THREE-DIMENSIONAL FRAGMENT TRACKING AND SIZE ESTIMATION USING STEREO FOCUSED SHADOWGRAPHY

by

Sean Palmer

Submitted in Partial Fulfillment
of the Requirements for the Degree of
Masters of Science in Mechanical Engineering
with Specialization in Explosives Engineering

NEW MEXICO TECH
SCIENCE • ENGINEERING • RESEARCH UNIVERSITY

New Mexico Institute of Mining and Technology
Socorro, New Mexico
September, 2022

This thesis is dedicated to Veronica Espinoza for supporting me in all things in life. I would like to thank Dr. Michael Hargather for his guidance and for taking a chance on me. I appreciate all of the members of the Shock and Gas Dynamics Laboratory for their continued friendship and aid throughout my time as a graduate student and onward. As always, I am grateful for my family for their unconditional support.

Sean Palmer
New Mexico Institute of Mining and Technology
September, 2022

ACKNOWLEDGMENTS

This research was performed under funding from PO N0017419P0260 from the Naval Surface Warfare Center Indian Head Division, subaward W911SR-14-2-0001 W911SR20F0018 #0041 from MSI STEM Research and Development Consortium.

This thesis was typeset with LaTeX[1] by the author.

[1]The LaTeX document preparation system was developed by Leslie Lamport as a special version of Donald Knuth's TeX program for computer typesetting. TeX is a trademark of the American Mathematical Society. The LaTeX macro package for the New Mexico Institute of Mining and Technology thesis format was written by John W. Shipman.

ABSTRACT

High-speed stereo focused shadowgraphy visualizes fragment projections resulting from reactive material specimens undergoing high-velocity impacts on a steel anvil. The diverging light in the conical sections of the stereo shadowgraph systems intersect at a test section at the impact point and allows for the depth information of the incident projectile and the resulting fragments to be extracted. Image segmentation techniques allow for centroids and pixel areas to be extracted for fragments in each camera view. Two-dimensional Kalman filtering and assignment algorithms were applied for simultaneous tracking in each camera view for a series of high-speed images. The identification of the same fragments in each camera view was determined via the exploitation of the epipolar geometry defined from the orientation of the shadowgraph systems. The triangulation of the fragment trajectories from each camera view is used to reconstruct the three-dimensional trajectory for each fragment. Fragment sizes are estimated via equivalent spherical diameter assumptions and to each tumbling fragment. Bivariate histograms describing the result of the fragmentation behavior of the impact-fragmented RM projectiles are constructed from the simultaneously measured fragment sizes and three-dimensional velocities.

Keywords: Stereo Shadowgraphy, Kalman Filter, Impact-Fragmentation, 3D reconstruction, Image segmentation

CONTENTS

LIST OF TABLES

LIST OF FIGURES

This thesis is accepted on behalf of the faculty of the Institute by the following committee:

Michael J. Hargather

Academic and Research Advisor

Jamie Kimberley

Demitrios Stamatis

I release this document to the New Mexico Institute of Mining and Technology.

Sean Palmer | August 15, 2022

NOTATION, NOMENCLATURE, AND ABBREVIATIONS

Chemistry

Al	Aluminum
Bi2O3	Bismuth(III) oxide
RM	Reactive Material
W	Tungsten

Other Abbreviations

1D	One-Dimensional
2D	Two-Dimensional
3D	Three-Dimensional
BOS	Background Oriented Schlieren
CMOS	Complementary Metal Oxide Semiconductor
DIA	Dynamic Image Analysis
DIC	Digital Image Correlation
PIV	Particle Image Velocimetry
PTV	Particle Tracking Velocimetry
TMD	theoretical maximum density

Variables

$\mathbf{A}$	transition matrix
$\mathbf{B}$	control matrix
$\mathbf{H}$	measurement or observation matrix
$\mathbf{I}$	Identity matrix
$\mathbf{K}$	Kalman Gain factor
$\mathbf{P}'$	measured image position coordinate in camera 2

$\mathbf{P_{k-1}}$	a priori state error covariance at time step k-1
$\mathbf{P_k}$	a posteriori state error covariance at time step k-1
$\mathbf{P}$	measured image position coordinate in camera 1
$\mathbf{Q}$	estimate or process noise covariance matrix
$\mathbf{R_k}$	measurement noise covariance matrix
$\mathbf{u}$	control input
$\mathbf{x_k}$	a posteriori state estimate at time k
$\mathbf{x_{k-1}}$	a priori state estimate at time step k-1
$\mathbf{x_{meas}}$	measurement vector
$\mathbf{z_{res}}$	measurement residual
δA_{px}	uncertainty with respect to pixel area
δC_s	uncertainty with respect to spatial calibration scale
δd_e	uncertainty with respect to equivalent spherical diameter
δd_{px}	uncertainty with respect to equivalent diameter in mm
δp	uncertainty with respect to distance in pixels
δr_{px}	reprojection error
Δt	time interval between measurements
δt	uncertainty with respect to time
δv_{3D}	uncertainty with respect to 3D velocity
δx_s	uncertainty with respect to scale distance
δx_{3D}	uncertainty with respect to the x 3D distance
δy_{3D}	uncertainty with respect to the y 3D distance
δz_{3D}	uncertainