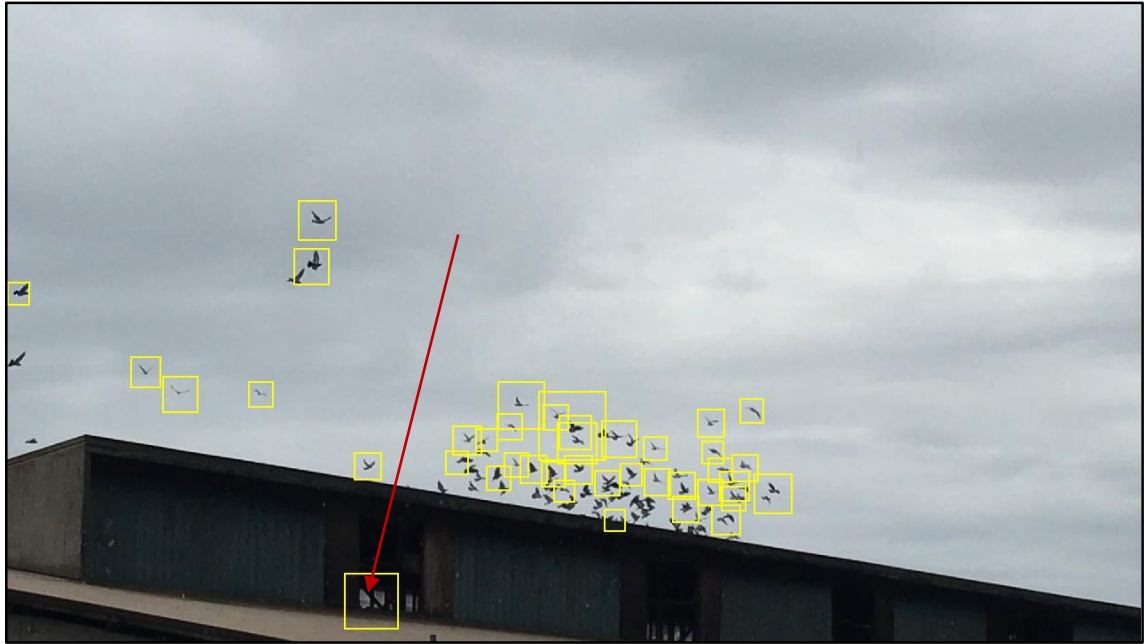


Figure 5-7 *Examples of cascade object detection applied to input bird images.* Using the HOG feature based albatross detector, other species were able to be detected. Note the FP instance indicated by the red arrow, the causation of which was likely the truss structure within the barn.

As may be seen from figures 5-6 and 5-7, the albatross based detection system is able to successfully detect numerous other species. Ground truth testing was not performed on the detection system; larger positive image datasets should be utilized before creating and testing the next iteration of cascade detection for this program. The detection system provided satisfactory results for testing the program operation and framework performed here, however before testing the operational capabilities of the program in a true wind farm setting, a more robust cascade object detector should be trained.

Blade Face Thresholding

The wind turbine blade provides a white backdrop which creates a potential platform for thresholding. For a grayscale image, thresholding is the creation of a binary image by setting an intensity value which when exceeded yields white (255 with 8-bit depth), and otherwise creates black (0 with 8-bit depth). Due to the high intensity of the wind turbine blade, and generally darker

colors of avian and bat species, the thresholded image is inverted. A bird or bat flying into the face of the wind turbine blade should provide a contrasting region, or blob, assuming their coloration is dark enough to trigger the threshold (figure 5-9). When starting the program, a pop-up window prompts the user to input a polygon surrounding the blade (using `impoly` MATLAB function), as can be seen in figure 5-8.

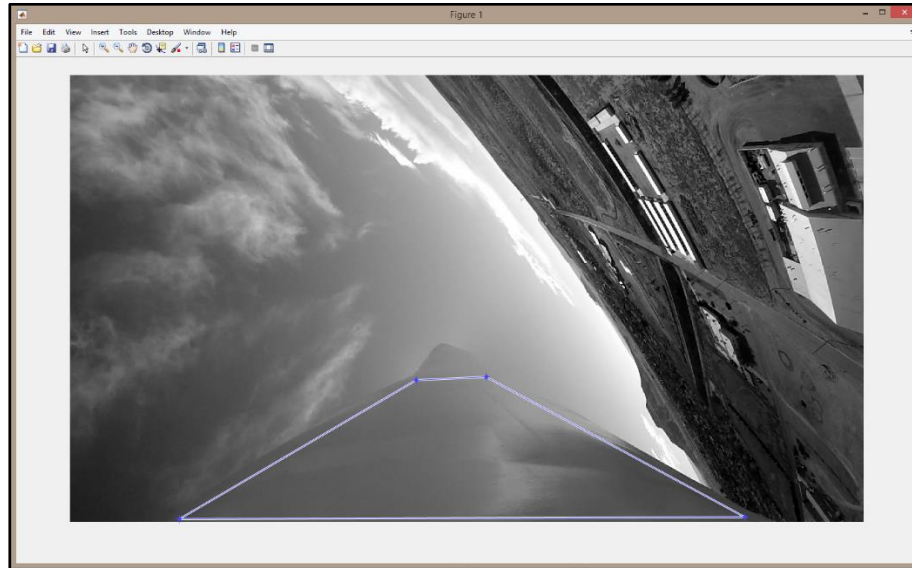


Figure 5-8 *Selection of the blade ROI.* The user is prompted to select the blade ROI before the program analyzes video.

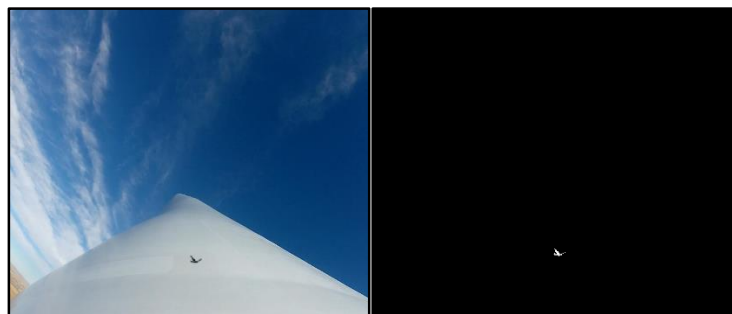


Figure 5-9 *Thresholding and its results.* The creation of the binary image on the right was performed using the ROI selection and thresholding components of the program.

For each input frame, the blade region selected by the user is thresholded, and blob analysis is performed inside the blade ROI. Blob analysis is the examination of high intensity regions in a binary image. Minimum and maximum values for blob area can be provided to filter out noise and shadows respectively. Additionally, the number of blobs can be limited, as more than one or two birds impacting the blade at once is extremely unlikely. Information about blob presence in each frame is passed on to the final classification stage, to aid in determining whether or not a collision has occurred.

Tracking

The detections made by the cascade object detector are tracked between frames to find the trajectory of the bird. The use of tracking is not only for determining the trajectory of the bird through the image plane, but also for assigning an identification number to the detection. Identification numbers are primarily for filtering and data management purposes, so that each detection can be recalled from the global data set for evaluation. Large wind turbine rotational speeds between 90 and 120 degrees per second [13] leave little room for error in the tracking process: at 30 frames per second, and aligning the horizontal FOV (11.14°) with the direction of motion, the rotating camera can only capture around 3 to 4 frames containing the target (with no significant target motion in the vertical direction).

Primary Tracking

The primary tracking framework was obtained from the Motion-Based Multiple Object Tracking example from the MathWorks website [42], and consists of a MATLAB data structure, containing the ID of the target, bounding box data, Kalman filter data, detection age, the number of visible instances, and the number of consecutive invisible instances. A bounding box is simply a box surrounding the object in question, with coordinates given in pixels measured from the upper left corner of the image. Kalman filtering allows for the prediction of motion for targets which existed

previously, but were not successfully detected in the current frame [42]. The predictions made using Kalman filtering keep the location of the target updated, so that when a future detection occurs, the predicted versus true location of the target are not greatly divergent [42]. The option for constant velocity for Kalman filtering was selected, as the rate of bird and bat motion across the image plane is not expected to change greatly between frames.

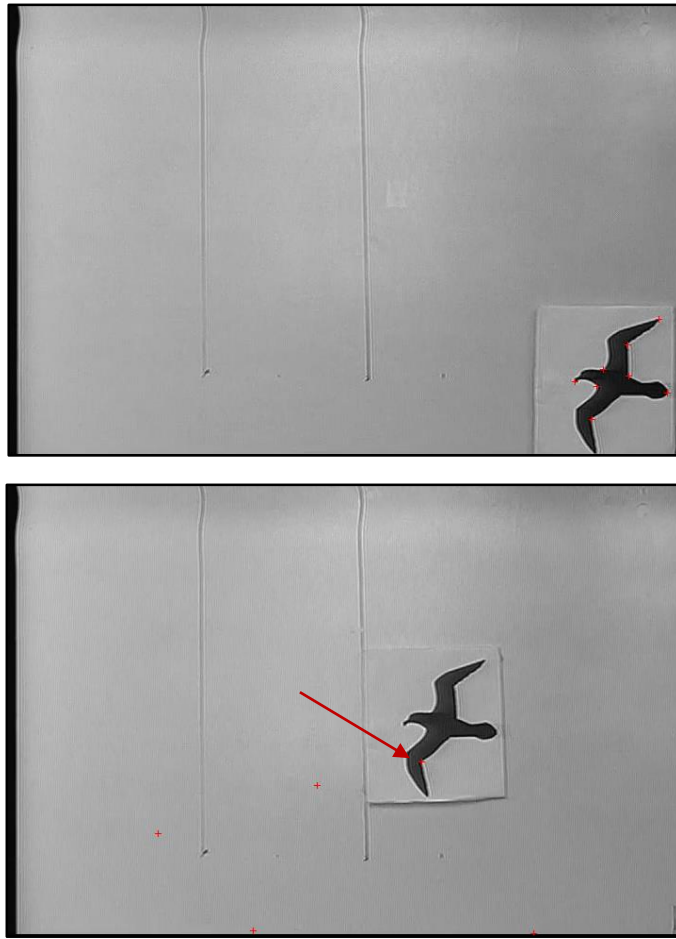
The importance of assigning a detection to a path, is both the capability of the program to match an identification number with the detection, and for determining the trajectory of the target. Assignment to a path is based off of the James Menkres' variation of the Hungarian assignment algorithm, which is used to output assigned tracks, unassigned tracks, and detections [43]. A mismatched pairing between the path and detection could lead to incorrect trajectory data.

If a path exists for too long, or does not have detections assigned to it for an extended period of time, it is deleted. Once this deletion occurs, the path is evaluated. Assuming more than one detection was assigned to the target, location information is used to evaluate the trajectory. The fast rotation rate of the camera means that the path of the avian target should be approximately linear, therefore a linear line of best fit is created using linear regression. The R squared value provides details on the quality of the tracking. Finally, the angle of the path travelled is calculated. If the angle is over 15 degrees, the target is considered to deviate significantly from the motion field. An important distinction is that a target flying away from the wind turbine blade plane is less likely to have collided than one that is moving toward the blade plane.

Secondary Tracking

If only one detection exists for a target, a secondary tracking method is used in an attempt to capture the path of the target. A single detection does not allow for the primary tracking method to function, since the Kalman filter requires at least two detections to make predictions for velocity and future location (assuming no target acceleration). The Kanade Lucas Tomasi (KLT) feature tracking algorithm in MATLAB Computer Vision System Toolbox [44] provides qualities that make it a

viable candidate for this application. Using the bounding box created using the singular detection, feature points are determined by finding the minimum eigenvalue features [45] for the region. The strongest of these points is then tracked between the preceding and proceeding three frames from the detection- this provides the best possible coverage of the event in order to extract the path of the target. Between each frame, the points which the algorithm determines as outliers are removed, so that the final point distribution reflects only successfully tracked feature points. The path travelled is determined in a similar manner to the primary tracking method, where the angle of the direction of travel gives insight to the deviation from the motion field.



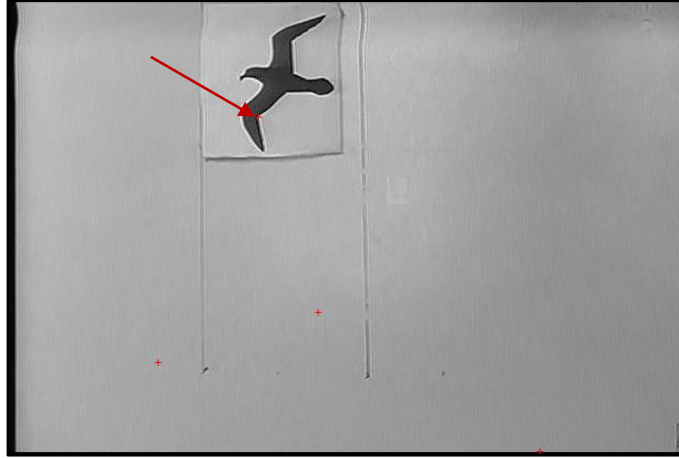


Figure 5-10 *Large displacement tracking via point tracking.* This example provides a low noise background with an unchanging object, which is displaced to simulate a bird traversing the image during camera rotation. Two points (overlapped) are successfully tracked during the image sequence.

The above example of point tracking required the tuning of tracking parameters to successfully track the target. In this example, the percentage of successfully tracked points across the three frames was 20%. There are critical parameters that can be varied to achieve point tracking with large object displacement between frames, including the number of pyramid levels, block size, and the quantity of iterations. To fully validate this method for the use of high displacement tracking, sets of images containing bird movement that is 1/4 to 1/3 of the image width between frames should be created. Using this data, the parameters can be optimized for this application.

Trajectory Classification

After a target disappears and tracking is complete, the trajectory of the target is classified. Four types of trajectory classifications were created to organize detections. Table 5-1 describes the types of events, and figure 5-11 provides a graphical display trajectory classifications.

Table 5-1 *Tracking Classification Types.*

Type	Description
1	Greater than 15° deviation from the motion field direction, and towards the blade plane.

2	Less than 15° deviation from the motion field direction.
3	Unable to successfully track the target.
4	Greater than 15° deviation from the motion field direction, and away from the blade plane.

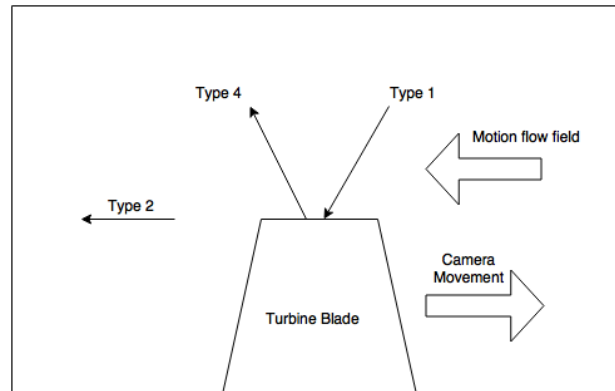


Figure 5-11 *Example of target motion and corresponding classification.* Type 3 classifications are a single detection, and do not have a determined direction of travel.

Supporting Structure

Detection and tracking provides a set of raw data which must be sorted and interpreted before passing on to the user.

Data Storage

The data structure for tracking is updated between each frame as new information replaces existing data. This saves time when there are no active paths present, by eliminating the need for adding additional data to a matrix. When detections or predictions occur, a matrix stores all relevant detection and prediction data including the ID, bounding box, centroid, detection versus prediction, and the frame number. For final evaluation, all data is retrieved from the matrix as opposed to the structure, so that data from any point in the monitoring process can be reviewed. Information about the trajectory of the target is stored in a separate matrix, as it exists on a per target basis as opposed to a per frame basis.

Filtering

Stationary objects triggering a FP response will cause a large quantity of non-avian or non-bat images which must be sorted through by the user. As a post-processing solution, a spatiotemporal filter was developed to eliminate repeating instances. Figure 5-12 outlines the process for obtaining repeating instances. Instances which were determined to be repeating are grouped together and passed to the next stage of filtering.

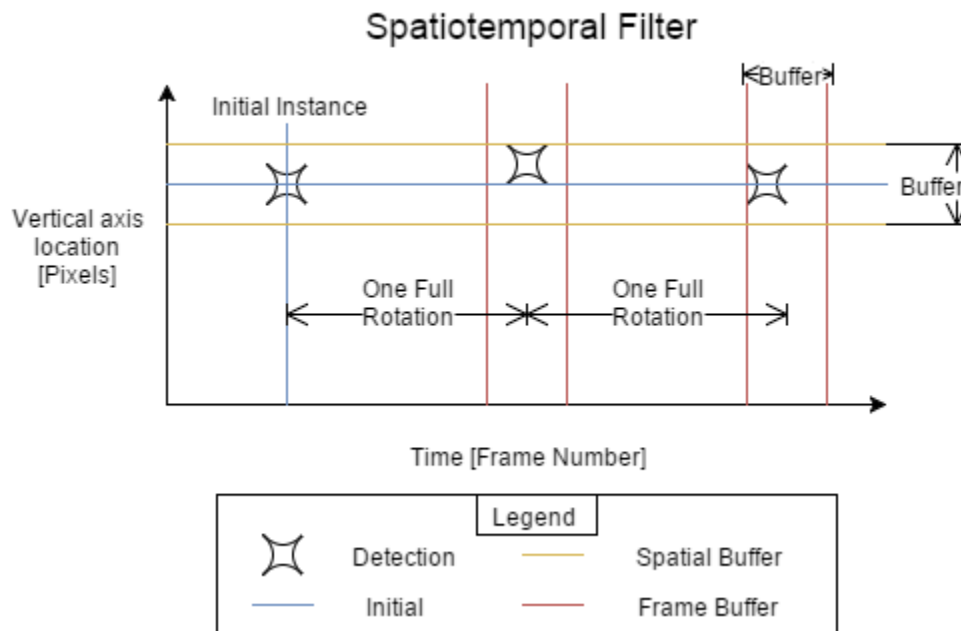


Figure 5-12 *Spatiotemporal filter principle.* After a detection, frames are examined at one, two, and three rotations from the initial instance. If two or more of these instances correlate closely to the initial detection in vertical location it is considered to be a repeating instance.

Relying on time and space alone for the filtering process could lead to the deletion of TP detections. In order to decrease the probability of such error, a comparison of the structural similarity between each detection in a group of repeating instances is made. The structural similarity index (SSIM), is a method for comparing the similarity of two images. The SSIM is scored 0 to 1, where 1 is an exact match [46]. SSIM is often used for determining image quality after compression, where a reference image free of distortion is compared to a modified image [46]. Repeat detections should

have the same object within their bounding box, meaning that the SSIM score should be close to 1. Those object which deviate significantly in structure (<0.65) are removed from the filtering process to be classified and presented to the user.

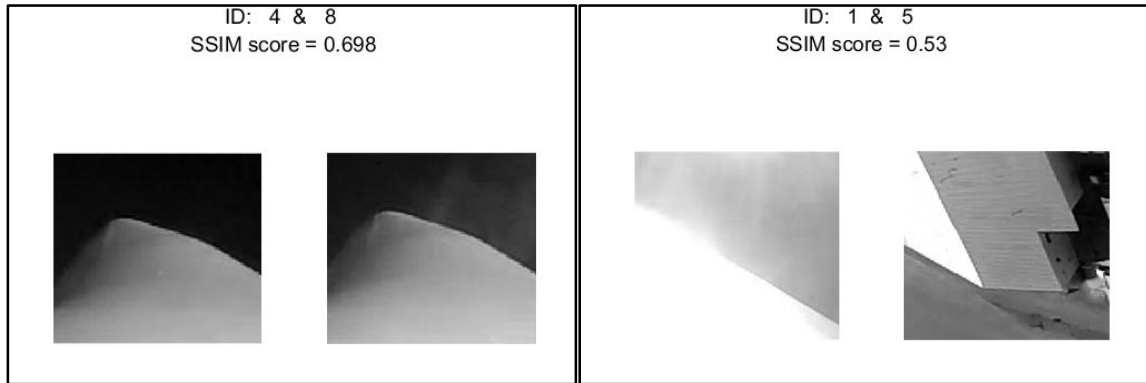


Figure 5-13 *SSIM comparison between detections.* Notice that the two sections compared on the left image encompass the same object, however the SSIM score is less than 0.7; this is likely due to the higher location of the blade tip and cloud presence in the right section. The right image shows a successful comparison, where the SSIM score reflects the large difference in objects that were detected.

The SSIM index accounts for luminance, contrast, and structure when evaluating the similarity between images [46]. A possible interference with this method is the shifting of bounding box location and size between rotations.

Collision Sensing

Once data collection is complete, and filtering has removed any significant quantity of FP instances from the data set, avian interactions must be classified based on the likelihood of collision. Likelihood of collision is decided based on three factors, the combinations of which were subjectively evaluated to determine how likely a collision occurred:

- Bird presence within blade ROI, as determined by thresholding and blob analysis
- Closest proximity of the target to the blade (further explained in Appendix F)

- The trajectory of the target (angle with the horizontal, and approaching or departing from the blade)

A table of the corresponding likelihood of collision may be found in Appendix F. Success of tracking is based on whether or not the primary or secondary tracking methods were able to create a reliable linear line of best fit.

Data Presentation

After the classification of instances, the data is ready to be presented to the user. In future versions, when integrating with the whole bird collision detection system, these results would instead be analyzed with any relevant data from the other sensors. For this program, presentation of data involves presenting the user with a separate window for each detected bird. The window provides whether or not the target was successfully tracked, the likelihood of collision, as well as any frames containing detections of the target (figure 5-14). The user is then prompted to enter the ID numbers of any FP instances presented.

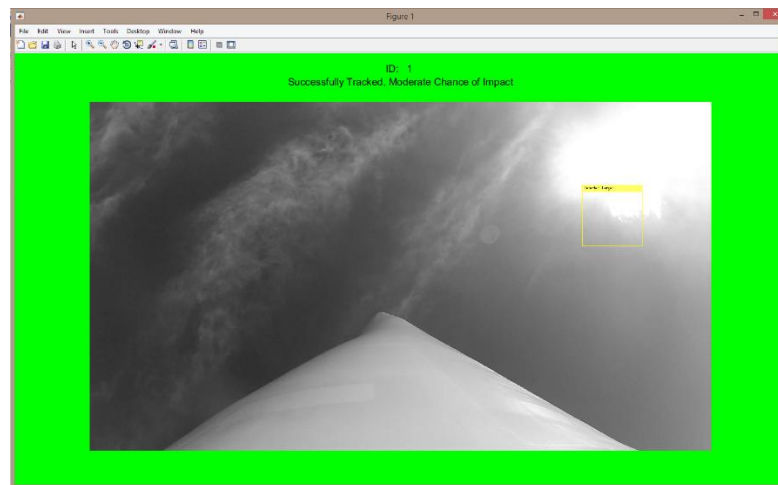


Figure 5-14 *Example false positive result in a window presenting the detection to the user. Results are color coded based on the collision likelihood, which assists the user in quickly locating detections which strongly correlate to an impact.*

Detection Improvement

Detections removed by the spatiotemporal filter, as well as any user inputted FPs from the data presentation section are added to a negative image set automatically. The cascade object detector then retraines using the new negative image set. Objects that trigger a large number of FP instances will saturate the negative image set, thus ensuring the immunity of the next iteration of the cascade classifier to that object. In a real-world setting, the program should be run on short time spans after installation, providing multiple opportunities for the object detector to improve. After a sufficient number of iterations, the object detector should be significantly better adapted to the environment of the wind turbine.



Figure 5-15 *Examples of repeating or user input FP detections.* The three images starting from the left comprise repeating FP detections caught by the spatiotemporal filter. The two images on the right comprise user inputted FP detections.

GUI for Detection and Primary Tracking Tests

Rapidly adjusting parameters for the Kalman filter tracking system and the cascade object detector is imperative for conditioning the system for a new setting or environment. As a tool for quick evaluation and adjustments, a graphical user interface (GUI) was produced using the MATLAB graphical user interface development environment (GUIDE) [47]. The options included on the GUI were the most frequently used during the tuning process: video filename, measurement noise, cost of non-assignment, and the merge threshold. Stages of the cascade object detector may each detect an object, meaning that multiple detections on the same object indicates a stronger overall detection; the merge threshold requires multiple overlapping detections on an object before

classifying the region as a detection [48]. Tuning this merge threshold is useful for reducing FPs (at the cost of increasing the chance of missing a TP instance) [48]. Measurement noise and cost of non-assignment are properties of the Kalman filter tracking framework.

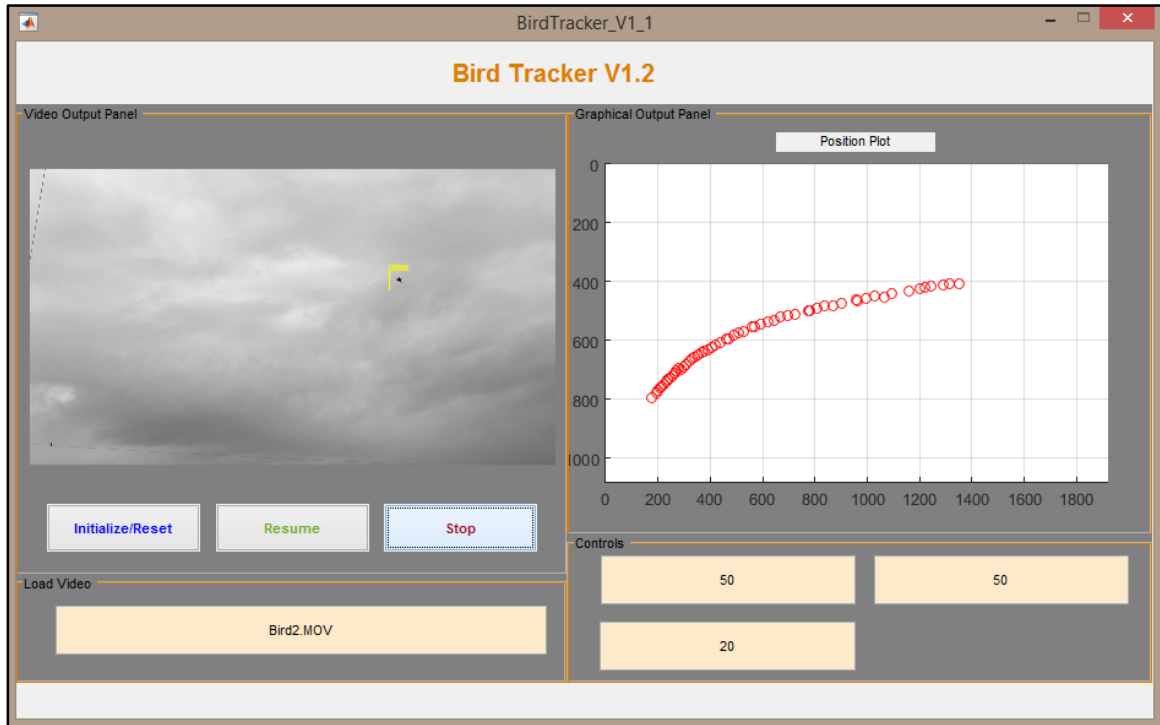


Figure 5-16 Example of successful tracking in the GUI.

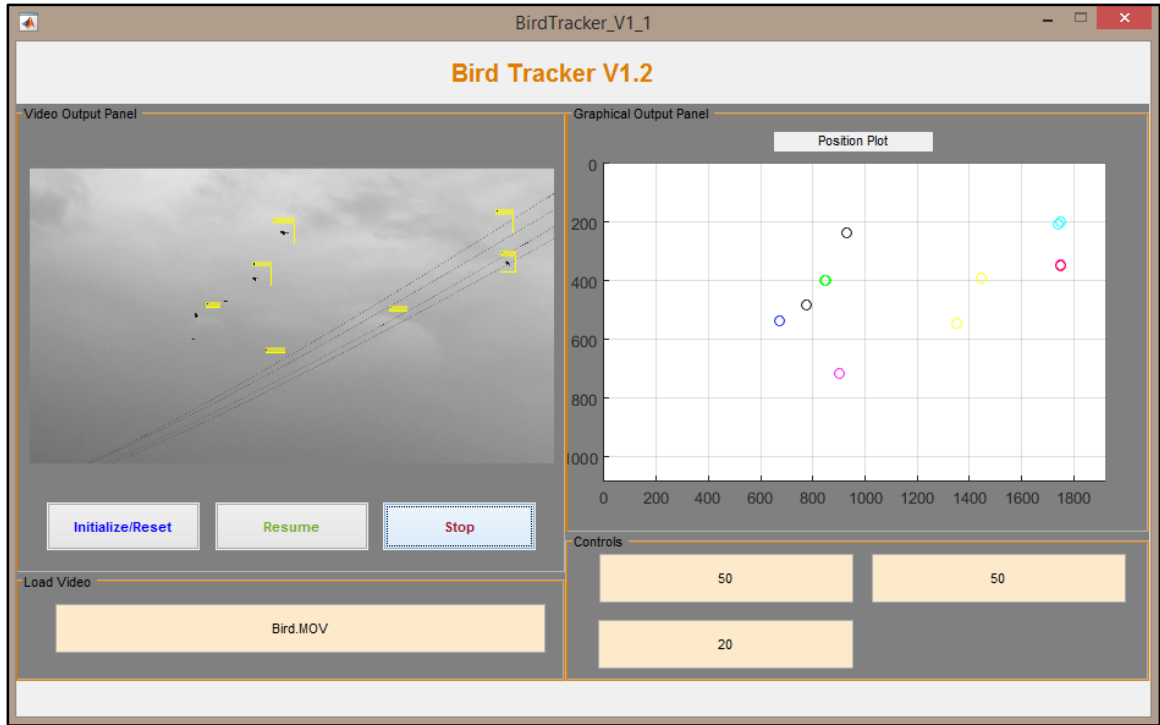


Figure 5-17 Example of less successful detection and tracking within the GUI.

The GUI displays the video on a left panel for viewing the frames as they are process, and the right panel shows the position of current and past tracked objects. Axes of the position plot are in pixels, displayed in the format that MATLAB interprets images. The GUI code and figure structure may be found in Appendix G. The videos analyzed above do not reflect a similar tracking situation to what is seen using an on-blade camera, however they provide high level insight to the behavior of the detection system when bird orientation and point in the flapping cycle is varied.

Operational Testing

A final proof of concept test demonstrating the operation of the entire program was needed for holistic validation. The purpose of this test was to demonstrate the program architecture in its operation, particularly the filtering and classification aspects of the system. The input video was high resolution footage taken from a blade mounted camera (1920 by 1080 resolution at 60 FPS). At a length of 6323 frames (1:45), the video was divided into 2000 frame segments for three trials.

The first 200 frames not used in the trials were added to the negative image set for the albatross based cascade detector. The settings of the trained cascade detector included a false alarm rate of 0.4, 19 complete stages, and HOG features.

To provide a target for tracking in the final trial video segment, a murrelet target was added to simulate a bird impact on the face of the blade. The ability of the beginning cascade detector to locate this target was ensured before testing, and the merge threshold was set to the highest value which allowed for consistent detection. By maintaining a higher merge threshold, FP detections are less frequent.

The same detector training parameters (0.4 false alarm rate, 19 stages, HOG features), and merge threshold were maintained through all three trials. The difference in FP instances between trials 1, 2, and 3 were examined, as well as the successful detection and classification of the murrelet in trial 3. Frequent sources of FPs should be picked up by the spatiotemporal filter, and a decrease in FPs should occur between each trial. The results from this test should provide insight to the overall operation of the program in a true on-blade camera environment. It is imperative to make the distinction between the footage used here, and the differing resolution and frame rate of the proposed Sentech STC-N632 micro camera.

Trial 1

For the first 2000 frame video segment, there were 47 detections overall, 38 of which were successfully removed as repeat false-positives (approximately 81%). The majority of the FPs that were deleted were due to clouds and infrastructure on the ground.

Table 5-2 *Trial 1 results, outlining the cause of the detection, success of tracking, and likelihood of collision.*

True Object Causing Detection	Successfully Tracked	Likelihood of Collision
----------------------------------	----------------------	-------------------------

Sun reflection on blade	Y	Strong
Sun reflection on blade	Y	Strong
Side of building	N	Moderate
Lamp post	N	Strong
Building roof	N	Strong
Turbine tower shadow	N	Strong
Roof, cars, lamp post	Y	Strong
Building roof	N	Strong
Building roof	N	Strong



Figure 5-18 *Different detected objects incorrectly tracked by the primary tracking system. The sequence of these objects across the FOV caused them to be linked together as one target. The varying objects within the bounding box likely caused the filter to reject this as a repeating FP.*

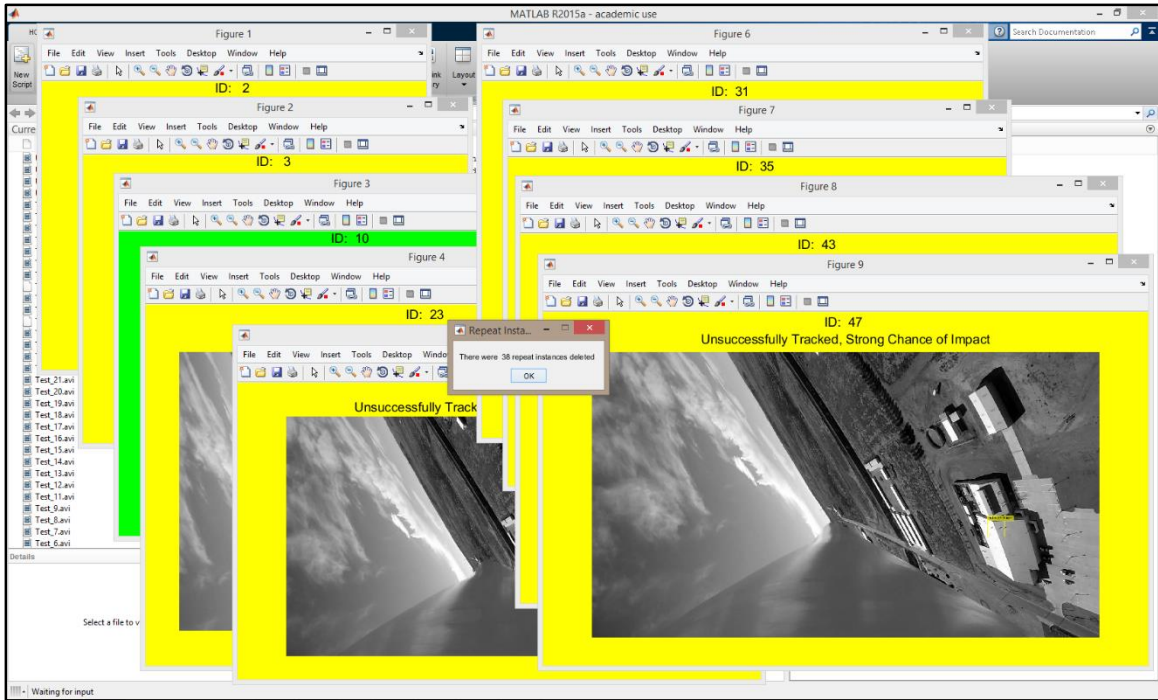


Figure 5-19 Results seen by the user after program operation. A pop-up window reports the number of instances deleted by the spatio-temporal filter.

The IDs of all nine of the presented FPs were reported to the system, and the detection system was updated using the same parameters as the original cascade detector.

Trial 2

Using the refreshed detector, the program was run once more with the same video. The number of overall FPs was 17, with 7 repeating instances removed (approximately 41%). An over 60% reduction in overall FPs occurred between the first and second trial.

Table 5-3 Trial 2 results, outlining the cause of the detection, success of tracking, and likelihood of collision.

True Object Causing Detection	Successfully Tracked	Likelihood of Collision
Sun	Y	Strong
Lamp post	Y	Strong

Lamp post	N	Strong
Lamp post	Y	Strong
Lamp post, car	Y	Strong
Lamp post	Y	Strong
Lamp post	Y	Strong
Sun	Y	Strong
Sun reflection on blade	N	Strong
Lamp post	N	Strong

The most frequent FP detection is caused by the lamp post, which comprised a significant portion of the FP detections presented to the user. After trial 2, the detection system was retrained using updated negative image set, and the same parameters specified for the original cascade detector.

Trial 3

Using the final refreshed detection algorithm, a third run of the program was made. This video segment contained the murrelet simulated impact on the turbine blade for a TP instance. There were 12 FPs overall, 9 of which (75%) were successfully removed by the spatiotemporal filter. Over a 25% reduction in FPs occurred between trials 2 and 3.

Table 5-4 Trial 3 results, outlining the cause of the detection, success of tracking, and likelihood of collision.

True Object Causing Detection	Successfully Tracked	Likelihood of Collision
Lamp post	Y	Strong
Murrelet	Y	Strong
Lamp post	N	Strong
Sun reflection on blade	N	Strong

During the tracking process for the murrelet, the tip of the blade was detected after the bird left the image plane, causing an incorrect assignment of this FP to the bird's trajectory.

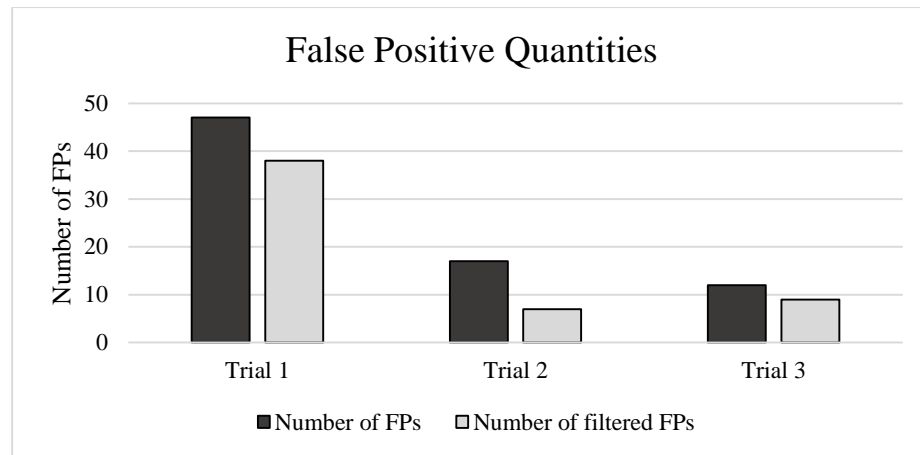


Figure 5-20 *False positive quantities, including total and filtered.* Note that the filtering no longer detects any repeating false positive instances after the first retraining of the algorithm.

Discussion

The structure of the program appeared to provide a strong supporting framework for the detection and tracking algorithms. High quantities of repeating instances caused by stationary objects were successfully removed using the spatiotemporal filter. The wind turbine blade, structures on the ground, and light blooms caused by the sun were large contributors to FP instances. A high quantity of “strong” collision likelihood ratings may be attributed to the locations of the FP detections, incorrect tracking, and incorrect blade thresholding. The blade thresholding function appeared to be falsely triggered during frames where the sun was causing light blooms and reflecting off the blade. A possible solution to the incorrect thresholding would be applying more stringent parameters to the low intensity regions seen within the blade ROI, such as a tighter size range or threshold value.

These trials suggest that the updating method for the detection algorithm may work as intended, however significantly more testing will be needed for further validation. This method of strengthening the detection algorithm against problematic objects in the FOV will be critical to the future application of this program. Further testing including extended length (>1 hour) true footage from bird interactions and flybys past an on-blade camera should be conducted, in order to further evaluate the performance of this program.

Speed Testing

Operational testing used an AMD FX-8350 Vishera 8-core (4.0GHz) processor with 16 gigabytes of RAM, and a detection algorithm with a false alarm rate of 0.4, HOG features, 19 cascade stages, and a merge threshold of 12. The third trial during operational testing required 2116.2 seconds to complete, with 1641.1 seconds of the total time allocated to the cascade object detector. In other words, the time for processing the video was greater than 63 times the length of the video. In order to obtain a realistic estimation of processing time using less data intensive video, and evaluate the possibility of real-time detection, the field test rotational footage from the hardware validation section in Chapter 4 was inputted to the program (38 seconds in length). This footage was captured with the Sentech STC-N632 camera. The merge threshold was maintained at 12 with the same detector used for trial 3 in operational testing. There were 13 detections in total, with 8 removed by the spatiotemporal filter. The time for processing the video was 225.4 seconds, with 150.2 seconds allocated to the cascade object detector. The processing time was nearly 6 times greater than the length of the footage. The reduction in processing time was an order of magnitude less than that of the GoPro footage. Figure 5-21 outlines the functions of the program consuming the greatest amount of time.










Profile Summary				
Generated 13-Feb-2016 21:42:26 using cpu time.				
<u>Function Name</u>	<u>Calls</u>	<u>Total Time</u>	<u>Self Time*</u>	Total Time Plot (dark band = self time)
...tor>CascadeObjectDetector.stepImpl	1170	150.207 s	148.374 s	
VideoReader.read	1894	21.969 s	20.884 s	
OutputStream>OutputStream.writePackets	1170	12.092 s	11.436 s	
blobCheck	1170	11.290 s	9.827 s	
uiwait	2	10.690 s	6.947 s	
imapplymatrixc (MEX-file)	1229	4.778 s	4.778 s	
FindPoints_rev5	1	225.410 s	3.927 s	
...VideoWriter>VideoWriter.writeVideo	1170	16.019 s	1.928 s	
...cadeObjectDetector.convertToGrayscale	1170	1.730 s	1.696 s	

Figure 5-21 Processing time for the overall program. These results are from 0:38 Sentech STC-N632 footage.

The length of time for processing may be reduced when the program architecture is transferred to a different coding language. The results outline here do not preclude the potential of future versions of the program operating in real-time.

Conclusions

The program outlined and tested in this chapter is ready for large dataset testing, to validate its use for on-blade camera avian, and in the future, bat interaction sensing applications. The cascade classifier will require ground truth testing for complete validation before use in an operational environment. Adaptation of the cascade object detector algorithm for bat targets would be a simple addition. Improvements in the speed of operation, and detection system accuracy should be investigated for the next version of the program. Testing using rotational footage including avian targets would give further insight to the detection, tracking, collision sensing, and filtering capabilities of this program.

With the highly dynamic background present in on-blade camera applications, object recognition is a logical detection choice. The ability of repeating FP elements to be added to the negative image set for training provides a method for strengthening the detection system against mass quantities of FP instances. Using all of the information extracted by thresholding the blade face, target trajectory, and for collision sensing The GUI developed provides a platform for the testing and tuning of cascade object detection and primary tracking system. Rapidly loading video and altering settings provides a quick method for validating newly trained cascade detectors in combination with tracking.

While speed testing has indicated that a real-time system is not out of the question for future versions of the program, the framework developed in this chapter would need significant revision to operate in real time. The major benefit of a real-time system is the ability of the on-blade camera to track avian and bat interactions without triggering from other sensors. Without a real-time running constraint, a less powerful processor may be used for the central computing unit, and more computationally expensive computer vision techniques can be utilized.

6. Blade Tracking

The purpose of the blade tracking algorithm is to determine the tip position of the wind turbine blade during operation. Constantly monitoring the deflection of the blade allows for operators to quickly shut down the blade if deflections exceed specified boundaries. This chapter outlines the program and testing performed to validate the accuracy and precision of an early version of blade tracking software.

Introduction

For the purposes of this program, the wind turbine blade was simplified as a cantilever beam. With a cantilever beam setup, where the root of the turbine blade is fixed, the tip will be the point of the blade seeing maximum displacement during operation. By tracking the blade tip, operators will be able to monitor maximum deflection, and have the ability to shut down the turbine if limits are exceeded.

The small displacements of the blade tip in comparison with the length of the blade should provide manageable feature shifts between frames. By knowing the length of the blade and behavior during tip deflection, vertical displacements on the image plane can allow for inference about the true tip position.

Success of the blade tip tracking program is based on its reliability, resolution, accuracy, precision, and ability to run in real-time. For maximum reliability during operation, future versions of the program should be able to refresh feature points if lost. Additionally, constraints on horizontal point movement could help indicate when feature points have lost their lock, since significant horizontal motion of the blade tip with respect to the blade-mounted camera is not expected during operation.

Program Structure

The MATLAB code produced for blade tracking represents an early stage of the software, meant to test the accuracy and precision of the selected method for this application. Components such as point regeneration are to be included in later revisions.

The first step in tracking the blade tip, is locking onto feature points. Subjective evaluation demonstrated that the `detectMinEigenFeatures` algorithm [45] in MATLAB Computer Vision System Toolbox worked well for this purpose. Ideally, the face of the turbine blade tapers to an approximate point at the tip, which provides a corner for tracking. Depending on the performance, and light variation of footage, true applications may require some type of marker near the blade tip for strong point tracking. In order to reduce the search area for strong feature points, the program prompts the user to select a region around the tip of the blade. This also helps to prevent unwanted background objects from being tracked.

The feature tracking algorithm [44] used for secondary tracking in Chapter 5 was chosen for this application. Smaller movements between frames, and a high contrast corner (the blade tip or marker) provide beneficial elements that should contribute to successful tracking. The strongest 8 points from the feature point selection are used. Points which are not successfully tracked between frames are removed during operation based on a binary confidence score output by the tracking algorithm. The foremost purpose of removing these points is preventing interpretation of incorrect data pertaining to the blade tip position.

Before deflection occurs, a horizontal reference line is created using the maximum point on the blade. The displacement of the maximum point on the blade at a given time is compared to this reference line, which yields the vertical displacement on the image plane in pixels. Using the known length of the blade (assumed linear during deflection), and assuming the out of plane distance

between the blade tip and camera lens remains constant, the tip deflection can be easily approximated. The MATLAB code for the blade tracking program may be found in Appendix H.

Testing

For determining the validity of this method, the accuracy must be tested with a range of displacements. The behavior of the program when tracking the tip of a beam during bending was unknown, so an initial validation was performed using a 1 meter long balsa wood plank (with a square end) mounted beneath the camera. During rapid and large beam deflection the corner points were successfully tracked. Torsion of the plank yielded multiple lost points.



Figure 6-1 *Point tracking on balsa wood beam.*

For testing accuracy, a dark backdrop with little to no intensity gradient was used. The balsa wood plank used for validation was securely mounted beneath the camera. Digital calipers, accurate to ± 0.0254 millimeters was clamped beneath the tip of the plank. The depth measurement feature on the calipers provided a method of displacement for the plank tip. The calipers are manually extended between each value. The system was adjusted so that the plank was level when the calipers were at zero. Fluorescent lights were shut off to prevent the noise they produce, and LED lighting kept the region near the blade tip illuminated. To test the difference in accuracy between small and large displacements from the zero reference line, the beam was moved to 2, 6, 10, and 20 millimeters (approximately 0.2% to 2% of the overall blade length). The test was repeated 20 times, with lighting and the zero reference line held approximately constant. When a displacement value

is reached, the system is left untouched to reduce vibrations and operator induced error. Values for the displacement calculated by the point tracking algorithm will be obtained by averaging the displacements across a small timespan, when the system is left alone.

Results

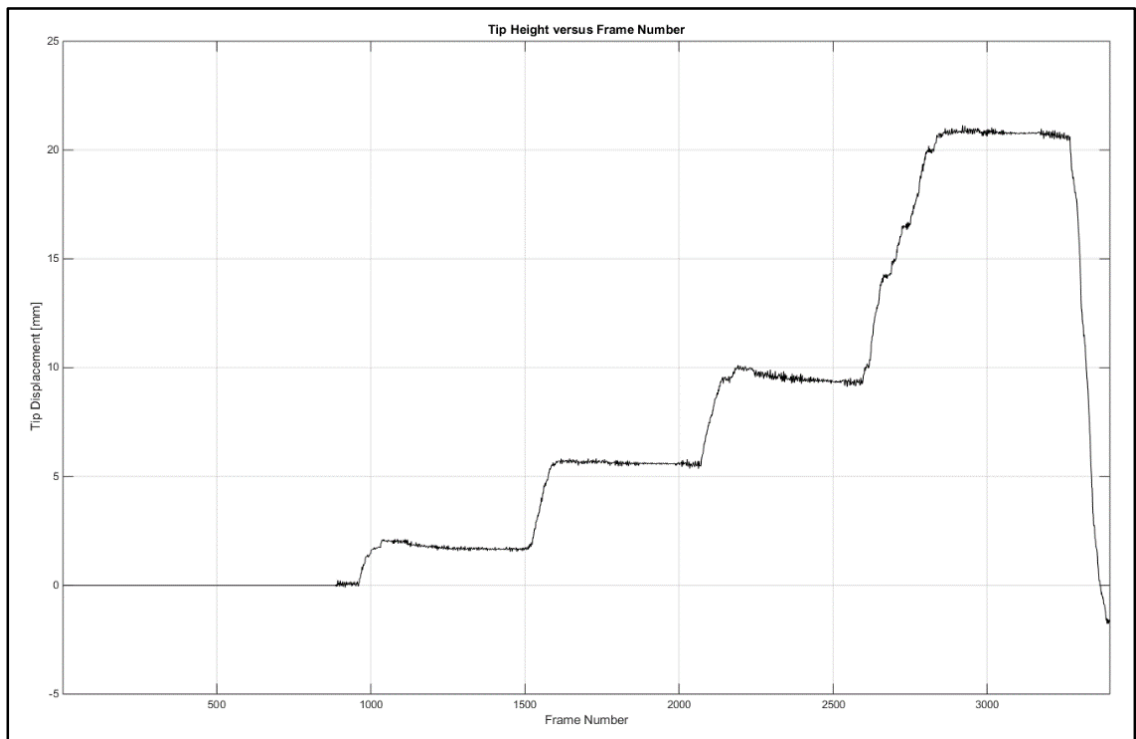


Figure 6-2 *Tip displacement in millimeters versus frame number.* This plot provides a visual for the displacement of the beam tip across time. Even when the tip has reached its desired value, there is still fluctuation in the tracked displacement.

The standard deviation of the tracked blade tip displacement increased at greater displacements. The mean values for 1, 2, 6, and 10 millimeter displacements were less than the caliper measured values, while the mean for the 20 millimeter displacement was greater. The maximum error between the caliper measurement and point tracking came from the 2 millimeter displacement, which yielded a 12% difference. The 20 millimeter displacement created a 3.9% disparity between the caliper measurement and point tracking data. The P value from t tests performed at each

displacement suggested a statistically significant difference between the means of the caliper measured values, and those measured by the point tracking program. The test values and statistical analysis may be found in Appendix I.

Table 6-1 *Comparative values between the caliper measured displacement and the mean (of 20 tests) displacement value from the point tracking program.*

Caliper Measured Displacement [mm]	Mean of Tracking Program [mm]	Standard Deviation of Tracking Program [mm]
2.00	1.76	0.16
6.00	5.65	0.26
10.00	9.64	0.35
20.00	20.78	0.48

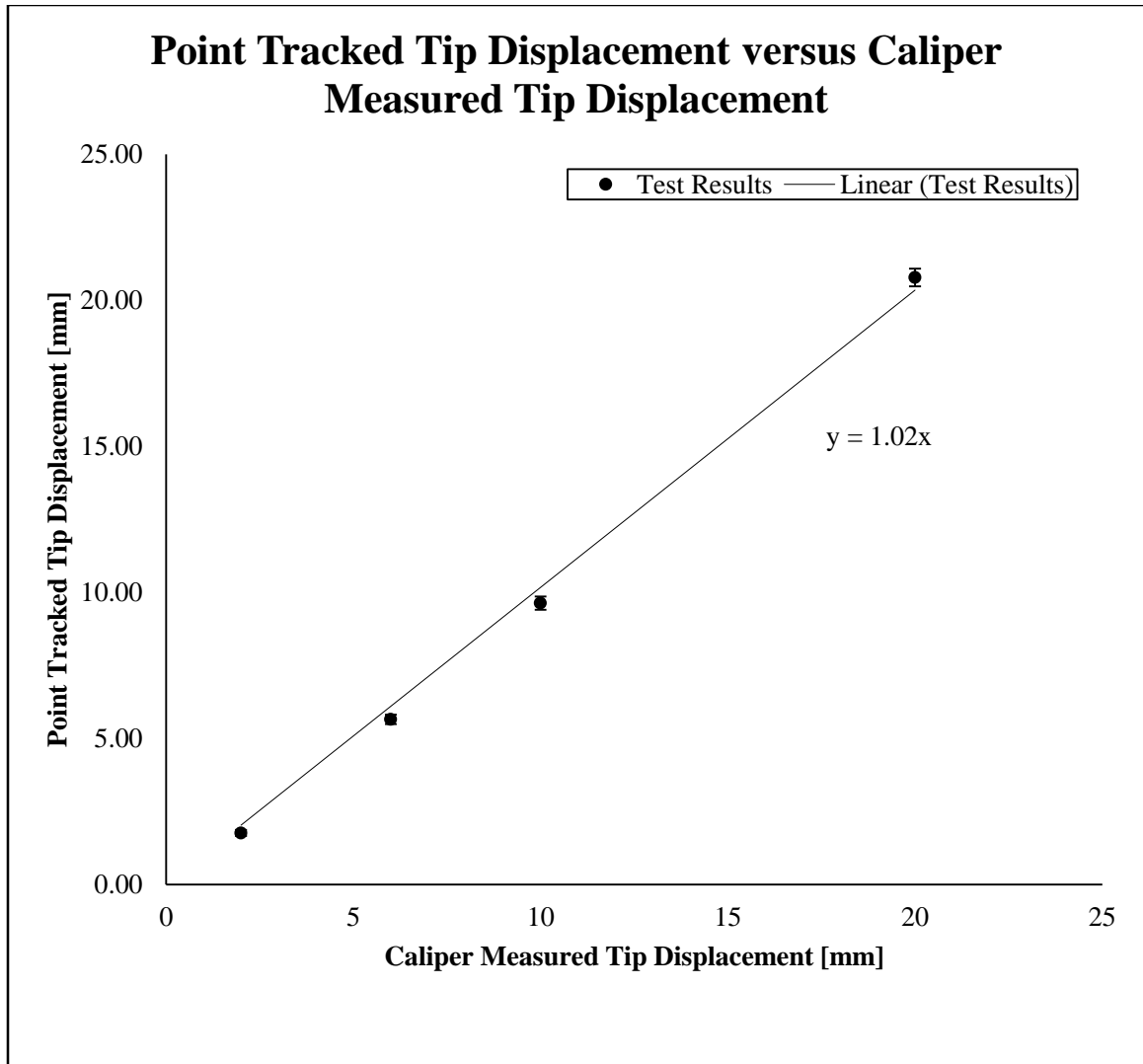


Figure 6-3 *Measured Tip Displacement versus True Tip Displacement.* Confidence bounds for each point tracked value are provided via error bars.

Using linear regression, the line of best fit for the data had a slope of 1.02.

Discussion

The differences between the caliper measured values and mean of tip tracking data may partially stem from errors within the experiment. If the camera was tilted even a small amount during deflection, it would cause error in the calculated displacement. The highest feature points on the blade may not exist exactly at the tip of the blade. This misalignment would cause error as the

perceived displacement is below that of the maximum of the blade tip. Additionally, videos of the point tracking process demonstrate shifting of the points during displacement, which may create systematic error.

The noise seen by the tracking algorithm during periods where the system was left untouched may stem from small vibrations, static of video transmission, or changes in illumination. Point drifting, where points move away from the original feature being tracked, was observed at small levels during each beam test video. Similar to the reflective strips used by [10], a possible solution to this point drifting would be adding a checkerboard sticker near the blade tip (as seen in figure 6-4), which would provide a stronger feature for tracking. Point drifting also causes hysteresis, as the point shifts to a new feature it is permanently moved to that location, unless some factor causes it to revert to its original position.

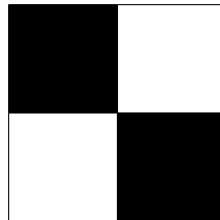


Figure 6-4 *A checkerboard pattern, frequently used for camera calibration and computer vision applications. This checkerboard applied at the blade tip would provide strong features for point tracking.*

Differences between the mean of the tracking data and the caliper values did not scale exactly with displacement size. This suggests that the extreme displacements that future iterations of the program will work to detect may have a higher comparative level of accuracy than small regular displacements. Testing provided some level of apparent systematic error, with mean values from point tracking consistently greater or less than the caliper values depending on the tip displacement. Overall the accuracy of this initial system appears to be at a useable level, and validates the system for further testing.

7. Future Developments

To take the on-blade camera system to a level that is ready for use on a widespread scale, further testing and validation will be required. The purpose of this section is to outline some of the imperative next steps in developing this system.

Hardware

The limited number of frames which a bird or bat target may occupy due to the fast rotation rate of the turbine blade puts an enormous emphasis on successful detection and tracking. A possible solution to this necessity is to select a higher framerate camera. To accommodate this, a more expensive and likely larger camera and transmitter would be needed. Low latency transmission of high definition video with high framerates has benefitted from the growing hobby of first person view (FPV) remote controlled quad rotor unmanned aerial vehicles (quadcopters). A recently released wireless video transmission and reception system referred to as Connex by Amimon, offers 1080p digital video transmission and reception at up to 60 frames per second with minimal delay [49]. Systems like this are becoming more prevalent, however the cost is still high in comparison to the traditional analog wireless video transmission options. The higher framerate could provide a significant improvement in the secondary tracking algorithm, and would offer an increased opportunities for detection by the object detector.

Sensor System Integration

As described in Chapter 3, the components comprising the avian and bat collision detection system work in unison to detect bird collisions [9]. Experimentation with the LabView framework developed in [9] to incorporate real-time or post-processed data from the on-blade camera software will be needed. Incorporating the collision likelihood rating given by the program in Chapter 5 will

offer the sensor system an additional collision detection method. Storage methods for the events detected by the on-blade camera will need to be researched for the central computing unit.

Camera Casing

As the current camera casing was designed as an early proof of concept, and to provide a platform for testing. There are several additions which must be made to future versions of the camera case. One of the foremost items for preparing this case for operational use is waterproofing. The following components are imperative for preventing moisture penetration:

- Sealed lens on the camera casing
- Gasket between upper shell and base plate
- Desiccant to prevent condensation on the lens

Incorporating the vibrational sensors (contact microphones and accelerometers) that would normally be placed on the wind turbine blade into the camera casing would require expansion of the case volume. Additionally, if using the same power source, a higher current draw would be placed on the battery. Despite these drawbacks, it is attractive to have the on-blade camera and vibration sensors combined into a single package for installation and maintenance.

The use of adhesives for mounting the camera casing to the turbine blade is a minimally invasive method. Determining the adhesion method between the base plate and the root of the turbine blade will require research about adhesive choice, time required for adhesion, maintaining mounting through varying weather conditions, and removal from the blade with no damage to the surface of the blade. A pull strip at the edge of the base plate could provide an easy removal method for adhered casing.

The utilization of a battery for an on-blade power source, and consequently removal of the need for a slip ring provides an immense reduction in cost of implementation. Despite this, there is a quantity

of supporting hardware needed to make this option truly viable. One such component is solar power: a natural choice given the green energy aspect of wind power, a photovoltaic cell may provide sufficient power to maintain charge within a battery. Lithium ion (Li-Ion) batteries do not have the inherent “memory effect” of some rechargeable batteries [50]. This aspect makes Li-Ion batteries an ideal choice for recharging via solar power. A 3 cell lithium pack would provide a nominal voltage of 11.1 volts, which should be sufficient for 12 volt electronics. To support this charging method, a high and low voltage cutoff board will be necessary to prevent damage to the battery from overcharging or dipping below minimum cell voltage. These circuit boards are cheap and widely available. A light sensor exposed through the case wall will provide a shutoff method when lighting become too poor for the on-blade camera to successfully detect birds or bats. Finally, a more efficient voltage drop method than a resistor in series should be implemented for power conservation, and excess heat reduction.

Safety and ease of maintenance are two key aspects this camera casing must include. To warn wind farm operators of insufficient power, an LED indicator could be implemented to provide a quick and simple method for checking the status of the on-blade camera power supply. Lithium based batteries have the potential to combust when improperly manufactured or mishandled. Extensive testing and checking of any battery cells should be made before installation. As a method of protection for the wind turbine blade and moreover its operators, flame retardant material should encase the battery, with porting to allow gases to vent without rapid expansion.

Avian Program

The results from program speed testing in Chapter 5 suggest that a real-time system is feasible, however significant development (likely outside of the MATLAB environment) will be necessary to bring the program to this state. Implementation of this program in C++ or another programming language may speed up the processing time. The filtering and classification methods used for the program would need to be changed to fit a real-time program. One possibility for FP filtering and

classification, is the storing of results in temporary storage, and at set time intervals using the stored data to perform these actions. Another processing issue not addressed by the current program, is the need for deinterlacing, or other preparation of footage before analysis. This would also need to be incorporated into a real-time system.

The presentation and storage of results was created to simulate a possible method for delivering results to the user, and for an easy evaluation method for the performance of the program. When integrating with the overall sensor system, these results should instead to pass to the central computing unit to be combined with input from the entire sensor suite. Vision based standard and IR cameras mounted on the nacelle could be used to cross check bird and bat flyby information obtained from the on-blade camera system.

The classification system for the likelihood of collision is based on a subjective scheme. As advancements are made in the computer vision aspects of the program, a more quantitative approach would likely yield better results than those presented here. A paramount quality of the classification system is utilizing all of the information collected during the video analysis process.

Work done in [39] investigated the use of MATLAB cascade object detection for the use of bird detection, and found Haar features to produce high accuracy rates. The work in [40] has confirmed that Haar-like features are high performing for low resolution bird detection. The detection of birds and bats near wind turbines will generally be concerned with targets that are at long distances from the camera, leading to poor resolution and minimal texture. The use of the created rotation rig in an outdoor setting with large quantities of birds may provide useful footage for ground truth testing for this application. An advisable later step is the implementation of the proposed on-blade camera system on a wind farm to obtain bird flyby or collision footage. The lighting changes, wind turbine structures, and camera rotation are unique to the on-blade environment, and detection in this setting needs to be evaluated using true footage.

The point tracking method for secondary tracking has been qualitatively evaluated. This seems to be a plausible option as a backup tracking system, due to the relatively low computing cost and potential for tracking single detection targets. It is apparent that the displacement and “quality” of the feature points are dictating factors for the success of tracking. Further validation of this method is needed, in order to understand its reliability. A secondary tracking system can add a large amount of value, as determining the path of the bird near the wind turbine blade can provide key information as to the existence of a collision. For real-time operation, this additional method will need to be closely evaluated to determine if the extra computational time is worth the tradeoff of secondary tracking.

Finally, the program must be extended to detect and track bats. Bat flight does not only occur during the night, meaning that the vision based camera of the on-blade system should be prepared to detect bat interactions with the turbine blade. Due to the length of operating time for the cascade object detector, adding an additional detector may prove impractical for real-time monitoring. For triggered analysis (using ring buffering) the addition of a second cascade object detector should not pose any large issues beyond an increase in computation time and likely an increase in FP instances.

Blade Track

The program and testing presented in Chapter 6 provided a preliminary evaluation of the accuracy and precision of a feature tracking method for monitoring flapwise blade deflection. Key operational related aspects need to be explored further in order to completely validate this method. One of the foremost aspects is the regeneration of points. Even from the controlled tests performed to evaluate accuracy, some amount of point drifting occurred. By setting maximum bounds and rates for point movement, the program should be able to automatically regenerate points which have lost a lock on their feature. As an additional solution to this issue, a checkerboard patterned sticker could be placed near the tip of the blade; this would provide a strong feature to track, and ease the process of point regeneration.

Conclusion

This thesis has contributed to the development of an on-blade camera system for the purpose of monitoring bird and bat impacts. This on-blade system, which will operate cooperatively with a greater sensor system, uses computer vision techniques to detect the presence of avian and bat interactions with the wind turbine blade. Specific contributions include the validation of hardware, design of on-blade casing, and software development for both bird interactions and turbine blade deflection.

The proposed avian interaction and collision sensing software architecture is ready for further testing, to validate its use in an operational wind turbine environment. Ground truth testing using rotational footage with avian targets will provide key information about the accuracy and FP rate of the cascade object detector. The detection system should be extended to bats as well, in order to obtain a complete understanding of the system behavior. With the unique rotating setting of this application, the spatiotemporal filtering and retraining technique provide a method for reducing the overall number of FPs detected and stored. Through testing, the Kanade Lucas Tomasi feature tracking algorithm appears to be a viable candidate for blade tip tracking in terms of its accuracy and precision.

Automation of monitoring bird and bat impacts with wind turbines will save time and cost for wind farms seeking to comply with national standards for avian and bat deaths. The improvement in monitoring accuracy from the more prominent manual methods will ensure that governing bodies will be able to protect species of interest. Ultimately automated techniques such as the one explored here may provide a platform for better integrating the green energy solution of wind power without endangering avian and bat populations.

Work Cited

- [1] American Wind Energy Association. (2015, Apr. 30). *Wind Energy Accelerates Record Growth in First Quarter*. AWEA [Online]. Available: <http://www.awea.org/MediaCenter/pressrelease.aspx?ItemNumber=7534> (Accessed: Nov. 5, 2015).
- [2] K. S. Smallwood, "Estimating wind turbine-caused bird mortality," *The Journal of Wildlife Management*, vol. 71, no. 8, pp. 2781-2791, Nov. 2007. [Online]. Available: <http://dx.doi.org/10.2193/2007-006> (Accessed: Sept. 16, 2015).
- [3] "Windfarms and birds: calculating a theoretical collision risk assuming no avoiding action," Scottish Natural Heritage, Inverness, Scotland, 2000. [Online]. Available: <http://www.snh.gov.uk/docs/C205425.pdf> (Accessed: Nov. 30, 2015).
- [4] C. L. Hull, S. C. Muir, "Behavior and turbine avoidance rates of eagles at two wind farms in Tasmania Australia," *Wildlife Society Bulletin*, vol. 37, no. 1, pp. 49-58, Mar. 2013. [Online]. Available: <http://dx.doi.org/10.1002/wsb.254> (Accessed: Nov. 15, 2015).
- [5] DeTect. n.d. *Bird and Bat Radar Systems*. [Online]. Available: <http://www.detect-inc.com/avian.html> (Accessed: Nov. 30, 2015).
- [6] DTBird. (2015, Oct.). *Bird Monitoring and Reduction of Collision Risk with Wind Turbines*. [Online]. Available: <http://www.dtbird.com/images/Downloads/DTBird-Brochure-OCT15-web.pdf> (Accessed: Nov. 30, 2015).
- [7] J. P. Verhoef, C. A. Westra, H. Korterink, A. Curvers, "WT-Bird a novel bird impact detection system," ECN Research Centre of the Netherlands, Petten, The Netherlands, n.d. [Online]. Available: <https://www.ecn.nl/docs/library/report/2002/rx02055.pdf> (Accessed: Nov. 30, 2015).
- [8] L. Spiegel, K. Birkinshaw, L. T. Hope, M. Krebs, B. B. Blevins, "Development of a cost-effective system to monitor wind turbines for bird and bat collisions, phase I: sensor

- system feasibility study,” EDM International, Inc., Fort Collins, CO, United States, No. 500-01-032, 2007. [Online]. Available: <http://www.energy.ca.gov/2007publications/CEC-500-2007-004/CEC-500-2007-004.PDF> (Accessed: Nov. 30, 2015).
- [9] J. Flowers, R. Albertani, T. Harrison, B. Polagye, R. M. Suryan, “Design and initial component tests of an integrated avian and bat collision detection system for offshore wind turbines,” in *2nd Marine Energy Technology Symposium*, Seattle, WA, United States, 2014. [Online]. Available: <http://depts.washington.edu/nnmrec/docs/METS%20paper2014v7.pdf> (Accessed: Apr. 12, 2015).
- [10] L. J. Fingersh, “Optical blade position tracking system test,” National Renewable Energy Laboratory (NREL), Golden, CO, United States, TP-500-39253, 2006. [Online]. Available: <http://www.nrel.gov/docs/fy06osti/39253.pdf> (Accessed Dec. 20, 2015).
- [11] X. Fu, L. He, H. Qiu, “MEMS gyroscope sensors for wind turbine blade tip deflection measurement,” in *Instrumentation and Measurement Technology Conference (I2MTC)*, Minneapolis, MN, 2013, pp. 1708-1712. [Online]. Available: <http://dx.doi.org/10.1109/I2MTC.2013.6555706> (Accessed: Oct. 12, 2015).
- [12] H. C. Kim, P. Giri, J. R. Lee, “A real-time deflection monitoring system for wind turbine blades using a built-in laser displacement sensor,” in *6th European Workshop on Structural Health Monitoring*, Dresden, Germany, 2012. [Online]. Available: <http://www.ndt.net/article/ewshm2012/papers/we2b2.pdf> (Accessed: Oct. 10, 2015).
- [13] European Wind Energy Association. n.d. *Wind Energy’s Frequently Asked Questions (FAQ)*. EWEA [Online]. Available: <http://www.ewea.org/wind-energy-basics/faq/> (Accessed: Nov 5, 2015).

- [14] U.S. Fish and Wildlife Service. (2015, Apr. 14). *Endangered Species Permits*. [Online]. Available: <http://www.fws.gov/Midwest/endangered/permits/hcp/index.html> (Accessed: Dec. 21, 2015).
- [15] J. Everaert, "Wind turbines and birds in Flanders: preliminary study results and recommendations," *Natuur Oriolus*, no. 69(4), pp. 145-155, n.d. [Online]. Available: <https://www.fws.gov/Midwest/wind/references/Flandersmortstudy.pdf> (Accessed: Dec. 15, 2015).
- [16] M. Morrison et al, "Wind turbine interactions with birds, bats, and their habitats: a summary of research results and priority questions," Department of Energy, United States, 2010. [Online]. Available: https://www1.eere.energy.gov/wind/pdfs/birds_and_bats_fact_sheet.pdf (Accessed: Jan. 29, 2016).
- [17] J. Zhang, Q. Xu, X. Cao, P. Yan, X. Li, "Hierarchical incorporation of shape and shape dynamics for flying bird detection," *Neurocomputing*, vol. 131, pp. 179-190, May 2014. [Online]. Available: <http://dx.doi.org/10.1016/j.neucom.2013.10.026> (Accessed: Nov. 24, 2015).
- [18] W. W. Verstraeten *et al.*, "Webcams for Bird Detection and Monitoring: A Demonstration Study," *Sensors*, vol. 10, pp. 3480-3503, Apr. 2010. [Online]. Available: <http://dx.doi.org/10.3390/s100403480> (Accessed: Nov. 24, 2015).
- [19] W. Li, D. Song, "Automatic Bird Species Detection from Crowd Sourced Videos," *Automation Science and Engineering, IEEE Transactions*, vol. 11, no. 2, pp. 348-358, Apr. 2014. [Online]. Available: <http://dx.doi.org/10.1109/TASE.2013.2247397> (Accessed: Nov. 24, 2015).
- [20] S. Campbell, "Annual blade failures estimated at around 3,800," *Wind Power Monthly*, May 2015. [Online]. Available:

- <http://www.windpowermonthly.com/article/1347145/annual-blade-failures-estimated-around-3800> (Accessed: Nov. 22, 2015).
- [21] M. Malkin, A. Byrne, D. Griffin, “Does the wind industry have a blade problem?,” *North American Wind Power*, vol 12, no. 4, May 2015. [Online]. Available: http://www.nawindpower.com/issues/NAW1505/FEAT_02_Does-The-Wind-Industry-Have-A-Blade-Problem.html (Accessed: Nov. 22, 2015).
- [22] *STC-N632/N632CS/N632L STC-P632/P632CS/P632L Product Specification*, Sentech America, Inc., Carrollton, TX, pp. 1-10. [Online]. Available: http://downloads.i-sentech.com/dl_documents/spec_STC-N632series_en_v1.0.pdf (Accessed: June 10, 2015).
- [23] Nicky Pages’ Digital Solutions. n.d. *NTSC, PAL & Interlace Explained*. [Online]. Available: <http://nickyguides.digital-digest.com/interlace.htm> (Accessed: Dec. 15, 2015).
- [24] Bambooav. n.d. *Information About Interlaced and Progressive Scan Signals*. [Online]. Available: <http://www.bambooav.com/information-about-interlaced-and-progressive-scan-signals.html> (Accessed: Dec. 15, 2015).
- [25] P. Bourke. (2013, Apr). *Field of View and Focal Length*. Paul Bourke [Online]. Available: <http://paulbourke.net/miscellaneous/lens/> (Accessed: Dec. 15, 2015).
- [26] J. Quilter, “GE to build 10MW turbine testing facility,” *Wind Power Monthly*, Mar. 2013. [Online]. Available: <http://www.windpowermonthly.com/article/1174697/ge-build-10mw-turbine-testing-facility> (Accessed: Dec. 15, 2015).
- [27] *SDX-26 Special 2.4 GHz Audio/Video Transmitter with 4 Channels*, RF-Links. [Online]. Available: <http://rf-links.com/newsite/pdf/sdx26.pdf> (Accessed: June 10, 2015).
- [28] *VRX-24l Audio/Video Receiver 2.4 GHz*, RF-Links. [Online]. Available: <http://www.rf-links.com/newsite/pdf/vrx24l.pdf> (Accessed: June 10, 2015).

- [29] MathWorks. n.d. *vision.Deinterlacer System Object*. [Online]. Available: <http://www.mathworks.com/help/vision/ref/vision.deinterlacer-class.html> (Accessed: Feb. 5, 2016).
- [30] R. Collins. CSE 598C, Class Lecture, Topic: “Towards crowd scene analysis.” Department of Computer Science and Engineering, Pennsylvania State University, State College, PA, n.d. [Online]. Available: <http://www.cse.psu.edu/~rtc12/CSE598C/MotionDetection.pdf> (Accessed: Feb. 2, 2016).
- [31] A. Singh, “State of the Art”, in *Optic Flow Computation*, Los Alamitos, CA, United States: IEEE Computer Society Press, 1991.
- [32] Y. Wu. EECS 432, Class Notes, Topic: “Optical flow and motion analysis,” Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, n.d. [Online]. Available: http://www.ece.northwestern.edu/~yingwu/teaching/EECS432/Notes/optical_flow.pdf (Accessed: Dec. 15, 2015).
- [33] R. Rojas. Class Notes, Topic: “Lucas-Kanade in a nutshell,” Department of Computer Science, Free Univeristy of Berlin, Germany, n.d. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/tutorials/Lucas-Kanade2.pdf (Accessed: Dec. 15, 2015).
- [34] T. Pock. Class Lecture, Topic: “Optical flow,” Institute for Computer Graphics and Vision, Graz University of Technology, Austria, n.d. [Online]. Available: https://cw.fel.cvut.cz/wiki/_media/courses/ae4m33mpv/optical_flow_2.pdf (Accessed: Dec. 15, 2015).
- [35] MathWorks. n.d. *Train a Cascade Object Detector*. [Online]. Available: http://www.mathworks.com/help/vision/ug/train-a-cascade-object-detector.html?s_tid=gn_loc_drop&refresh=true (Accessed: June 3, 2015).

- [36] B. Raluca. Class Notes, Topic: “Histograms of oriented gradients,” Computer Science Department, Technical University of Cluj-Napoca, Romania, n.d. [Online]. Available: http://users.utcluj.ro/~raluca/prs/prs_lab_05e.pdf (Accessed: Dec. 15, 2015).
- [37] P. Viola, M. Jones, “Rapid Object Detection Using a Boosted Cascade of Simple Features,” in *Conference on Computer Vision and Pattern Recognition*, 2001. [Online]. Available: <http://dx.doi.org/10.1109/CVPR.2001.990517> (Accessed: Dec. 10, 2015).
- [38] B. S. Hanzra. (May 30, 2015). *Texture Matching Using Local Binary Patterns (LBP), OpenCV, Scikit-Learn and Python*. HanzraTech [Online]. Available: <http://hanzratech.in/2015/05/30/local-binary-patterns.html> (Accessed: Feb. 3, 2016).
- [39] E. Reyes, “A comparison of image processing techniques for bird detection,” M.S. thesis, Electrical Engineering Department, California Polytechnical State University, San Luis Obispo, 2014. [Online]. Available: <http://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=2341&context=theses> (Accessed: Feb. 3, 2016).
- [40] R. Yoshihashi, R. Kawakami, M. Lida, T. Naemura, “Construction of a bird image dataset for ecological investigations,” in *2015 IEEE International Conference on Image Processing*, Quebec City, 4248-4254. [Online]. Available: <http://dx.doi.org/10.1109/ICIP.2015.7351607> (Accessed: Jan. 12, 2016).
- [41] “Short-tailed Albatross (*Phoebastria albatrus*) threatened and endangered species,” U.S. Fish & Wildlife Service, United States, 2001 [Online]. Available: <http://www.fws.gov/alaska/fisheries/endangered/pdf/STALfactsheet.pdf> (Accessed: Dec. 29, 2015).
- [42] MathWorks. n.d. *Motion-Based Multiple Object Tracking*. [Online]. Available: http://www.mathworks.com/help/vision/examples/motion-based-multiple-object-tracking.html?s_tid=gn_loc_drop (Accessed: Dec. 20, 2015).

- [43] MathWorks. n.d. *assignDetectionsToTracks*. [Online]. Available: <http://www.mathworks.com/help/vision/ref/assigndetectionstotracks.html> (Accessed: May 10, 2015).
- [44] MathWorks. n.d. *vision.PointTracker System Object*. [Online]. Available: <http://www.mathworks.com/help/vision/ref/vision.pointtracker-class.html> (Accessed: Aug. 5, 2015).
- [45] MathWorks. n.d. *detectMinEigenFeatures*. [Online]. Available: <http://www.mathworks.com/help/vision/ref/detectmineigenfeatures.html> (Accessed: Aug. 28, 2015).
- [46] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600-612, Apr. 2004. [Online]. Available: <http://dx.doi.org/10.1109/TIP.2003.819861> (Accessed: Dec. 2, 2015).
- [47] J. Doke. n.d. *Creating a GUI With GUIDE*. MathWorks [Online]. Available: <http://www.mathworks.com/videos/creating-a-gui-with-guide-68979.html> (Accessed: July 15, 2015).
- [48] MathWorks. n.d. *vision.CascadeObjectDetector System Object*. [Online]. Available: <http://www.mathworks.com/help/vision/ref/vision.cascadeobjectdetector-class.html> (Accessed: June 1, 2015).
- [49] Amimon. n.d. *Connex Solution*. [Online]. Available: <http://connex.amimon.com/> (Accessed: Jan. 28, 2016).
- [50] I. Buchmann. (2016). *BU-204: How do Lithium Batteries Work?* Battery University [Online]. Available: http://batteryuniversity.com/learn/article/lithium_based_batteries (Accessed: Jan. 28, 2016).
- [51] P. H. Madsen. n.d. *Introduction to the IEC 61400-1 Standard*. Danish Wind Industry Association [Online]. Available:

- http://www.windpower.org/download/461/introduction_to_the_iecpdf (Accessed: Dec. 15, 2015).
- [52] General Electric. (2015). *Wind Turbines*. GE Renewable Energy [Online]. Available: <https://renewables.gepower.com/wind-energy/turbines.html> (Accessed: June 1, 2015).
- [53] HIS Engineering 360. (2016). *2.85-100 and 2.85-103 Wind Turbines*. [Online]. Available: <http://datasheets.globalspec.com/ds/2797/GEEnergy/D46FFBD2-767E-4B8D-AC79-FADBA3A0CE83> (Accessed: June 1, 2015).
- [54] Vestas. (2014). *Vestas*. [Online]. Available: <https://www.vestas.com/> (Accessed: June 1, 2015).
- [55] Siemens. (2016). *Wind Power Solutions for Offshore, Onshore, and Service Projects*. [Online]. Available: <http://www.energy.siemens.com/hq/en/renewable-energy/wind-power/> (Accessed: June 1, 2015).
- [56] I. Analyst. *Masking Out Image Area Using Binary Mask*. MathWorks [Online]. Available: <http://www.mathworks.com/matlabcentral/answers/38547-masking-out-image-area-using-binary-mask> (Accessed: July 23, 2015).
- [57] MathWorks. n.d. *Face Detection and Tracking Using the KLT Algorithm*. [Online]. Available: <http://www.mathworks.com/help/vision/examples/face-detection-and-tracking-using-the-klt-algorithm.html> (Accessed: Aug. 5, 2015).
- [58] GraphPad Software. (2016). *P Value Calculator*. QuickCalcs [Online]. Available: <http://graphpad.com/quickcalcs/PValue1.cfm> (Accessed: Jan. 28, 2016).

© 2015 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Appendix A: Turbine Size Estimation

This appendix provides the MATLAB code and corresponding output, to support the turbine size estimation plot seen in Chapter 4. This code estimates the blade radius of a wind turbine given its power output and IEC class.

The IEC class rating, dealing with operational wind speed, is split into the categories of 1, 2, 3, and S [51]. An IEC rating of 1 represents the highest defined rating, while class S details that the operational wind speed is specified by the designer [51]. A power output of 10 MW and the assumption of an IEC class of 1 were entered for the MATLAB code.

MATLAB Code

```
% Wind Turbine Characterization
% Given Power, IEC Turbine Class, Find Blade Length

clear
clc
clf
EstimateP = 10; %Turbine Power [MW]
EstimateTC = 1; %Turbine Class
bladeinfGE = [1.7,51.5,3; 1.85,41.25,2; 1.85,43.5,2; 2.5,60,3;
2.75,60,3; 2.85,50,2; 2.85,51.5,2; 3.2,51.5,2; 1.6,41.25,2; 1.5,38.5,1;
1.7,50.2,3];
[a b] = size(bladeinfGE); %GE Turbine information (see proceeding
citation list) %MW, Blade L, IEC Class
bladeinfVESTAS = [1.7,50.2,3; 2,55,3; 2,50,2; 1.8,50,3; 2,50,3;
1.8,45,2; 2,45,3; 3.3,63,3; 3.3,58.5,2; 3.3,56,2; 3.3,52.5,1;
8.0,82,1];
[c d] = size(bladeinfVESTAS); %Vestas Turbine information (see
proceeding citation list)
bladeinfSIEMENS = [2.3,50.5,2; 2.3,54,2; 3,50.5,1; 3.2,50.5,1; 3,54,1;
3.2,54,1; 3,56.5,2; 3.2,56.5,2; 3.6,60,1; 4,60,1; 4,65,1; 6,77,1];
[e f] = size(bladeinfSIEMENS); %Siemens Turbine information (see
proceeding citation list)
colors = ['r','b','k'];
leg = ['class 1','class 1 best fit','class 2','class 2 best fit','class
3','class 3 best fit'];
x = [0:0.1:EstimateP+2];
for i = 1:3
classGEfind = bladeinfGE(:,3) == i;
classGE = bladeinfGE(classGEfind,:);
classVESTASfind = bladeinfVESTAS(:,3) == i;
classVESTAS = bladeinfVESTAS(classVESTASfind,:);
classSIEMENSfind = bladeinfSIEMENS(:,3) == i;
classSIEMENS = bladeinfSIEMENS(classSIEMENSfind,:);
```

```

[o p] = size(classGE);
[g h] = size(classVESTAS);
[m n] = size(classSIEMENS);
class_sort = classGE;
    for j = 1:g
        class_sort(end+1,:) = classVESTAS(j,:);
    end
    for k = 1:m
        class_sort(end+1,:) = classSIEMENS(k,:);
    end
P = polyfit(class_sort(:,1),class_sort(:,2),1);
lines = polyval(P,x);
scatter(class_sort(:,1),class_sort(:,2),colors(i))
hold on
plot(x,lines,colors(i))
hold on
    if i == EstimateTC
        plot(EstimateP,polyval(P,EstimateP),'mX')
        estimated_radius = polyval(P,EstimateP);
        hold on
        line_y = plot([EstimateP,EstimateP],[0,estimated_radius],'--m');
        lineann_y = get(line_y,'Annotation');
        legendmanip_y = get(lineann_y,'LegendInformation');
        set(legendmanip_y,'IconDisplayStyle','off');
        hold on
        line_x =
plot([0,EstimateP],[estimated_radius,estimated_radius],'--m');
        lineAnnotate = get(line_x,'Annotation');
        legendoff = get(lineAnnotate,'LegendInformation');
        set(legendoff,'IconDisplayStyle','off');
        hold on
    end
hold on
end
if EstimateTC == 1
    legend('class 1','class 1 best fit','Estimated Blade Radius','class
2','class 2 best fit','class 3','class 3 best fit')
elseif EstimateTC == 2
    legend('class 1','class 1 best fit','class 2','class 2 best
fit','Estimated Blade Radius','class 3','class 3 best fit')
else
    legend('class 1','class 1 best fit','class 2','class 2 best
fit','class 3','class 3 best fit','Estimated Blade Radius')
end
title('Blade Radius versus Power Output for Wind Turbines')
xlabel('Power Output (MW)')
ylabel('Blade Radius (m)')
grid on
hold off
%The Vestas V164-8.0 has a hub radius of 2m, so in order to estimate
blade length of the GE 10Mw, a hub radius of 2.5m is assumed
estimated_blade = estimated_radius - 2.5;
fprintf('\nThe estimated blade length is %6.2f m\n \n \n',
estimated_blade)
fprintf('The values displayed were obtained from the websites of GE,
Vestas, and Siemens.\nThe Vestas V164-8.0 is actually a class S, but
due to being offshore has been assumed to be class 1. \n')

```

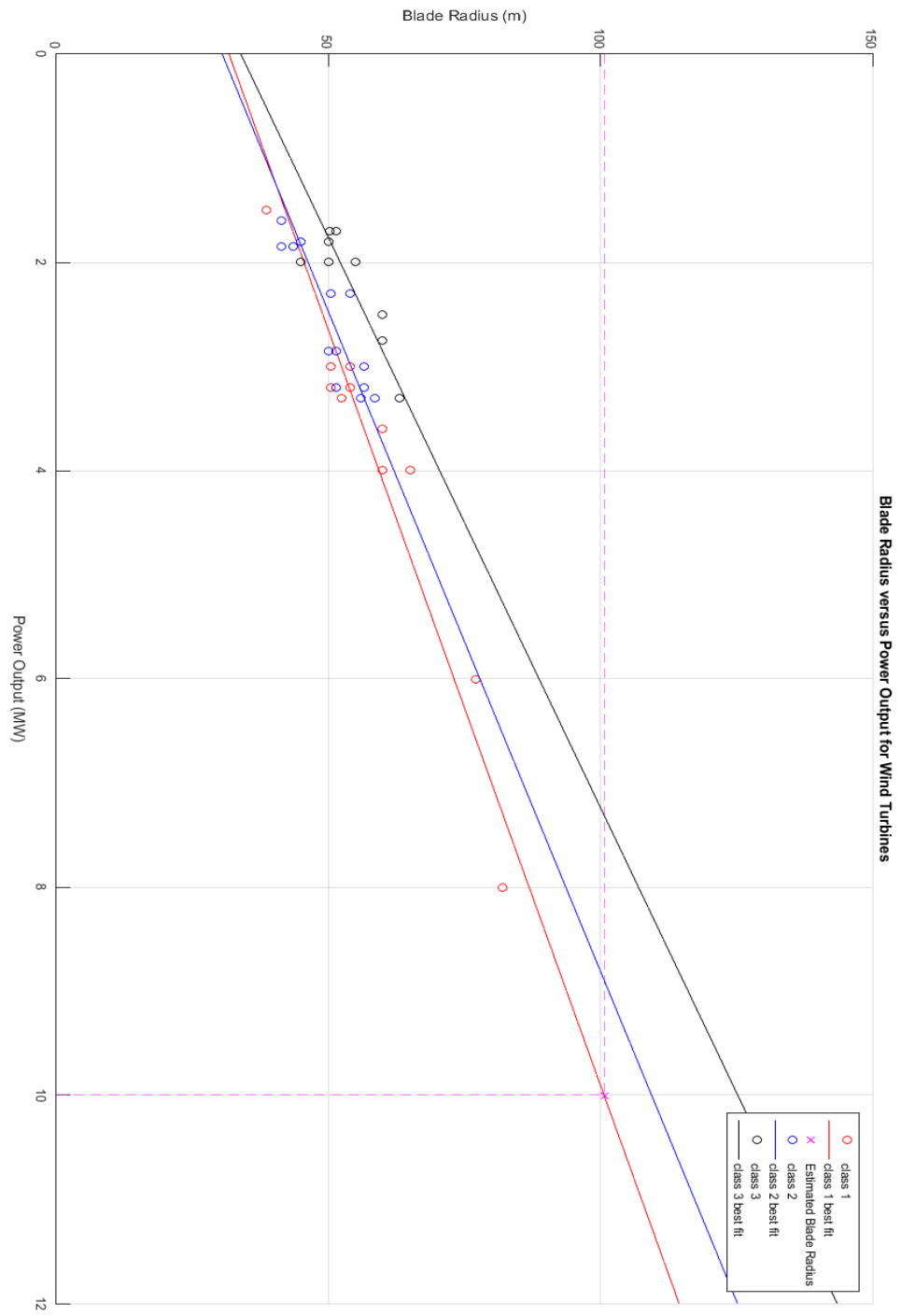

Output

The estimated blade length is 98.16 m

The values displayed were obtained from the websites of GE, Vestas, and Siemens.

The Vestas V164-8.0 is actually a class S, but due to being offshore has been assumed to be class 1.

>>



Sources for Wind Turbine Data

Turbine	Source
GE 1.7-100/103	[52]
GE 1.85-82.5	
GE 1.85-87	
GE 2.5-120	
GE 2.75-120	
GE 2.85-100	[53]
GE 2.85-103	
GE 3.2-103	[52]
GE 1.6-82.5	
GE 1.5-77	
GE 1.7-103	
V110-2.0	[54]
V100-2.0	
V100-1.8	
V100-2.0	
V90-1.8	
V90-2.0	
V126-3.3	
V117-3.3	
V112-3.3	
V105-3.3	
V164-8.0	
SWT-2.3-101	[55]
SWT-2.3-108	
SWT-3.0-101	
SWT-3.2-101	
SWT-3.0-108	
SWT-3.2-108	
SWT-3.0-113	
SWT-3.2-113	
SWT-3.6-120	
SWT-4.0-120	
SWT-4.0-130	
SWT-6.0-154	

Appendix B: Pixel Size Calculation Program

This MATLAB code provides pixel size information for a camera on a wind turbine blade. The camera may be rotated up or down, and placed anywhere along the length of the blade.

MATLAB Code

```
%% Pixel Area Calculator
%Accounts for camera placement and tilt. Two plots:
%Pixel area along blade length    %Pixel size compared to 23 by 23cm
target

clear
clc
clf

%Turbine information (currently projected GE 10MW blade radius)
BladeL = 98.16; %m
%Camera information (SENTECH STC-N632 Microcamera)
FL = 25*10^(-3); %m
CellHorz = 6.35*10^(-6); %m
CellVert = 7.4*10^(-6); %m
PixelHorz = 720; %Pix
PixelVert = 480; %Pix
SensHorz = CellHorz*PixelHorz; %m
SensVert = CellVert*PixelVert; %m
%Calculate FOV height and width
AngleHorz = 2*atand(SensHorz/(2*FL)); %Degrees
AngleVert = 2*atand(SensVert/(2*FL)); %Degrees
%User input data
Camloc = input('Distance from root of blade: ');
Camtilt = input('Camera vertical tilt: ');
%Account for camera tilt
AngleVert1 = AngleVert/2+Camtilt;
AngleVert2 = -AngleVert/2+Camtilt;
%Span of blade seen by camera FOV
x = (Camloc:0.1:BladeL);
%Preallocate matrices
PixAreaTop = zeros(length(x),1);
PixAreaBot = zeros(length(x),1);
PixWidthTop = zeros(length(x),1);
PixWidthBot = zeros(length(x),1);
PixHeight = zeros(length(x),1);
%Perform pixel calculations
for i = 1:length(x)
    HypTop = x(i)/abs(cosd(AngleVert1));
    HypBot = x(i)/abs(cosd(AngleVert2));
    WidTop = 2*tand(AngleHorz/2)*HypTop;
    WidBot = 2*tand(AngleHorz/2)*HypBot;
    HtTop = x(i)*tand(AngleVert1);
    HtBot = x(i)*tand(AngleVert2);
    Height = abs(HtTop-HtBot);
    PixHeight(i) = Height/PixelVert;
```

```

    PixWidthTop(i) = WidTop/PixelHorz;
    PixWidthBot(i) = WidBot/PixelHorz;
    PixAreaTop(i) = PixWidthTop(i)*PixHeight(i);
    PixAreaBot(i) = PixWidthBot(i)*PixHeight(i);
end
%First figure: pixel area versus position along length of blade
plot(x,PixAreaTop*100*100,'r')
hold on
plot(x,PixAreaBot*100*100,'b')
grid on
legend('Pixel Area at Top of FOV','Pixel Area at Bottom of FOV')
xlabel('Distance Along Blade (m)')
ylabel('Pixel Area (cm^2)')
title('Pixel Area versus Position Along Length of Turbine Blade')
hold off
%Print Values
[n m] = max(PixAreaTop);
[b c] = max(PixAreaBot);
targetArea = 23*23; %cm
if n < b
    nconv = n*100*100;
    PWT = PixWidthTop(m);
    PH = PixHeight(m);
    fprintf('The min pixel size occurs at top of FOV\n pixel size is
    %6.4f cm^2\n',nconv)
    PixelsPerTarget = targetArea/nconv;
    fprintf('%6.3f pixels fit within the target
    area\n',PixelsPerTarget)
else
    bconv = b*100*100;
    PWB = PixWidthBot(c);
    PH = PixHeight(c);
    fprintf('The min pixel size occurs at bottom of FOV\n pixel size is
    %6.4f cm^2\n',bconv)
    PixelsPerTarget = targetArea/bconv;
    fprintf('%6.3f pixels fit within the target
    area\n',PixelsPerTarget)
end
%Second figure: Compare target area to that of a single pixel
figure
for i = 1:3
    subplot(2,3,i)
    grid on
    Back = area([0,24],[24,24],'FaceColor',[0 0 1]);
    hold on
    Top =
    area([0,PixWidthTop(round((i)*length(x)/3))*100],[PixHeight(round((i)*length(x)/3))*100,PixHeight(round((i)*length(x)/3))*100],'FaceColor',[0 1 0]);
    hold on
    xlabel('Length (cm)')
    ylabel('Length (cm)')
    axis([0 30 0 30])
    axis square
    if i == 2
        title('Target Size versus Pixel Area at 1/3, 2/3, and End of the
        Turbine Blade')
    end
end

```

```

end
end
legend('Target Size','Pixel Area Near Top of FOV')
for j = 1:3
subplot(2,3,j+3)
grid on
Back = area([0,24],[24,24],'FaceColor',[0 0 1]);
hold on
Bottom =
area([0,PixWidthBot(round((j)*length(x)/3))*100],[PixHeight(round((j)*length(x)/3))*100,PixHeight(round((j)*length(x)/3))*100,'FaceColor',[1 0 0]);
hold on
xlabel('Length (cm)')
ylabel('Length (cm)')
axis([0 30 0 30])
axis square
end
legend('Target Size','Pixel Area Near Bottom of FOV')

```

Output

```

Distance from root of blade: 0

Camera vertical tilt: 4

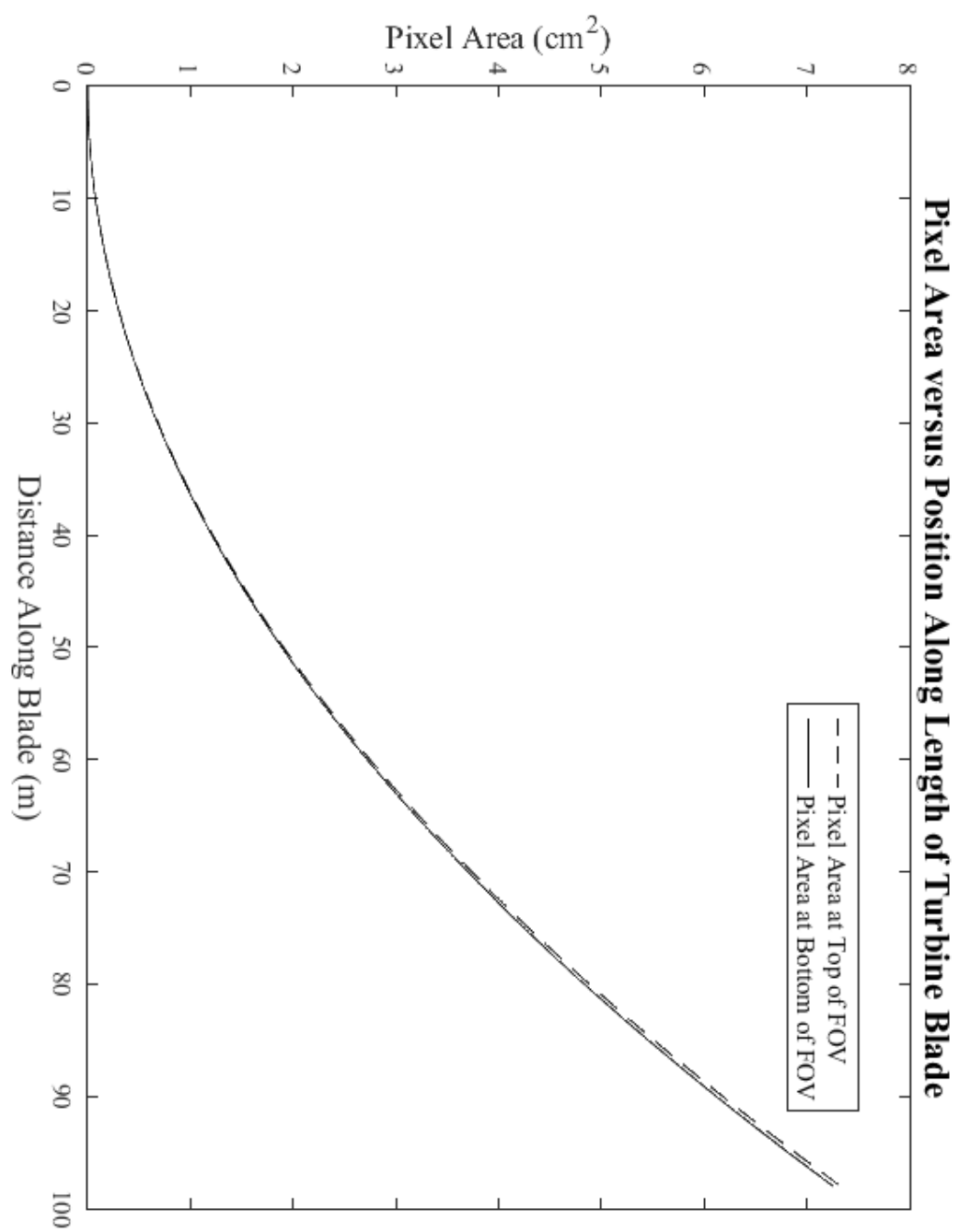
The min pixel size occurs at top of FOV

pixel size is 7.2710 cm^2

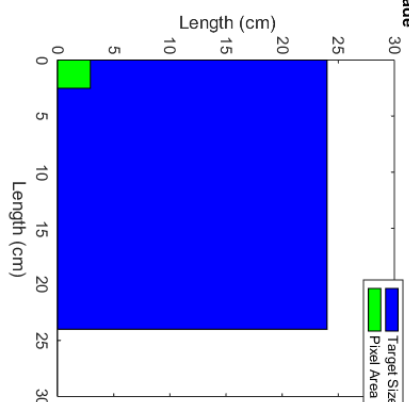
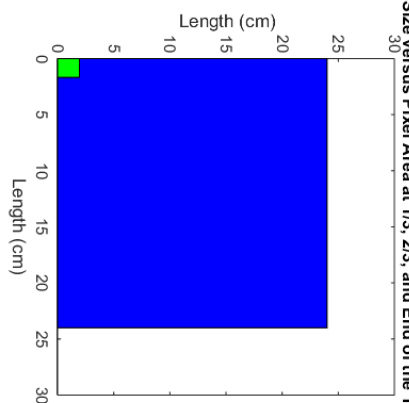
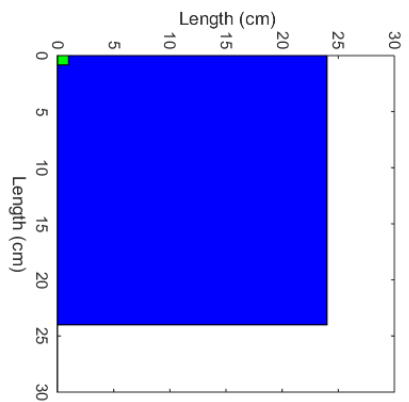
72.755 pixels fit within the target area

>>

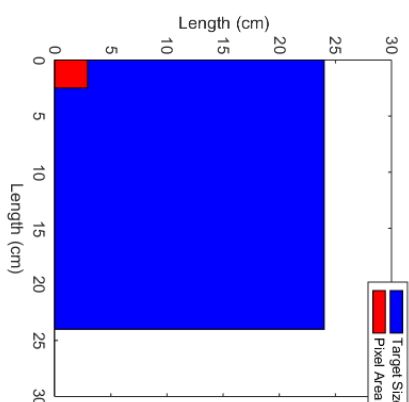
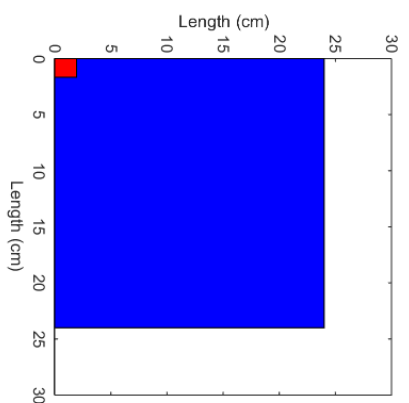
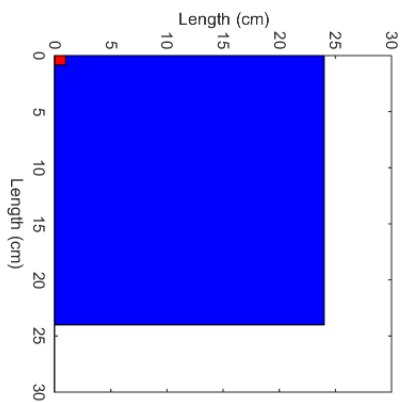
```



Target Size versus Pixel Area at 1/3, 2/3, and End of the Turbine Blade



Target Size
Pixel Area Near Top of FOV



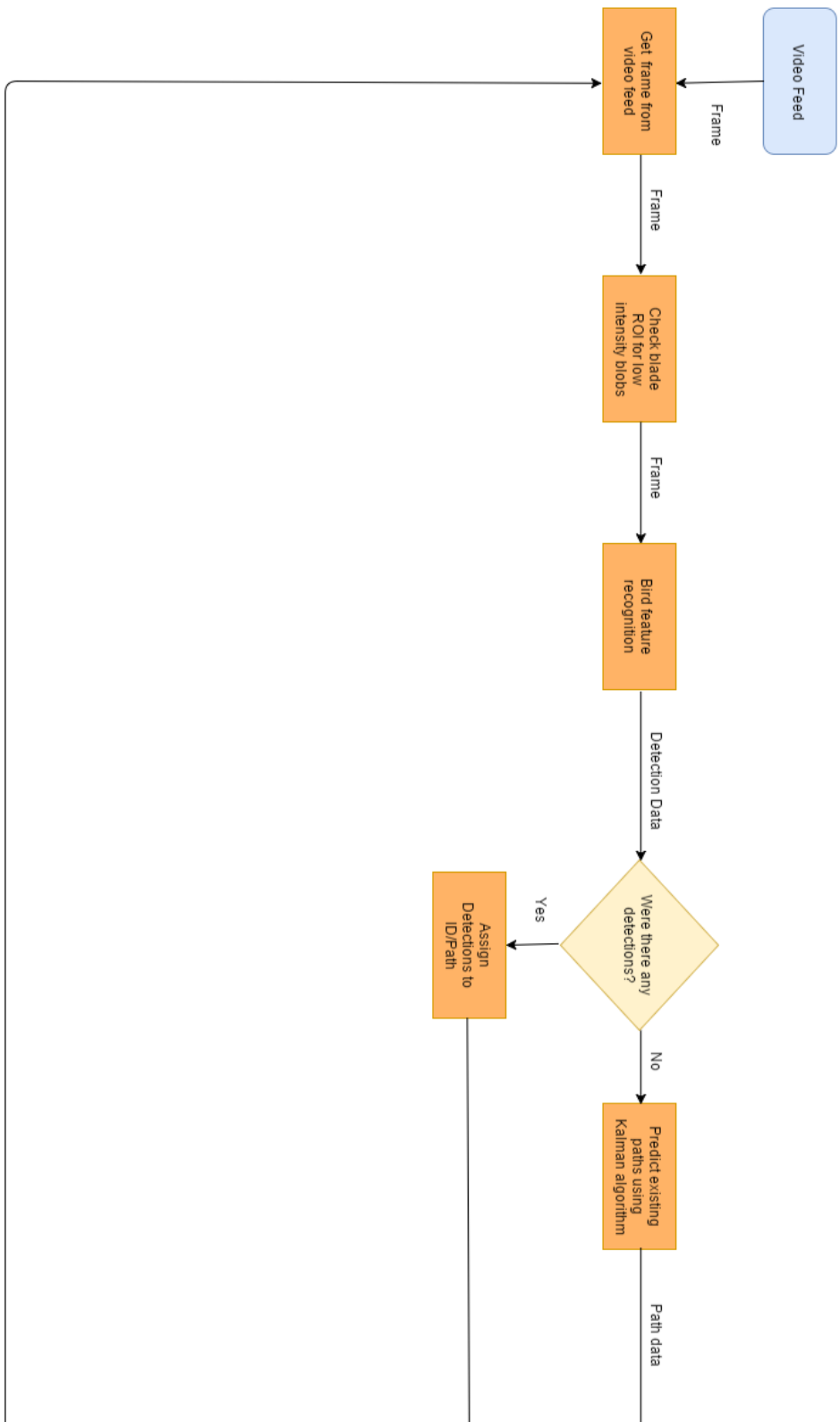
Target Size
Pixel Area Near Bottom of FOV

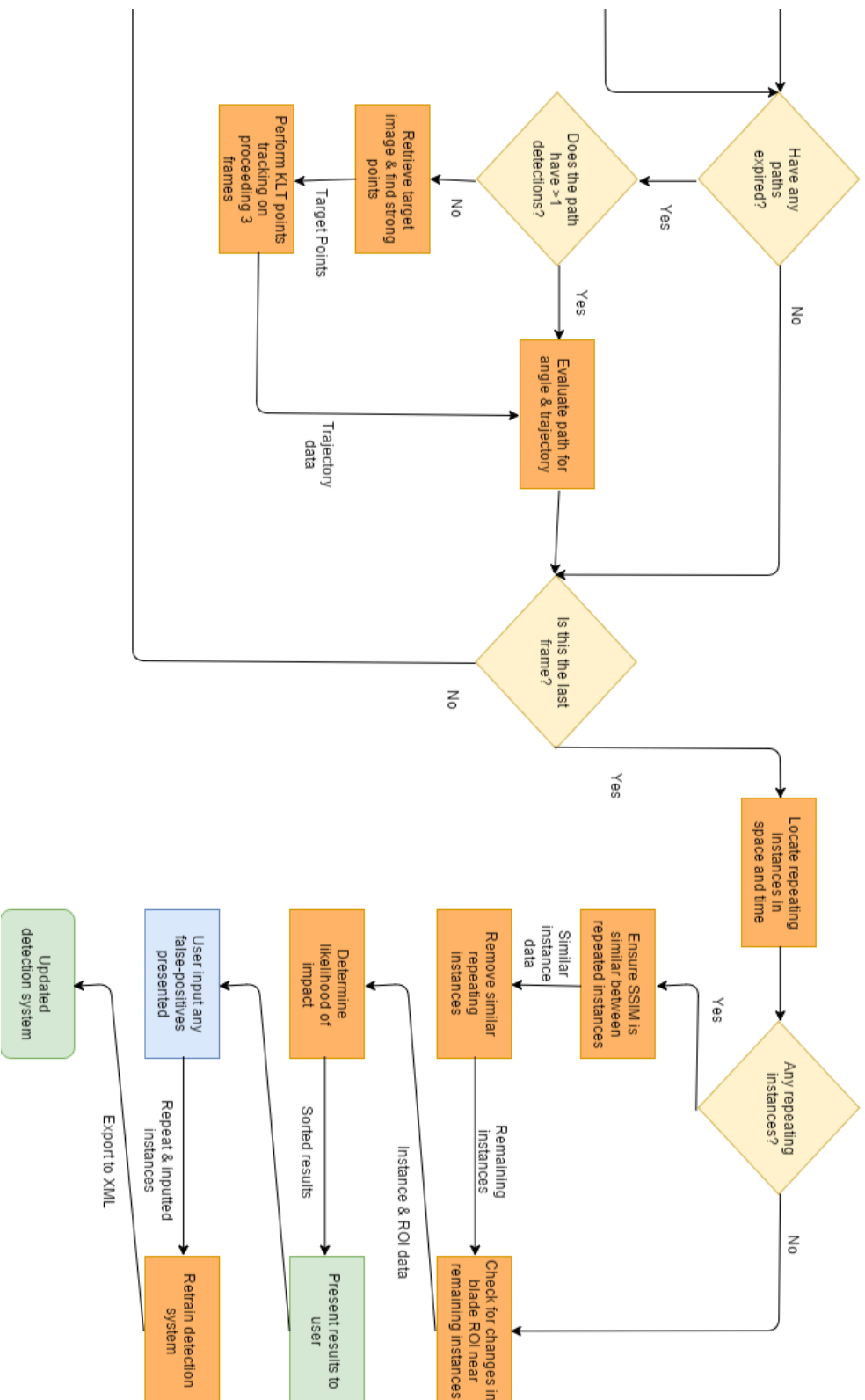
Appendix C: Camera Case Dimension

The following drawings provide dimensions for the critical parameters of the first iteration of on-blade camera casing. This model has been toleranced to allow for error during printing, by expanding through holes and regions for component installment. The model was printed as two separate parts, including the base plate, and upper shell.

Appendix D: Avian Program Logic Diagram

This appendix outlines the overall avian interaction program architecture through a block diagram.





Appendix E: Avian Interaction Program

In this appendix, the entire avian interaction MATLAB program is presented. For ease of use, the primary code is listed first, followed by its satellite functions in the order that they are utilized. The program can be operated in one of two ways, including "continuous", and "triggered". Continuous operation runs through the entire input video before presenting results, while triggered searches for frames before and after a peak in an input signal. The latter option was used to simulate vibrational input from the contact microphones or accelerometers used in [9]. Further options include the recording of video to review detections, and updating the cascade object detection algorithm after processing. The tracking framework and management of multiple targets was adapted from [42].

Primary Code

```
clear
clc
clf
close all

%% ***Avian Interaction Program***
% Two modes of operation:
% 1. analyze one video until completion (continuous)
% 2. analyze video surrounding an event using a frame buffer
% (triggered)

% option 2 simulates triggering from vibrational sensors
% Additional Options:
% Record the detections in a new video file
% Update detection system using FP instances

%% NOTES
% Check Zonecheck for framesize before use
% Tick regen & video +1
% Detection using GoPro is on 16th iteration
% Rotation in frames for GoPro is 187
% Blade pitch for GoPro is ~-30 degrees
% Rotation in frame for Sentech outdoor trials is 100
% detector = vision.CascadeObjectDetector('Example_Detection_1.xml');
% for Sentech trial

%% User Prompts
Type = input('Continuous [1] or Triggered [2]? ');
Record = input('Would you like to record this session? [1] yes [2] no ');
Update = input('Should the detection algorithm be updated after processing? [1] yes [2] no ');
```

```

%% Detect birds using cascade object detector
% Launch detector
detector = vision.CascadeObjectDetector('Test_02112016_2nd.xml');
% Set detector parameters
detector.MergeThreshold = 12; %This parameter can be varied --> lower
for
% increased number of false positives %14 for regen
rotationSpeed = 187; %frames per rotation
BladePitch = -30; %degrees

%% Video Setup
Vid = VideoReader('New_blade_cropped4.avi'); %Read the file
numOfFrames = Vid.NumberOfFrames %Get the number of frames
if Record == 1
vidWrite = VideoWriter('Test_02112016_Trial3rd_mergethresh12.avi');
%Record results
vidWrite.FrameRate = 25; %Framerate for recorded results
open(vidWrite); %Begin recording
end
switch Type
    case 1 %Continuous
        EndFrame = 2200;
        StartFrame = 200; %Begin video from this frame
        type = 1;
    case 2 %Triggered
        StartFrame = 1;
        signal = load('signal.mat'); %Select signal file
        signal = signal.signal;
        timeline = (1/60).*(1:numOfFrames);
        plot(timeline,signal)
        axis([0 numOfFrames/60 -14 0]);
        title('Accelerometer Data');
        xlabel('Time [sec]');
        ylabel('Acceleration [m/s^2]');
        figure
        frameStore = signalCheck(signal);
        EndFrame = length(frameStore);
        type = 2;
end

%% Initialize blob analysis
% Used to find centroids of detections
blob = vision.BlobAnalysis; %Launch 1st blob analysis
blob.AreaOutputPort = false; %No use for area
blob.BoundingBoxOutputPort = false; %Bounding boxes already created by
cascade object detector
blob.MinimumBlobArea = 1; %1 in order to prevent errors
blob.MaximumCount = 1; %For no more than one centroid per detection
% Used to locate blobs inside blade perimeter
bladeBlob = vision.BlobAnalysis; %Launch 2nd blob analysis
bladeBlob.AreaOutputPort = false; %No current use for area
bladeBlob.BoundingBoxOutputPort = true; %Bounding boxes
bladeBlob.CentroidOutputPort = false; %No need for centroids
bladeBlob.MinimumBlobArea = 40; %Prevent noise from triggering system
bladeBlob.MaximumBlobArea = 400; %Prevent large shadows from triggering

```



```

bladeBlob.MaximumCount = 3; %10 max detections
bladeBlob.ExcludeBorderBlobs = true; %Requires movement fully in
perimeter

%% Initialize Kalman Filter REF: [42]
vision.KalmanFilter; %Initialize the Kalman filter
% Setup tracks structure with fields for ID's, bounding boxes, age,
etc.
tracks = struct(...
    'id', {}, ...
    'bbox', {}, ...
    'kalmanFilter', {}, ...
    'age', {}, ...
    'totalVisibleCount', {}, ...
    'consecutiveInvisibleCount', {}); %Key tracking items
stored in struct
nextId = 1; %First ID to assign
centroidKeeper = []; %An empty array to store the centroids and
information of detections
reporter = []; %An empty array to store detection & tracking data for
later analysis

% References to MATLAB's Multiple Object Tracking framework will appear
% throughout code.

%% Setup blade ROI structure
frame = read(Vid, StartFrame); %Obtain the first frame
frame = rgb2gray(frame); %Convert RGB frame to grayscale
[Mask] = boundaries(frame); %Boundaries function allows the user to
outline the blade
bboxes = []; %Empty array for future use

%% For loop for reading frames
% The each frame is analyzed in consecutive order

for num = StartFrame:EndFrame %Loop through specified length of video
    %% Obtain frame from video
    if type == 2 %If signal activated
        num = frameStore(num); %The variable controlled by the outer-
most for loop iterates through the frames located via the signalCheck
function
    end
    img = read(Vid,num); %Read frames
    framegray = rgb2gray(img); %Convert frame to grayscale

    %% Locate blobs within blade ROI
    % This checks the blade ROI for new blobs
    bboxFind = blobCheck(framegray,Mask,bladeBlob,num); %Blob analysis
fcf
    for q = 1:size(bboxFind,1);
        % Add any located blobs to a matrix to be evaluated at a later
time
        if isempty(bboxes)
            bboxes(1,1:5) = bboxFind(q,:); %If empty create 1st row
        else
            bboxes(end+1,1:5) = bboxFind(q,:); %Expand matrix

```

```

        end
    end

    %% Detect potential birds
    bbox = step(detector,img); %Step the detector with the color image
    [R,C] = size(bbox); %Find size of bbox struct (provides the number
of detections)
    if R > 0 %In order to prevent this section from running with no
detections
        %Combine overlapping bounding boxes (NOTE: only for two bboxes)
        [bboxC] = bboxCombine(bbox); %Collapse any intersecting bounding
boxes
        [R2,C2] = size(bboxC); %Determine the size of the reformatted
bboxes
        centroid = zeros(R2,2); %Create an empty array for centroids
        for i = 1:R2 %Loop through detections
            framesect = framegray(bboxC(i,2):bboxC(i,4)+bboxC(i,2),...
%Examine bbox sections
                bboxC(i,1):bboxC(i,3)+bboxC(i,1));
            level = graythresh(framesect); %Determine local threshold value
            framesectBW = im2bw(framesect,level); %Convert section to
binary
            sectBW = imcomplement(framesectBW); %Birds are usually dark
compared to background- inverse
            centroid(i,:) = step(blob,sectBW); %Locate the centroid
            centroid(i,1) = centroid(i,1) + bboxC(i,1); %Assign the
centroids
            centroid(i,2) = centroid(i,2) + bboxC(i,2); %Assign the
centroids
        end
    end

    %% Predict new locations REF: [42] (Framework)
    for j = 1:length(tracks)
        bboxA = tracks(j).bbox; %Cycle through bounding boxes
        predictedCentroid = predict(tracks(j).kalmanFilter); %Predict
location
        predictedCentroid = int16(predictedCentroid) -
int16(bboxA(3:4)/2); %Convert to 16bit int. Bbox starts from corner,
t/f shift by 0.5 height and width
        tracks(j).bbox = [predictedCentroid, bboxA(3:4)]; %Update
bounding box
    end

    %% Assignment cost analysis REF: [42]
    if R > 0 %If detections occurred in this frame
        nTracks = length(tracks); %Number of existing tracks
        nDetections = size(centroid, 1); %Number of detections in this
frame
        cost = zeros(nTracks, nDetections); %Empty matrix
        for k = 1:nTracks
            cost(k,:) = distance(tracks(k).kalmanFilter, centroid); %Find
distance
        end
        costOfNonAssignment = 160; %Adjustable

```

```

[assignments, unassignedTracks, unassignedDetections] =
assignDetectionsToTracks(cost, costOfNonAssignment); %Assign detections
end

%% Updating assigned tracks REF: [42]
if R > 0 %If detections occurred in this frame
numAssignedTracks = size(assignments, 1);
for l = 1:numAssignedTracks
    trackIdx = assignments(l, 1); %Assigned track
    detectionIdx = assignments(l, 2); %Assigned ID
    centroids = centroid(detectionIdx, :); %Assign centroid
    bboxB = bboxC(detectionIdx, :); %Bbox is equal to the combined
of detection
    tracks(trackIdx).bbox = bboxB; %Update bounding box
    correct(tracks(trackIdx).kalmanFilter, centroids); %Updating
Kalman filter data
    tracks(trackIdx).age = tracks(trackIdx).age + 1; %Getting older
    tracks(trackIdx).totalVisibleCount =
tracks(trackIdx).totalVisibleCount + 1; %Visible +1
    tracks(trackIdx).consecutiveInvisibleCount = 0; %No longer
invisible
    centroidKeeper(end+1,1) = tracks(trackIdx).id; %Store the ID
    centroidKeeper(end,2:5) = bboxB; %Store the bbox
    centroidKeeper(end,6:7) = centroids; %Store the centroid
    centroidKeeper(end,8) = 1; %One signifies this is a detection
    centroidKeeper(end,9) = num; %Record the frame
end
end

%% Updating unassigned tracks REF [42]
if R > 0 %If detections occurred in this frame
    for o = 1:length(unassignedTracks)
        ind = unassignedTracks(o);
        tracks(ind).age = tracks(ind).age + 1; %Getting older
        tracks(ind).consecutiveInvisibleCount =
tracks(ind).consecutiveInvisibleCount + 1; %Invisible +1
        centroidKeeper(end+1,1) = tracks(ind).id; %Store the ID
        centroidKeeper(end,2:5) = [0,0,0,0]; %No bbox
        centroidKeeper(end,6:7) =
predict(tracks(ind).kalmanFilter); %Store prediction
        centroidKeeper(end,8) = 2; %Prediction
        centroidKeeper(end,9) = num; %Record the frame
    end
elseif ~isempty(tracks) && R == 0
    % this is necessary for maintaining Kalman filter when
detections
    % do not occur
    for z = 1:length(tracks)
        tracks(z).age = tracks(z).age + 1;
        tracks(z).consecutiveInvisibleCount =
tracks(z).consecutiveInvisibleCount + 1;
        centroidKeeper(end+1,1) = tracks(z).id;
        centroidKeeper(end,2:5) = [0,0,0,0];
        centroidKeeper(end,6:7) = predict(tracks(z).kalmanFilter);
        centroidKeeper(end,8) = 2; %Two signifies this is a
prediction
        centroidKeeper(end,9) = num;
    end
end

```

```

end
%% Deleting lost tracks REF: [42]
if ~isempty(tracks)
    invisibleForTooLong = 15; %Variable parameter
    ageThreshold = 25; %Variable parameter
    ages = [tracks(:).age]; %Listing of ages
    totalVisibleCounts = [tracks(:).totalVisibleCount]; %Number of
visible instances per track
    visibility = totalVisibleCounts ./ ages; %Proportion of visible
and age
    lostInds = (ages < ageThreshold & visibility < 0.1) |
[tracks(:).consecutiveInvisibleCount] >= invisibleForTooLong; %Find
tracks to remove
    if ~isempty(tracks(lostInds))
        endedTracks = tracks(lostInds); %Find ended tracks
        for j = 1:size(endedTracks,2)
            ID = endedTracks(j).id; %ID per loop iteration
            check = find(centroidKeeper(:,1) == ID); %Find this ID
in storage
            Detections = find(centroidKeeper(check,8) == 1); %Find
which of these are detections
            if max(Detections) == 1 %Only one detection, requiring
backup tracking
                status = KLTpoints_revised(centroidKeeper(check,:),
Vid, BladePitch); %Secondary tracking
                if status == 1 %Angled and towards blade
                    if isempty(reporter)
                        reporter(1,1) = ID; %Record ID
                        reporter(1,2) = 1; %Path type
                    else
                        reporter(end+1,1) = ID; %Record ID
                        reporter(end,2) = 1; %Path type
                    end
                elseif status == 2 %Little to no angle of path
                    if isempty(reporter)
                        reporter(1,1) = ID; %Record ID
                        reporter(1,2) = 2; %Path type
                    else
                        reporter(end+1,1) = ID; %Record ID
                        reporter(end,2) = 2; %Path type
                    end
                elseif status == 3 %Unsuccessfully tracked
                    if isempty(reporter)
                        reporter(1,1) = ID; %Record ID
                        reporter(1,2) = 3; %Path type
                    else
                        reporter(end+1,1) = ID; %Record ID
                        reporter(end,2) = 3; %Path type
                    end
                else %ergo status == 4 %Angled and away from
blade
                    if isempty(reporter)
                        reporter(1,1) = ID; %Record ID
                        reporter(1,2) = 4; %Path type
                    else
                        reporter(end+1,1) = ID; %Record ID
                        reporter(end,2) = 4; %Path type
                    end
                end
            end
        end
    end
end

```

```

        end
    end
    else
        status =
evalKalmanTracks(centroidKeeper(check,:),BladePitch); %Enough
detections for simple path evaluation
        if status == 1
            if isempty(reporter)
                reporter(1,1) = ID; %Record ID
                reporter(1,2) = 1; %Path type
            else
                reporter(end+1,1) = ID; %Record ID
                reporter(end,2) = 1; %Path type
            end
        elseif status == 2
            if isempty(reporter)
                reporter(1,1) = ID; %Record ID
                reporter(1,2) = 2; %Path type
            else
                reporter(end+1,1) = ID; %Record ID
                reporter(end,2) = 2; %Path type
            end
        elseif status == 3
            if isempty(reporter)
                reporter(1,1) = ID; %Record ID
                reporter(1,2) = 3; %Path type
            else
                reporter(end+1,1) = ID; %Record ID
                reporter(end,2) = 3; %Path type
            end
        else
            if isempty(reporter)
                reporter(1,1) = ID; %Record ID
                reporter(1,2) = 4; %Path type
            else
                reporter(end+1,1) = ID; %Record ID
                reporter(end,2) = 4; %Path type
            end
        end
    end
end
end
    tracks = tracks(~lostInds); %Refine struct to exclude lost
tracks
end
    %% Create new tracks REF: [42]
    if R > 0 %If detections occurred in this frame
        centroid = centroid(unassignedDetections, :); %The centroids for
each new/unassigned detection
        for p = 1:size(centroid, 1)
            centroidNew = centroid(p,:); %New centroid for each iteration
of for loop
            bboxD = bboxC(p,:);
            %Create a Kalman filter object
            kalmanFilter =
configureKalmanFilter('ConstantVelocity',centroidNew,[200, 50], [100,
25], 100);

```

```

        %Create a new track
        newTrack = struct('id', nextId, 'bbox', bboxD, 'kalmanFilter',
kalmanFilter, 'age', 1, 'totalVisibleCount', 1,
'consecutiveInvisibleCount', 0);
        %Add it to the array of tracks
        tracks(end + 1) = newTrack;
        %Add results to storage
        if isempty(centroidKeeper)
            centroidKeeper(1,1) = nextId;
            centroidKeeper(1,2:5) = bboxD;
            centroidKeeper(1,6:7) = centroidNew;
            centroidKeeper(1,8) = 1;
            centroidKeeper(1,9) = num;
        else
            centroidKeeper(end+1,1) = nextId;
            centroidKeeper(end,2:5) = bboxD;
            centroidKeeper(end,6:7) = centroidNew;
            centroidKeeper(end,8) = 1;
            centroidKeeper(end,9) = num;
        end
        %Increment the next id
        nextId = nextId + 1;
    end
end
%% Display the results. For the following five lines, REF: [42]
if ~isempty(tracks)
    ids = int32([tracks(:).id]);
    labels = cellstr(int2str(ids'));
    bboxNotation = cat(1, tracks.bbox);
    frameNotated = insertObjectAnnotation(framegray,
'rectangle', bboxNotation, labels);
    framegray = im2uint8(frameNotated);
end
framegray = im2uint8(framegray); %8 bit unsigned integer RGB image
(despite name, allows for yellow bboxes)
if Record == 1 %If specified to record
    writeVideo(vidWrite,framegray)
end
end
%^End of master loop
if Record == 1
    close(vidWrite); %Close recording if needed
end

%% Finish tracking any remaining instances.
reporter =
FinishTracking(tracks,reporter,centroidKeeper,Vid,BladePitch);
%% Filter repeat instances
[cleanedCache, data] =
reportAnalysis2(reporter,centroidKeeper,rotationSpeed); %Locates repeat
instances (based on space/time) and separates them from global data
[dataRevised, DifferingDetections] = RepeatCheck(Vid, data,
centroidKeeper); %Checks the repeat instances by comparing SSIM
if ~isempty(DifferingDetections)
    [cleanedCache] = RevisedCache(cleanedCache, DifferingDetections,
centroidKeeper); %Places low SSIM score instances back into review
end

```

```

close all %Close current figures (in preparation for results
presentation)

%% Instance Information (check for changes in blade ROI, Zone,
Trajectory/Path Type)
if cleanedCache ~= 0 %Ensuring there are results
NumberIDs(:,1) = unique(cleanedCache(:,1)); %Find unique IDs
LengthAnalysis = size(NumberIDs,1); %How many unique IDs are there?
for y = 1:size(cleanedCache,1)
    bladeRoiCheck = zeros(1,1);
    for t = 1:61 %Check 30 frames before and after blade ROI
activity occurs
        instanceCheck = find(bboxes(:,5) == cleanedCache(y,2)-
t+31 ...
            & bboxes(:,3) ~= 0 & bboxes(:,4) ~= 0); %If no
blob, 0's will exist for bboxes
        if isempty(instanceCheck)
            bladeRoiCheck(t,1) = 0;
        else
            bladeRoiCheck(t,1) = size(instanceCheck,1);
        end
    end
    changesInRoi = unique(bladeRoiCheck(:,1)); %Unique changes in blade
ROI
    changesInRoi = size(changesInRoi,1); %Number of unique changes in
blade ROI
    if changesInRoi > 1
        key(y,1) = 1; %If we get some changes in the blade ROI lets
write that down
    else
        key(y,1) = 0; %No detectable changes
    end
    zone(y,1) = zoneCheck(cleanedCache(y,8)); %Proximity of targets to
blade
    typeFind = find(reporter(:,1) == cleanedCache(y,1)); %Recall the
trajectory
    if ~isempty(typeFind)
        type(y,1) = reporter(typeFind(1,1),2); %Assign path
    else
        type(y,1) = 0; %No type. Error.
    end
    IDy(y,1) = cleanedCache(y,1); %Do not forget the ID
end
%% Classification of results
for x = 1:LengthAnalysis
    Id = NumberIDs(x,1);
    rows = find(IDy(:,1) == Id);
    mtype = max(type(rows,1)); %Recall type
    mkey = max(key(rows,1)); %Recall blade ROI changes
    mzone = max(zone(rows,1)); %Recall zone
    %% Likelihood of collision
    %Very Strong
    if mkey == 1 && mtype == 1 && (mzone == 2 || mzone == 3)
        FinalMat(x,1) = Id; FinalMat(x,2) = 2; FinalMat(x,3) = 4;
    %Strong
    elseif (mkey == 0 && mtype == 1 && mzone == 3) || (mkey == 1 && ...

```

```

        (mtype == 1 && mzone == 1) || (mtype == 2 && mzone == 3) ||
...
        (mtype == 3 && mzone == 3) || (mtype == 4 && mzone == 3))
    if mtype == 1 || mtype == 2 || mtype == 4
        FinalMat(x,1) = Id; FinalMat(x,2) = 2; FinalMat(x,3) = 3;
    else
        FinalMat(x,1) = Id; FinalMat(x,2) = 1; FinalMat(x,3) = 3;
    end
    %Moderate
    elseif (mkey == 0 && ((mtype == 1 && (mzone == 1 || mzone == 2)))
|| (mtype == 2 ...
        && mzone == 3) || (mtype == 3 && (mzone == 2 || mzone ==
3))) || ...
        (mtype == 4 && mzone == 3) || (mkey == 1 && ((mtype == 2 &&
...
        (mzone == 1 || mzone == 2))) || (mtype == 3 && (mzone == 1
|| mzone == 2)) ...
        || (mtype == 4 && mzone == 2))
    if mtype == 1 || mtype == 2 || mtype == 4
        FinalMat(x,1) = Id; FinalMat(x,2) = 2; FinalMat(x,3) = 2;
    else
        FinalMat(x,1) = Id; FinalMat(x,2) = 1; FinalMat(x,3) = 2;
    end
    %N/A
    elseif mtype == 0
        FinalMat(x,1) = Id; FinalMat(x,2) = 0; FinalMat(x,3) = 0;
    %Low
    else
        if mtype == 1 || mtype == 2 || mtype == 4
            FinalMat(x,1) = Id; FinalMat(x,2) = 2; FinalMat(x,3) = 1;
        else
            FinalMat(x,1) = Id; FinalMat(x,2) = 1; FinalMat(x,3) = 1;
        end
    end
end
end
%% Present results to user
for z = 1:LengthAnalysis
    %Determine the plot size
    range = find(cleanedCache(:,1) == NumberIDs(z,1));
    plotSize = size(range,1);
    if plotSize == 1
        sqr(1,1) = 1; sqr(1,2) = 1; fact = 0; limit = 0;
    elseif plotSize == 2
        sqr(1,1) = 1; sqr(1,2) = 2; fact = 0; limit = 0;
    elseif plotSize == 3
        sqr(1,1) = 1; sqr(1,2) = 3; fact = 0; limit = 0;
    elseif plotSize <= 6 && plotSize > 3
        sqr(1,1) = 2; sqr(1,2) = 3; fact = 0; limit = 0;
    elseif plotSize <= 9 && plotSize > 6
        sqr(1,1) = 3; sqr(1,2) = 3; fact = 0; limit = 0;
    elseif plotSize <= 12 && plotSize > 9
        sqr(1,1) = 4; sqr(1,2) = 3; fact = 0; limit = 0;
    elseif plotSize <= 15 && plotSize > 12
        sqr(1,1) = 5; sqr(1,2) = 3; fact = 0; limit = 0;
    elseif plotSize <= 35
        sqr(1,1) = 6; sqr(1,2) = 3; fact = 1; limit = 0;
    elseif plotSize <= 41

```



```

        sqr(1,1) = 7; srr(1,2) = 3; fact = 1; limit = 0;
    else
        fact = 1; limit = 1;
    end
figure
if fact == 0;
    for r = 1:plotSize
        rungl = range(r,1);
        frameDisp = read(Vid,cleanedCache(rungl,2)); %Obtain frame for
instance
        frameDisp = rgb2gray(frameDisp); %Convert to grayscale
        bboxDisp = cleanedCache(rungl,3:6); %Find the bbox
        dispNotated = insertObjectAnnotation(frameDisp, 'rectangle',
bboxDisp, 'Detected Target'); %Annotate
        dispNotated = im2uint16(dispNotated); %Convert image to
unsigned 16bit
        subplot(sqr(1,1),sqr(1,2),r) %Subplot ration
        imshow(dispNotated); %Show
    end
elseif fact == 1 && limit == 0;
    for r = 1:plotSize
        if mod(r,2) ~= 0
            rungl = range(r,1);
            frameDisp = read(Vid,cleanedCache(rungl,2)); %Obtain frame
for instance
            frameDisp = rgb2gray(frameDisp); %Convert to grayscale
            bboxDisp = cleanedCache(rungl,3:6); %Find the bbox
            dispNotated = insertObjectAnnotation(frameDisp,
'rectangle', bboxDisp, 'Detected Target'); %Annotate
            dispNotated = im2uint16(dispNotated); %Convert image to
unsigned 16bit
            subplot(sqr(1,1),sqr(1,2),r) %Subplot ration
            imshow(dispNotated); %Show
        end
    end
else
    rungl = range(1,1);
    frameDisp = read(Vid,cleanedCache(rungl,2)); %Obtain frame
for instance
    frameDisp = rgb2gray(frameDisp); %Convert to grayscale
    bboxDisp = cleanedCache(rungl,3:6); %Find the bbox
    dispNotated = insertObjectAnnotation(frameDisp,
'rectangle', bboxDisp, 'Detected Target'); %Annotate
    dispNotated = im2uint16(dispNotated); %Convert image to
unsigned 16bit
    subplot(1,1,1) %Subplot ration
    imshow(dispNotated); %Show
end
rungr2 = find(FinalMat(:,1) == NumberIDs(z,1));
idZ = NumberIDs(z,1);
%% Provide labeling for instances
if FinalMat(rung2,2) == 1
    if FinalMat(rung2,3) == 4
        suptitle(sprintf('ID: %3.3g \nUnsuccessfully Tracked, Very
Strong Chance of Impact',idZ));
        set(gcf,'Color','r');
    elseif FinalMat(rung2,3) == 3

```

```

        suptitle(sprintf('ID: %3.3g \nUnsuccessfully Tracked,
Strong Chance of Impact',idZ));
        set(gcf,'Color','y');
    elseif FinalMat(rung2,3) == 2
        suptitle(sprintf('ID: %3.3g \nUnsuccessfully Tracked,
Moderate Chance of Impact',idZ));
        set(gcf,'Color','g');
    else
        suptitle(sprintf('ID: %3.3g \nUnsuccessfully Tracked, Low
Chance of Impact',idZ));
        set(gcf,'Color','b');
    end
    elseif FinalMat(rung2,2) == 2
        if FinalMat(rung2,3) == 4
            suptitle(sprintf('ID: %3.3g \nSuccessfully Tracked, Very
Strong Chance of Impact',idZ));
            set(gcf,'Color','r');
        elseif FinalMat(rung2,3) == 3
            suptitle(sprintf('ID: %3.3g \nSuccessfully Tracked, Strong
Chance of Impact',idZ));
            set(gcf,'Color','y');
        elseif FinalMat(rung2,3) == 2
            suptitle(sprintf('ID: %3.3g \nSuccessfully Tracked,
Moderate Chance of Impact',idZ));
            set(gcf,'Color','g');
        else
            suptitle(sprintf('ID: %3.3g \nSuccessfully Tracked, Low
Chance of Impact',idZ));
            set(gcf,'Color','b');
        end
    else
        suptitle(sprintf('ID: %3.3g \nUnknown Flight Parameters',idZ));
        set(gcf,'Color','w');
    end
end

end

else
    msgbox('no detected events'); %Report if no instances
end

if Update == 1
    %Execute learning program
    NegativeSave(Vid, dataRevised, centroidKeeper);
end

```

Satellite Functions (Supporting Architecture)

signalCheck

Finding the peak of a signal, and retrieve a frame buffer surrounding the peak.

```

function frameStore = signalCheck(signal)
frameStore = [];
frameKeep = [];

```

```

frameFind = [];
for i = 1:length(signal)
    if signal(i) > -5 || signal(i) < -8
        if ~isempty(frameKeep)
            frameKeep(end+1,1) = i;
        else
            frameKeep(1,1) = i;
        end
    end
end
if ~isempty(frameKeep)
for j = 1:length(frameKeep)
    for k = 1:61
        if isempty(frameFind)
            frameFind(1,1) = frameKeep(j) + k - 31;
        else
            frameFind(end+1,1) = frameKeep(j) + k - 31;
        end
    end
end
end
end
frameStore = unique(frameFind(:,1));
end

```

boundaries

This function allows the user to draw a polygon surrounding the inner blade ROI. These bounds are used later in the program to apply a mask to the image, essentially cropping out everything but the inner regions of the blade.

```

function [Mask] = boundaries(firstFrame)
%% User creates boundary safely within the constraints of the wind
turbine blade
% This will be used to check for large contrast changes

%the input frame should be grayscale & representative of the blade
%with no occlusions and nominal contrast
imshow(firstFrame);
bounds = impoly;
wait(bounds);
Mask = createMask(bounds);
End

```

blobCheck

This function is used to crop the frame from the boundary surrounding the blade. The cropped region is then thresholded, and blob analysis is performed. Cropping is performed using the perimeter set by the user prior to operation. Bounding boxes created around any detected blobs are

then stored. These bounding boxes provide all necessary information for later analysis. Note that the cropping technique was adapted from [56].

```
function [bboxes] = blobCheck(frame,Mask,bladeBlob,num)
%% User creates boundary safely within the constraints of the wind
turbine blade
% This will be used to check for constrast changes
%the input frame should be grayscale & representative of the blade
%with no occlusions and nominal contrast
cropped = frame;
cropped(~Mask) = 0;
threshold = graythresh(cropped(Mask));
threshold = threshold*0.5;
cropped = im2bw(cropped,threshold);
cropped = imcomplement(cropped);
cropped(~Mask) = 0;
%Cropping technique from
%[56]
bbox = step(bladeBlob, cropped);
if isempty(bbox)
    bboxes(1,1:4) = 0;
    bboxes(1,5) = num;
else
    for i = 1:size(bbox,1)
        bboxes(i,1:4) = bbox(i,:);
        bboxes(i,5) = num;
    end
end

end
```

bboxCombine

Often multiple detections occur on the same target- this function combines two bounding boxes if they overlap.

```
function [bboxC] = bboxCombine(bbox)
%This function is used to combine the bounding boxes of overlapping
%detections.
bboxes = size(bbox);
flag = 0;
if bboxes(1) == 0
    disp('NO DETECTIONS')
    flag = 1;
end
Combine = zeros(1,2);
if bboxes(1) > 1
    for i = 1:bboxes(1)
        for j = i:bboxes(1)-1
            overlapRatio = bboxOverlapRatio(bbox(i,:),bbox(j+1,:));
            if overlapRatio > 0 && Combine(1,1) == 0
```

```

        Combine(1,1) = i;
        Combine(1,2) = j+1;
    elseif overlapRatio > 0
        Combine(end+1,1) = i;
        Combine(end,2) = j+1;
    end
end
end
end
DontCombine = zeros(1,1);
[R2,C2] = size(Combine);
for u = 1:bboxes(1)
    mark = 0;
    for v = 1:R2
        if Combine(v,1) ~= u && Combine(v,2) ~= u
            mark = mark+1;
        end
    end
    if mark == R2 && DontCombine(1,1) == 0
        DontCombine(1,1) = u;
    elseif mark == R2
        DontCombine(end+1) = u;
    end
end
end
sized = size(DontCombine);
bboxC = zeros(R2,4);
if Combine(1,1) ~= 0
    for k = 1:R2
        if bbox(Combine(k,1),1) < bbox(Combine(k,2),1)
            bboxC(k,1) = bbox(Combine(k,1),1);
        else
            bboxC(k,1) = bbox(Combine(k,2),1);
        end
        if bbox(Combine(k,1),2) < bbox(Combine(k,2),2)
            bboxC(k,2) = bbox(Combine(k,1),2);
        else
            bboxC(k,2) = bbox(Combine(k,2),2);
        end
        if bbox(Combine(k,1),1)+bbox(Combine(k,1),3) <
bbox(Combine(k,2),1)+bbox(Combine(k,2),3)
            bboxC(k,3) = bbox(Combine(k,2),1)+bbox(Combine(k,2),3) -
bboxC(k,1);
        else
            bboxC(k,3) = bbox(Combine(k,1),1)+bbox(Combine(k,1),3) -
bboxC(k,1);
        end
        if bbox(Combine(k,1),2)+bbox(Combine(k,1),4) <
bbox(Combine(k,2),2)+bbox(Combine(k,2),4)
            bboxC(k,4) = bbox(Combine(k,2),2)+bbox(Combine(k,2),4) -
bboxC(k,2);
        else
            bboxC(k,4) = bbox(Combine(k,1),2)+bbox(Combine(k,1),4) -
bboxC(k,2);
        end
    end
end
if DontCombine(1,1) ~= 0

```

```

    for y = 1:sized(2)
        if bboxC(1,1) == 0 && flag == 0
            bboxC(1,:) = bbox(DontCombine(y),:);
        elseif flag == 0
            bboxC(end+1,:) = bbox(DontCombine(y),:);
        end
    end
end

end

end

```

KLTpoints_revised

This function constitutes the entirety of secondary tracking. If the number of detections for a target is one or less, this function uses the KLT algorithm to track points across frames. Within this function, the direction of travel is evaluated and classified for later use. [57] provided guidance on using the point tracking algorithm.

```

function [status] = KLTpoints_revised(centroidKeeper,Vid,BladePitch)
%% Initialize KLT Point Tracker, Ref for KLT point tracking: [44], [57]
indices = find(centroidKeeper(:,8) == 1,1,'first'); %Locate the
detection for this ID
frameSt = centroidKeeper(indices,9); %Locate the starting frame
bbox = centroidKeeper(indices,2:5); %Find the bounding box
frame = read(Vid,frameSt); %Read the frame
framegray = rgb2gray(frame); %Convert to grayscale
points = detectMinEigenFeatures(framegray,'ROI',bbox); %Find minimum
eigen features
points = points.selectStrongest(6); %Select the strongest points
sum = 0; %Initialize sum
if ~isempty(points)
    pointStore = points.Location(:,:); %Create a matrix to store points
    InitialPoints = points.Location(:,:);
    %writel = insertMarker(framegray,points,'+');
    %imshow(writel)
    %figure
    Ymean = mean(pointStore(:,2)); %Find the mean of the Y values
    for i = 1:2
        pointTracker = vision.PointTracker('NumPyramidLevels',2,...
            'BlockSize',[11 11],'MaxIterations',50);
        initialize(pointTracker,InitialPoints,framegray);
        for j = 1:3 %Cycle through proceeding frames
            if i == 1 %Frame buffer (after instance)
                framenum = frameSt + j;
            else %Frame buffer (before instance)
                framenum = frameSt - j;
            end
            if framenum > 0
                if size(points,1) >= 2
                    frame = read(Vid,framenum); %Read frame

```


evalKalmanTracks

This function is utilized if the number of detections for a target is greater than one. Linear regression is used to determine the directions of travel for the target. Direction of travel is determined, classified, and stored for later use.

```
function [status] = evalKalmanTracks(centroidKeeper,BladePitch)

%% Function to find LoBF & spread of data
x = centroidKeeper(:,6);
y = centroidKeeper(:,7);
linReg = fitlm(x,y);
Rsquared = linReg.Rsquared.Ordinary; %R squared
line = linReg.Coefficients.Estimate;
xtest = 100;
ytest = line(2,1)*xtest;
angle = atan2(ytest/xtest);
Difference = abs(angle - BladePitch); %To determine if divergent from
flow field
sum = 0;
for i = 2:length(y)
    sum = (y(i-1) - y(i)) + sum; %Determine direction of travel
end
status = [];
if Rsquared >= 0.500 && Difference >= 15
    if sum < 0
        status(1,1) = 1; %Diverging from flow field
        %Path towards blade
    else
        status(1,1) = 4; %Diverging from flow field
        %Path away from blade
    end
elseif Rsquared >= 0.500 && Difference < 15
    status(1,1) = 2; %Direction similar to flow field
else
    status(1,1) = 3; %Unsuccessful tracking
end
end
```

FinishTracking

This function finishes the tracking process for any remaining targets whose tracking did not end before the final frame of the video.

```
function [reporter] =
FinishTracking(tracks,reporter,centroidKeeper,Vid,BladePitch)
%% Finish tracking any remaining instances at the conclusion of the
video. This prevents any tracks from remaining unevaluated.
```



```

        for u = 1:length(tracks)
            ID = tracks(u).id;
            check = find(centroidKeeper(:,1) == ID); %Find this ID
in storage
            Detections = find(centroidKeeper(check,8) == 1); %Find
which of these are detections
            if max(Detections) == 1 %Only one detection, requiring
backup tracking
                status = KLTpoints_revised(centroidKeeper(check,:),
Vid, BladePitch); %Secondary tracking
                if status == 1 %Angled and towards blade
                    if isempty(reporter)
                        reporter(1,1) = ID; %Record ID
                        reporter(1,2) = 1; %Path type
                    else
                        reporter(end+1,1) = ID; %Record ID
                        reporter(end,2) = 1; %Path type
                    end
                elseif status == 2 %Little to no angle of path
                    if isempty(reporter)
                        reporter(1,1) = ID; %Record ID
                        reporter(1,2) = 2; %Path type
                    else
                        reporter(end+1,1) = ID; %Record ID
                        reporter(end,2) = 2; %Path type
                    end
                elseif status == 3 %Unsuccessfully tracked
                    if isempty(reporter)
                        reporter(1,1) = ID; %Record ID
                        reporter(1,2) = 3; %Path type
                    else
                        reporter(end+1,1) = ID; %Record ID
                        reporter(end,2) = 3; %Path type
                    end
                else %ergo status == 4 %Angled and away from
blade
                    if isempty(reporter)
                        reporter(1,1) = ID; %Record ID
                        reporter(1,2) = 4; %Path type
                    else
                        reporter(end+1,1) = ID; %Record ID
                        reporter(end,2) = 4; %Path type
                    end
                end
            end
        else
            status =
evalKalmanTracks(centroidKeeper(check,:),BladePitch); %Enough
detections for simple path evaluation
            if status == 1
                if isempty(reporter)
                    reporter(1,1) = ID; %Record ID
                    reporter(1,2) = 1; %Path type
                else
                    reporter(end+1,1) = ID; %Record ID
                    reporter(end,2) = 1; %Path type
                end
            elseif status == 2

```

```

        if isempty(reporter)
            reporter(1,1) = ID; %Record ID
            reporter(1,2) = 2; %Path type
        else
            reporter(end+1,1) = ID; %Record ID
            reporter(end,2) = 2; %Path type
        end
    elseif status == 3
        if isempty(reporter)
            reporter(1,1) = ID; %Record ID
            reporter(1,2) = 3; %Path type
        else
            reporter(end+1,1) = ID; %Record ID
            reporter(end,2) = 3; %Path type
        end
    else
        if isempty(reporter)
            reporter(1,1) = ID; %Record ID
            reporter(1,2) = 4; %Path type
        else
            reporter(end+1,1) = ID; %Record ID
            reporter(end,2) = 4; %Path type
        end
    end
end
end
end
end

```

reportAnalysis2

This function is the first stage of the filtering process- its use is to locate groupings of instances which are similar in vertical location, and spaced approximately one rotation apart.

```

function [cleanedCache,dataTrove] =
reportAnalysis2(reporter,centroidKeeper,rotationSpeed)
%% This function was created for analyzing data from the detection
stage
% of the master program. Here false positives will be filtered by
% examining the time and location of reoccurring detections

%Examine all cases for reoccurrence
deleteThese = [];
cleanedCache = [];
ImageInds = [];
sortNum = 1;
%Ensure detections and event classifications were made
if ~isempty(reporter)
    if ~isempty(centroidKeeper)
        %Begin filtering process
        reports = reporter(:, :);
        for i = 1:size(reports,1)
            instance = reports(i,1); %Find where reports is equal to
the instance being examined

```

```

        instanceInd = find(centroidKeeper(:,1) == instance);
%Determine indices of cK
        centroidKeeper2 = centroidKeeper(instanceInd,:);
        DetectInd = find(centroidKeeper2(:,8) == 1);
        centroidKeeper3 = centroidKeeper2(DetectInd,:);
        instanceTime = centroidKeeper3(1,9); %Find time values
        instancePosY = centroidKeeper3(1,7); %Find Y pos
        CheckInd = find(centroidKeeper(:,1) ~= instance &
centroidKeeper(:,8) == 1 ... %Check future bounds
        & ((centroidKeeper(:,9)-instanceTime >= 0.90*rotationSpeed
& centroidKeeper(:,9) ...
        -instanceTime <= 1.10*rotationSpeed) |
(centroidKeeper(:,9)-instanceTime >= 1.89*rotationSpeed &
centroidKeeper(:,9) ...
        -instanceTime <= 2.11*rotationSpeed) |
(centroidKeeper(:,9)-instanceTime >= 2.88*rotationSpeed &
centroidKeeper(:,9) ...
        -instanceTime <= 3.12*rotationSpeed)) & centroidKeeper(:,7)
>= instancePosY - 15 ...
        & centroidKeeper(:,7) <= instancePosY + 15);
        if ~isempty(CheckInd)
            Idz = centroidKeeper(CheckInd,1);
            Idz = unique(Idz);
            if size(Idz,1) >= 2
                for j = 1:size(Idz,1)
                    if isempty(deleteThese)
                        deleteThese(1,1) = instance;
                        deleteThese(1,2) = sortNum;
                        deleteThese(end+1,1) = Idz(j);
                        deleteThese(end,2) = sortNum;
                    elseif ~isempty(deleteThese) && j == 1
                        deleteThese(end+1,1) = instance;
                        deleteThese(end,2) = sortNum;
                        deleteThese(end+1,1) = Idz(j);
                        deleteThese(end,2) = sortNum;
                    else
                        deleteThese(end+1,1) = Idz(j);
                        deleteThese(end,2) = sortNum;
                    end
                end
                sortNum = sortNum + 1;
            end
        end
    end
end
if ~isempty(deleteThese)
    NumberMatches = max(deleteThese(:,2));
    deleteInds = unique(deleteThese(:,1));
    for q = 1:size(deleteInds,1)
        SimInds = find(deleteThese(:,1) == deleteInds(q));
        if SimInds > 1
            Combined = deleteThese(SimInds,2);
            numLabel = min(Combined);
            for w = 1:length(Combined)
                CombInds = find(deleteThese(:,2) == Combined(w));
                for z = 1:length(CombInds)
                    deleteThese(CombInds(z),2) = numLabel;
                end
            end
        end
    end
end

```

```

        end
    end
    deleteInds2 = unique(deleteThese(:,2));
    for s = 1:size(deleteInds2,1)
        for t = 1:size(deleteThese,1)
            if deleteThese(t,2) == deleteInds2(s,1)
                deleteThese(t,2) = s;
            end
        end
    end
    dataTrove = unique(deleteThese,'rows');
    numberOfDelInst = size(deleteInds,1);
else
    numberOfDelInst = 0;
    deleteInds = 0;
    dataTrove = 0;
end
else
    numberOfDelInst = 0;
    deleteInds = 0;
    dataTrove = 0;
end
end
else
    cleanedCache = 0;
    numberOfDelInst = 0;
    deleteInds = 0;
    dataTrove = 0;
end
%msgbox(sprintf('There were %3.3g repeat instances marked for potential
deletion',numberOfDelInst),'Repeat Instances')
%Uncomment above for a readout of the quantity of deletion candidates
if numberOfDelInst > 0 && ~isempty(centroidKeeper)
    ImageInds = centroidKeeper(:,1);
    for k = 1:length(deleteInds)
        deleteSelection = deleteInds(k);
        ImInds = find(ImageInds(:,1) ~= deleteSelection);
        ImageInds = ImageInds(ImInds,1);
    end
    ImageInds = unique(ImageInds);
elseif ~isempty(centroidKeeper)
    ImageInds = unique(centroidKeeper(:,1));
end
if ~isempty(ImageInds)
    for l = 1:size(ImageInds,1)
        ImageInd = ImageInds(l);
        Indz = find(centroidKeeper(:,1) == ImageInd &
centroidKeeper(:,8) == 1);
        Time = centroidKeeper(Indz,9);
        Bbox = centroidKeeper(Indz,2:5);
        ID = centroidKeeper(Indz,1);
        Centroid = centroidKeeper(Indz,6:7);
        for h = 1:size(Indz,1)
            if isempty(cleanedCache)
                cleanedCache(1,1) = ID(h);
                cleanedCache(1,2) = Time(h);
                cleanedCache(1,3:6) = Bbox(h,:);
            end
        end
    end
end

```

```

        cleanedCache(1,7:8) = Centroid(h,:);
    else
        cleanedCache(end+1,1) = ID(h);
        cleanedCache(end,2) = Time(h);
        cleanedCache(end,3:6) = Bbox(h,:);
        cleanedCache(end,7:8) = Centroid(h,:);
    end
end
end
else
    cleanedCache = 0;
end
end

```

RepeatCheck

RepeatCheck is used to compare the similarity between detections. This provides an additional layer of safety to the filtering process. Detections are cropped to the same size, then the SSIM index is used to check similarity [46].

```

function [dataRevised, DifferingDetections] = RepeatCheck(Vid, data,
centroidKeeper)
%% This function uses the SSIM index to compare detections
%This process includes cropping detections to the same size, then
utilizes
%SSIM index
vheight = 1080;
vwidth = 1920;
if data ~= 0
    NumIter = unique(data(:,2));
    scores = [];
    NoRepeats = [];
    for i = 1:length(NumIter)
        currentSet = NumIter(i);
        segmentInd = find(data(:,2) == currentSet);
        for j = 1:size(segmentInd,1)
            for k = 1:size(segmentInd,1)
                compare1 = segmentInd(j,1);
                compare2 = segmentInd(k,1);
                ID1 = data(compare1,1);
                ID2 = data(compare2,1);
                if ID1 ~= ID2
                    if ~isempty(scores)
                        NoRepeats = find(scores(:,1) == ID2 & scores(:,2)
== ID1);
                    end
                    if ~isempty(NoRepeats)
                        for r = 1:length(NoRepeats)
                            scores(end+1,1) = ID2;
                            scores(end,2) = ID1;
                            scores(end,3) = scores(NoRepeats(r), 3);
                        end
                    end
                end
            end
        end
    end
end

```

```

else
    iterations1 = find(centroidKeeper(:,1) == ID1 &
centroidKeeper(:,8) == 1);
    iterations2 = find(centroidKeeper(:,1) == ID2 &
centroidKeeper(:,8) == 1);
    for l = 1:length(iterations1)
    for m = 1:length(iterations2)
        bbox1 = centroidKeeper(iterations1(l),2:5);
        bbox2 = centroidKeeper(iterations2(m),2:5);
        frame1 = centroidKeeper(iterations1(l),9);
        frame2 = centroidKeeper(iterations2(m),9);
        Frame1 = read(Vid,frame1);
        Frame2 = read(Vid,frame2);
        height1 = bbox1(4); height2 = bbox2(4);
        width1 = bbox1(3); width2 = bbox2(3);
        hdiff = abs(height1 - height2);
        wdifff = abs(width1 - width2);
        if height1 > height2
            if (bbox2(4) + bbox2(2) + hdiff) <
vheight
                bbox2(4) = bbox2(4) + hdiff;
            else
                bbox2(2) = bbox2(2) - hdiff;
                bbox2(4) = bbox2(4) + hdiff;
            end
        else
            if (bbox1(4) + bbox1(2) + hdiff) <
vheight
                bbox1(4) = bbox1(4) + hdiff;
            else
                bbox1(2) = bbox1(2) - hdiff;
                bbox1(4) = bbox1(4) + hdiff;
            end
        end
        if width1 > width2
            if (bbox2(3) + bbox2(1) + wdifff) <
vwidth
                bbox2(3) = bbox2(3) + wdifff;

            else
                bbox2(1) = bbox2(1) - wdifff;
                bbox2(3) = bbox2(3) + wdifff;
            end
        else
            if (bbox1(3) + bbox1(1) + wdifff) <
vwidth
                bbox1(3) = bbox1(3) + wdifff;

            else
                bbox1(1) = bbox1(1) - wdifff;
                bbox1(3) = bbox1(3) + wdifff;
            end
        end
        framesect1 =
Frame1(bbox1(2):bbox1(4)+bbox1(2),...

```



```

else
    dataRevised = 0;
    DifferingDetections = 0;
end
end

```

RevisedCache

The purpose of this function is to implement any instances which were shown to be dissimilar in the RepeatCheck function. Such instances are placed back into the matrix used for classification.

```

function [cleanedCacheRevised] = RevisedCache(cleanedCache,
DifferingDetections, centroidKeeper)
%% This function restores any instances that were deemed dissimilar in
the RepeatCheck function
if DifferingDetections ~= 0
    for i = 1:length(DifferingDetections)
        IDtba = DifferingDetections(i);
        IDrungs = find(centroidKeeper(:,1) == IDtba &
centroidKeeper(:,8) == 1);
        for j = 1:length(IDrungs)
            cleanedCache(end+1,1) = IDtba;
            cleanedCache(end,2) = centroidKeeper(IDrungs(j),9);
            cleanedCache(end,3:6) = centroidKeeper(IDrungs(j),2:5);
            cleanedCache(end,7:8) = centroidKeeper(IDrungs(j),6:7);
        end
    end
end
cleanedCacheRevised = cleanedCache;
end

```

zoneCheck

The zoneCheck function divides the image plane into thirds, and checks the vertical coordinates of instances to determine the closest approach of a target to the blade.

```

function zone = zoneCheck(yCoord)
%% This function divides the image plane and categorizes the vertical
location of targets
Zone1 = 0.3*1080;
Zone2 = 0.6*1080;

if yCoord <= Zone1
    zone = 1;
elseif yCoord > Zone1 && yCoord <= Zone2
    zone = 2;
else
    zone = 3;
end
end

```


NegativeSave

NegativeSave provides the necessary tools for storing negative instances to a negative image dataset, and retraining the cascade object detection algorithm. This function only operates if the user selected for the detection algorithm to be updated after processing.

```
function NegativeSave(Vid, data, centroidKeeper)
%% This function provides a retraining method for the cascade object
detection algorithm

%% This section saves negative instances to a negative image set folder
%%The negative instances here are from repeating FPs
if data ~= 0
for i = 1:length(data)
    Inds = find(centroidKeeper(:,1) == data(i,1) & centroidKeeper(:,8)
== 1);
    NumFrames = length(Inds);
    Frames = centroidKeeper(Inds,9);
    Bboxes = centroidKeeper(Inds,2:5);
    for j = 1:NumFrames
        Frame = read(Vid,Frames(j),'native');
        framesect = Frame(Bboxes(j,2):Bboxes(j,4)+Bboxes(j,2),...
%Examine bbox sections
        Bboxes(j,1):Bboxes(j,3)+Bboxes(j,1),:);
        name =
sprintf('D:/MATLAB/toolbox/vision/visiondata/Negatives/%s%d_%d.png','ne
gImageR2',i,j);
        imwrite(framesect,name);
    end
end
end
Cases = input('Please enter any IDs of false-positives: ');
Cases = transpose(Cases);
%% This section saves negative instances to a negative image set folder
%%The negative instances here are from user inputted FPs
if ~isempty(Cases)
    for i = 1:length(Cases)
        Inds2 = find(centroidKeeper(:,1) == Cases(i,1) &
centroidKeeper(:,8) == 1);
        NumFrames2 = length(Inds2);
        Frames2 = centroidKeeper(Inds2,9);
        Bboxes2 = centroidKeeper(Inds2,2:5);
        for j = 1:NumFrames2
            Frame = read(Vid,Frames2(j),'native');
            framesect = Frame(Bboxes2(j,2):Bboxes2(j,4)+Bboxes2(j,2),...
%Examine bbox sections
            Bboxes2(j,1):Bboxes2(j,3)+Bboxes2(j,1),:);
            name =
sprintf('D:/MATLAB/toolbox/vision/visiondata/Negatives/%s%d_%d.png','Ca
sesNegImageR2',i,j);
            imwrite(framesect,name);
        end
    end
end
```

```

        end
    end
end

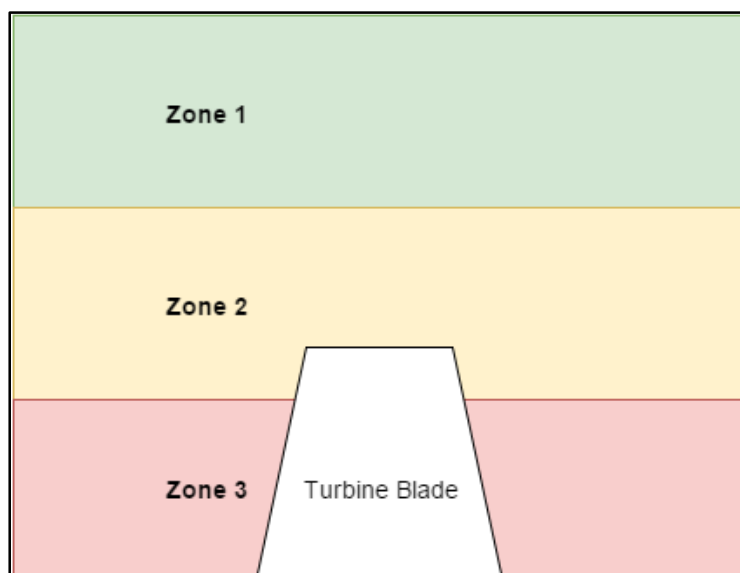
%Retraining section
if ~isempty(Cases) | data ~= 0
load('Documents/MATLAB/NewAlbaLabel.mat');
NewNegatives =
fullfile(matlabroot,'toolbox','vision','visiondata','Negatives');
trainCascadeObjectDetector('Test_01072016_1st_Regen.xml',labelingSession.ImageSet.ROIBoundingBoxes,NewNegatives,'FalseAlarmRate',0.4,'NumCascadeStages',20);
end

```

Appendix F: Collision Likelihood

The following table provides a reference for how avian interaction information is utilized to produce collision results. Each combination was considered in the context of the blade-mounted camera, in order to subjectively determine the likelihood that a collision occurred.

Zones (proximity to blade):



Bird Path: As described in sub-section Trajectory Classification, section Tracking, in Chapter 5

Contrasting object in front of blade face: Y/N

Type of Path	Contrasting Object Infront of Blade Face	Zone	Resulting Classification
1	Y	1	Strong
1	Y	2	Very Strong
1	Y	3	Very Strong
1	N	1	Moderate
1	N	2	Moderate
1	N	3	Strong
2	Y	1	Moderate
2	Y	2	Moderate
2	Y	3	Strong
2	N	1	Low
2	N	2	Low
2	N	3	Moderate
3	Y	1	Moderate
3	Y	2	Strong

3	Y	3	Strong
3	N	1	Low
3	N	2	Moderate
3	N	3	Moderate
4	Y	1	Low
4	Y	2	Moderate
4	Y	3	Strong
4	N	1	Low
4	N	2	Low
4	N	3	Moderate

Appendix G: GUI Code

The following is the code for the GUI program. This code was generated via GUIDE [47]. The raw figure created for the GUI is presented last. Tracking and dealing with multiple detected objects was adapted from [42].

Primary Code

```
function varargout = BirdTracker_V1_1(varargin)
% BIRDTRACKER_V1_1 MATLAB code for BirdTracker_V1_1.fig
%   BIRDTRACKER_V1_1, by itself, creates a new BIRDTRACKER_V1_1 or
%   raises the existing
%   singleton*.
%
%   H = BIRDTRACKER_V1_1 returns the handle to a new
%   BIRDTRACKER_V1_1 or the handle to
%   the existing singleton*.
%
%   BIRDTRACKER_V1_1('CALLBACK',hObject,eventData,handles,...) calls
%   the local
%   function named CALLBACK in BIRDTRACKER_V1_1.M with the given
%   input arguments.
%
%   BIRDTRACKER_V1_1('Property','Value',...) creates a new
%   BIRDTRACKER_V1_1 or raises the
%   existing singleton*. Starting from the left, property value
%   pairs are
%   applied to the GUI before BirdTracker_V1_1_OpeningFcn gets
%   called. An
%   unrecognized property name or invalid value makes property
%   application
%   stop. All inputs are passed to BirdTracker_V1_1_OpeningFcn via
%   varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only
%   one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help BirdTracker_V1_1

% Last Modified by GUIDE v2.5 16-Jan-2016 17:44:52

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn', @BirdTracker_V1_1_OpeningFcn, ...
                  'gui_OutputFcn',  @BirdTracker_V1_1_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
```

```

        'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before BirdTracker_V1_1 is made visible.
function BirdTracker_V1_1_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to BirdTracker_V1_1 (see VARARGIN)
clc
% Choose default command line output for BirdTracker_V1_1
handles.output = hObject;
% Remove tick marks from video plot
set(handles.axes1, 'ytick', [], 'xtick', []);
handles.stopper = 0;
%% Update handles structure
guidata(hObject, handles);

% UIWAIT makes BirdTracker_V1_1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = BirdTracker_V1_1_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)
global filename
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
filename = get(hObject, 'String');

% Hints: get(hObject, 'String') returns contents of edit1 as text

```

```

%         str2double(get(hObject,'String')) returns contents of edit1 as
a double

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit3_Callback(hObject, eventdata, handles)
global CONA
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
CONA = get(hObject,'String');
CONA = str2double(CONA);
% Hints: get(hObject,'String') returns contents of edit3 as text
%         str2double(get(hObject,'String')) returns contents of edit3 as
a double

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit2_Callback(hObject, eventdata, handles)
global MN
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
MN = get(hObject,'String');

```

```

MN = str2double(MN);

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as
a double

% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit4_Callback(hObject, eventdata, handles)
global MT
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
MT = get(hObject,'String');
MT = str2double(MT);
% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as
a double

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton1 (START/RESET).
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%% Button Controls
set(handles.stopper,'UserData',0);
cla(handles.axes2)

```



```

axes(handles.axes2)
set(handles.axes2,'Ydir','reverse');
xlim([0,1920]);
ylim([0,1080]);
grid on

global filename MN CONA MT
%% Tracker Program
%Launch detector
detector = vision.CascadeObjectDetector('AlbatrossDetector3.xml');
%Set detector parameters
detector.MergeThreshold = MT;
%Launch blob analysis
blob = vision.BlobAnalysis;
blob.AreaOutputPort = false;
blob.BoundingBoxOutputPort = false;
blob.MinimumBlobArea = 8;
blob.MaximumCount = 1;
vision.KalmanFilter; %Initialize the Kalman filter
% Setup tracks structure with fields for ID's, bounding boxes, age,
etc.
tracks = struct(...
    'id', {}, ...
    'bbox', {}, ...
    'kalmanFilter', {}, ...
    'age', {}, ...
    'totalVisibleCount', {}, ...
    'consecutiveInvisibleCount', {});

nextId = 1;
centroidKeeper = [];
count = 0;
color = ['r','b','k','g','c','m','y'];
% References to MATLAB's Multiple Object Tracking framework [42] will
appear
% throughout code.
%% Load Video
Vidin = VideoReader(filename); %Call global variable
numOfFrames = Vidin.NumberOfFrames;
%% loop for reading frames
for j = 1:numOfFrames
    hold(handles.axes2,'on')
    if get(handles.stopper,'UserData') %Pause button
        pause on
        uiwait
        pause off
    end
    hold(handles.axes2,'on')
    %Read frames
    img = read(Vidin,j);
    framegray = rgb2gray(img); %Convert frame to intensity
    %Detect potential birds
    bbox = step(detector,img); %Creation of bounding box
    [R,C] = size(bbox); %Find size of bbox struct
    if R > 0
        %% Combine overlapping bounding boxes (NOTE: only for two bboxes)
        [bboxC] = bboxCombine(bbox);
        %% Find centroids

```

```

[R2,C2] = size(bboxC);
centroid = zeros(R2,2);
for i = 1:R2
    framesect = framegray(bboxC(i,2):bboxC(i,4)+bboxC(i,2),...
%Examine bbox sections
        bboxC(i,1):bboxC(i,3)+bboxC(i,1));
    level = graythresh(framesect); %Determine local threshold value
    framesectBW = im2bw(framesect,level); %Convert section to
binary
    sectBW = imcomplement(framesectBW); %Birds are usually dark
compared to background- inverse
    centroid(i,:) = step(blob,sectBW); %Locate the centroid
    centroid(i,1) = centroid(i,1) + bboxC(i,1);
    centroid(i,2) = centroid(i,2) + bboxC(i,2);
end
end
%% Predict new locations REF: [42] (Framework)
for j = 1:length(tracks)
    bboxA = tracks(j).bbox;
    predictedCentroid = predict(tracks(j).kalmanFilter);
    predictedCentroid = int16(predictedCentroid) -
int16(bboxA(3:4)/2);
    tracks(j).bbox = [predictedCentroid, bboxA(3:4)];
end
%% Assignment cost analysis REF: [42]
if R > 0
    nTracks = length(tracks);
    nDetections = size(centroid, 1);
    cost = zeros(nTracks, nDetections);
    for k = 1:nTracks
        cost(k,:) = distance(tracks(k).kalmanFilter, centroid);
    end
    costOfNonAssignment = CONA;
    [assignments, unassignedTracks, unassignedDetections] =
assignDetectionsToTracks(cost, costOfNonAssignment);
end

%% Updating assigned tracks REF: [42]
if R > 0
    numAssignedTracks = size(assignments, 1);
    for l = 1:numAssignedTracks
        trackIdx = assignments(l, 1);
        detectionIdx = assignments(l, 2);
        centroids = centroid(detectionIdx, :);
        bboxB = bboxC(detectionIdx, :);
        tracks(trackIdx).bbox = bboxB;
        correct(tracks(trackIdx).kalmanFilter, centroids);
        tracks(trackIdx).age = tracks(trackIdx).age + 1;
        tracks(trackIdx).totalVisibleCount =
tracks(trackIdx).totalVisibleCount + 1;
        tracks(trackIdx).consecutiveInvisibleCount = 0;
        centroidKeeper(end+1,1) = tracks(trackIdx).id;
        centroidKeeper(end,2:3) = centroids;

scatter(centroids(1,1),centroids(1,2),color(tracks(trackIdx).id),'Paren
t',handles.axes2)
drawnow

```

```

        hold(handles.axes2, 'on')
    end
end
%% Updating unassigned tracks REF: [42]
if R > 0
    for o = 1:length(unassignedTracks)
        ind = unassignedTracks(o);
        tracks(ind).age = tracks(ind).age + 1;
        tracks(ind).consecutiveInvisibleCount =
tracks(ind).consecutiveInvisibleCount + 1;
        centroidKeeper(end+1,1) = tracks(ind).id;
        centroidKeeper(end,2:3) =
predict(tracks(ind).kalmanFilter);

scatter(centroidKeeper(end,2),centroidKeeper(end,3),color(centroidKeeper(end,1)), 'Parent',handles.axes2)
        drawnow
        hold(handles.axes2, 'on')
    end
elseif ~isempty(tracks) && R == 0
    % this is necessary for maintaining Kalman filter when
detections
    % do not occur
    for z = 1:length(tracks)
        tracks(z).age = tracks(z).age + 1;
        tracks(z).consecutiveInvisibleCount =
tracks(z).consecutiveInvisibleCount + 1;
        centroidKeeper(end+1,1) = tracks(z).id;
        centroidKeeper(end,2:3) = predict(tracks(z).kalmanFilter);

scatter(centroidKeeper(end,2),centroidKeeper(end,3),color(centroidKeeper(end,1)), 'Parent',handles.axes2)
        drawnow
        hold(handles.axes2, 'on')
    end
end
end
%% Deleting lost tracks REF: [42]
if ~isempty(tracks)
    invisibleForTooLong = 18; %was set to 15
    ageThreshold = 25;
    ages = [tracks(:).age];
    totalVisibleCounts = [tracks(:).totalVisibleCount];
    visibility = totalVisibleCounts ./ ages;
    lostInds = (ages < ageThreshold & visibility < 0.1) |
[tracks(:).consecutiveInvisibleCount] >= invisibleForTooLong;
    tracks = tracks(~lostInds);
end
%% Create some new tracks REF: [42]
if R > 0
    centroid = centroid(unassignedDetections, :);
    for p = 1:size(centroid, 1)
        centroids = centroid(p,:);
        bboxA = bboxC(p,:);
        %Create a Kalman filter object
        kalmanFilter =
configureKalmanFilter('ConstantVelocity',centroids,[200, 50], [100,
25], MN);

```

```

        %Create a new track
        newTrack = struct('id', nextId, 'bbox', bboxA, 'kalmanFilter',
kalmanFilter, 'age', 1, 'totalVisibleCount', 1,
'consecutiveInvisibleCount', 0);
        %Add it to the array of tracks
        tracks(end + 1) = newTrack;
        if isempty(centroidKeeper)
            centroidKeeper(1,1) = nextId;
            centroidKeeper(1,2:3) = centroids;

scatter(centroid(1,1),centroid(1,2),color(nextId),'Parent',handles.axes
2)

            drawnow
            hold(handles.axes2,'on')
        else
            centroidKeeper(end+1,1) = nextId;
            centroidKeeper(end,2:3) = centroids;

scatter(centroid(1,1),centroid(1,2),color(nextId),'Parent',handles.axes
2)

            drawnow
            hold(handles.axes2,'on')
        end
        %Increment the next id
        nextId = nextId + 1;
    end
end
%% Display the results. For next 5 lines, REF: [42]
if ~isempty(tracks)
    ids = int32([tracks(:).id]);
    labels = cellstr(int2str(ids));
    bboxNotation = cat(1, tracks.bbox);
    frameNotated = insertObjectAnnotation(framegray,
'rectangle', bboxNotation, labels);
    framegray = im2uint16(frameNotated);
end
framegray = im2uint16(framegray);
imshow(framegray,'parent',handles.axes1)
drawnow

figureSnap = getframe(gcf);
%writeVideo(vid,figureSnap)
end
%close(vid);
% --- Executes on button press in pushbutton2 (STOP).
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject      handle to pushbutton2 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
set(handles.stopper,'UserData',1);

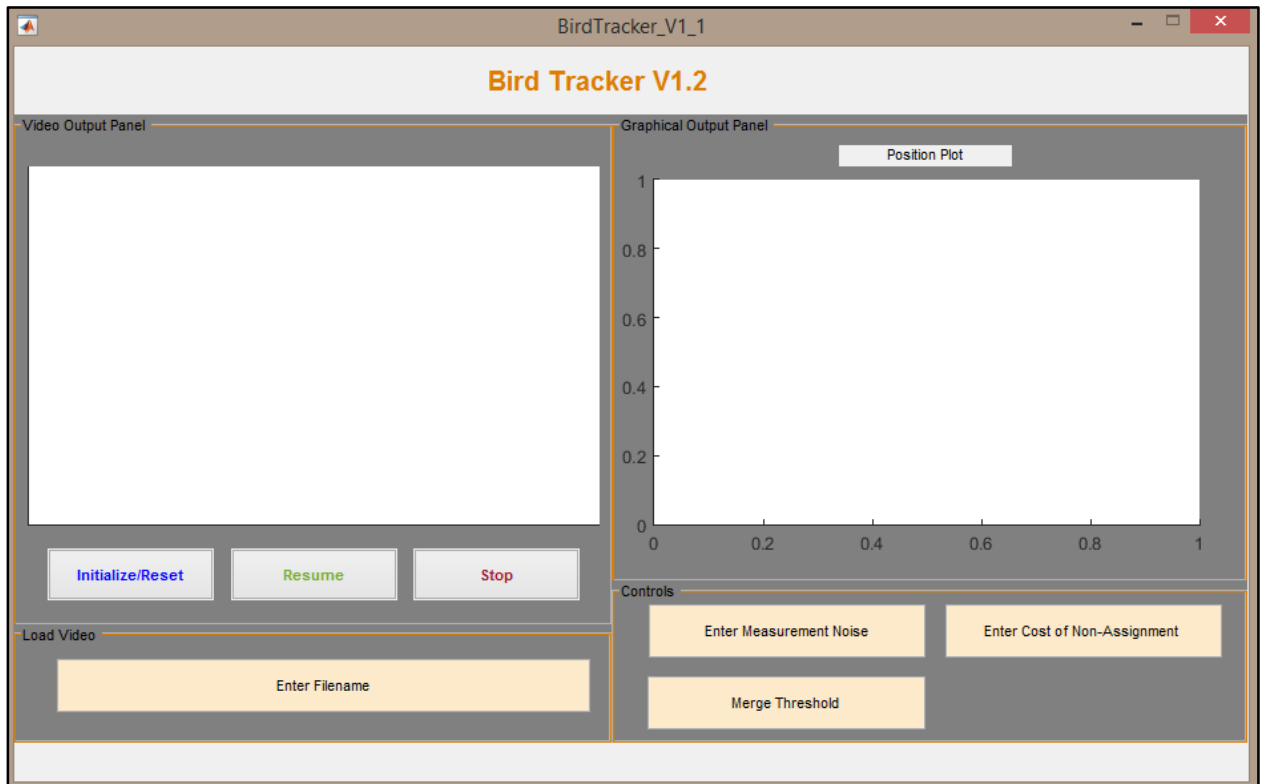
% --- Executes on button press in pushbutton3 (Resume).
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject      handle to pushbutton3 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB

```

```
% handles      structure with handles and user data (see GUIDATA)
set(handles.stopper,'UserData',0);
uiresume
```

GUI Figure

Below is the figure used for the GUI; this figure was created in guide [47]. The version was rolled to V1.2 after updating the primary tracking framework, and adding a Merge Threshold input box.



Appendix H: Blade Tracking Code

In this appendix the MATLAB code for the blade tracking program is presented. Satellite functions proceed the main code, and are shown in the order that they are called. [57] provided guidance on utilizing the point tracking function. [44] gives details on the point tracking function itself.

Primary Code

```
clear all
close all
clc
clf

%% Kanade-Lucas-Tomasi feature tracker for the monitoring of wind
turbine
%blade deflection.

% Functions will be used for edge finding & point tracking.

% Setup video. In this case a premade video file will be loaded &
played...
% In the final version the video feed will be realtime
Vidin = VideoReader('Blade_Animation_2.avi');
vidStart = 500; %Start later in test
numOfFrames = Vidin.NumberOfFrames;
markerInserter = vision.MarkerInserter('BorderColor','black');
%uncomment below for video writing
%vid = VideoWriter('Blade_Animation_2');
%vid.FrameRate = 18;
%open(vid)
position = [1 50]; %Position on frame for annotation
position2 = [1 150]; %Secondary position on frame for annotation

%User input of beam length
Length = 1068.4; %mm
(-3); %m
CellHorz = 6.35*10(-6); %m
CellVert = 7.4*10(-6); %m
PixelHorz = 720; %Pix (cropped by USB capture device)
PixelVert = 480; %Pix (cropped by USB capture device)
SensHorz = CellHorz*PixelHorz; %m
SensVert = CellVert*PixelVert; %m
%Calculate FOV height and width
AngleHorz = 2*atand(SensHorz/(2*FL)); %Degrees
AngleVert = 2*atand(SensVert/(2*FL)); %Degrees

%Tip location calculations
Height = 2*tand(AngleVert/2)*Length;
```

```

tipHeight = [];
frame = [];
tick(1,1) = 1;
%For-loop to cycle through frames
for num = vidStart:numOfFrames
    frame = read(Vidin,num); %Load frame
    framegray = rgb2gray(frame); %Convert to grayscale
    if num == vidStart
        [points, Tracker] = pointFinder(framegray);
        pointsRef = min(points.Location(:,2)); %Minimum equates to
highest vertical point on frame
        refLine = (PixelVert/pointsRef)*Height;
    end
    pointsPre = pointTracker(Tracker, framegray, points);
    points = int32(pointsPre);
    outline = sortrows(points,1);
    tip = min(pointsPre(:,2));
    tipHeightAbs = (PixelVert/tip)*Height;
    tipHeightn = tipHeightAbs - refLine;
    tipHeight(num-(vidStart-1),1) = round(tipHeightn,3);
    frameNum(num-(vidStart-1),1) = num;
    Annotized = step(markerInserter, framegray, points);
    Notated = insertText(Annotized, position, tipHeight(num-(vidStart-
1),1), 'AnchorPoint',
'LeftBottom', 'FontSize', 40, 'BoxColor', 'white', 'BoxOpacity', 0.9);
    imshow(Notated)
    %uncomment below for simulated beam shape
    %plt = plot(outline(:,1),outline(:,2));
    %set(gca,'Ydir','reverse')
    %axis([0 720 0 480])
    drawnow
    %uncomment below for live feed during processing
    %hold(gca,'off')
    %subplot(1,2,2)
    %vid = imshow(Notated2);
    %plot(tick(:,1),tipHeight(:,1),'k')
    %axis([1 numOfFrames -2 25])
    %grid on
    %xlabel('Frame Number')
    %ylabel('Tip Displacement [mm]')
    %title('Tip Displacement vs Frame Number')
    %drawnow
    %hold(gca,'off')
    tick(tick+1,1) = tick + 1;
    %uncomment below for video writing
    %figureSnap = getframe(gcf);
    %writeVideo(vid,Notated);
end
%close(vid);
figure
plot(frameNum(:,1),tipHeight(:,1),'k') %plot results
grid on
axis([vidStart numOfFrames -5 25])
title('Tip Height versus Frame Number')

%input vibration plateaus (system left untouched during testing)
st2 = input('start 2: ');

```

```

nd2 = input('end 2: ');
st6 = input('start 6: ');
nd6 = input('end 6: ');
st10 = input('start 10: ');
nd10 = input('end 10: ');
st20 = input('start 20: ');
nd20 = input('end 20: ');
%Average values in that region
Ave_2 = sum(tipHeight(st2:nd2,1))/size(tipHeight(st2:nd2,1),1)
Ave_6 = sum(tipHeight(st6:nd6,1))/size(tipHeight(st6:nd6,1),1)
Ave_10 = sum(tipHeight(st10:nd10,1))/size(tipHeight(st10:nd10,1),1)
Ave_20 = sum(tipHeight(st20:nd20,1))/size(tipHeight(st20:nd20,1),1)

```

Satellite Functions

pointFinder

This function locates points on the blade, and initializes the point tracking process. [57] provided guidance on using the point tracking function. [44] gives details on the point tracking function itself.

```

function [points, Tracker, validity] = pointFinder(framegray)
%allow user to specify bbox
subplot(1,2,1)
imshow(framegray);
rect = imrect; %Specify bbox
drawnow
wait(rect); %Wait for user input
bbox = getPosition(rect);
bbox = int32(bbox);
points = detectHarrisFeatures(framegray,'ROI',bbox); %find points
points = points.selectStrongest(8);
Tracker = vision.PointTracker('NumPyramidLevels',2);
initialize(Tracker, points.Location, framegray);
markerInserter = vision.MarkerInserter('BorderColor','white');
pointsToTrack = int32(points.Location); %Convert to integer (pixels)
Annotized = step(markerInserter,framegray,pointsToTrack);
hold on
subplot(2,1,1)
imshow(Annotized)
drawnow
hold on
validity = zeros(size(points.Location,1),1) + 1; %Validity of one
end

```

pointTracker

This function steps the point tracking algorithm, and checks the validity of the remaining points.

[57] provided guidance on using the point tracking function. [44] gives details on the point tracking function itself.


```

function [points, validity] = pointTracker(Tracker, framegray, points,
validity)
    if size(points,1) >= 2
        [points, validity] = step(Tracker, framegray); %Step function
        DelInd = find(validity ~= 0); %Find points to keep
        points = points(DelInd,:); %Keep valid points
        if ~isempty(points)
            setPoints(Tracker,points); %Continue with valid points
        else
            disp('Need to recalibrate') %No more valid points
        end
    else
        disp('Need to recalibrate')
        %Here lies the future location of recalibration
    end
end

```

Appendix I: Blade Tracking Results

The results of the blade tracking trials are presented here. Shown for each trial are the average values for each displacement. Proceeding the results, the statistical analysis is shown, which includes standard deviation and t-testing.

Tip Deflection					
Trial 1					
True Value:	2	6	10	20	mm
Estimated Value:	1.99	6.05	10.17	21.45	mm
Trial 2					
True Value:	2	6	10	20	mm
Estimated Value:	1.74	5.33	9.30	20.59	mm
Trial 3					
True Value:	2	6	10	20	mm
Estimated Value:	1.85	5.71	9.84	21.04	mm
Trial 4					
True Value:	2	6	10	20	mm
Estimated Value:	1.65	5.62	9.63	20.06	mm
Trial 5					
True Value:	2	6	10	20	mm
Estimated Value:	1.66	5.40	9.17	20.45	mm
Trial 6					
True Value:	2	6	10	20	mm
Estimated Value:	1.80	5.58	9.33	20.67	mm
Trial 7					
True Value:	2	6	10	20	mm
Estimated Value:	1.79	5.56	9.58	20.82	mm
Trial 8					
True Value:	2	6	10	20	mm
Estimated Value:	1.68	5.57	9.55	20.50	mm

Trial 9					
True Value:	2	6	10	20	mm
Estimated Value:	1.76	5.64	9.85	21.00	mm
Trial 10					
True Value:	2	6	10	20	mm
Estimated Value:	1.86	6.01	10.03	21.30	mm
Trial 11					
True Value:	2	6	10	20	mm
Estimated Value:	1.52	5.28	9.03	19.92	mm
Trial 12					
True Value:	2	6	10	20	mm
Estimated Value:	1.92	5.83	9.86	19.83	mm
Trial 13					
True Value:	2	6	10	20	mm
Estimated Value:	1.99	6.02	9.92	21.25	mm
Trial 14					
True Value:	2	6	10	20	mm
Estimated Value:	1.71	5.67	9.59	21.13	mm
Trial 15					
True Value:	2	6	10	20	mm
Estimated Value:	1.46	5.51	9.69	20.99	mm
Trial 16					
True Value:	2	6	10	20	mm
Estimated Value:	1.69	5.83	9.81	20.97	mm
Trial 17					
True Value:	2	6	10	20	mm
Estimated Value:	2.08	6.02	9.99	21.27	mm
Trial 18					
True Value:	2	6	10	20	mm
Estimated Value:	1.85	5.85	10.05	21.32	mm
Trial 19					

True Value:	2	6	10	20	mm
Estimated Value:	1.67	5.28	9.35	20.75	mm
Trial 20					
True Value:	2	6	10	20	mm
Estimated Value:	1.55	5.34	8.96	20.36	mm

Below is the statistical analysis for the blade deflection program. Note that the final P values for each displacement were significantly low, so the null hypothesis that there is no difference between the means and measured values can be rejected. P values were obtained from GraphPad [58].

Disp. (mm)	Averages (mm)	Std Dev (mm)
2	1.76	0.16
6	5.65	0.26
10	9.64	0.35
20	20.78	0.48

Confidence Intervals (99%)				
Disp. (mm)	Stdev (mm)	Alpha	Smpl Size	Confid. (mm)
2	0.16	0.01	20	0.104534
6	0.26	0.01	20	0.164123
10.00	0.35	0.01	20	0.224033
20	0.48	0.01	20	0.306322

Disp. (mm)	T-Value	P Value
2	-6.52479	<0.0001
6	-6.01643	<0.0001
10	-4.65887	0.0002
20	7.319152	<0.0001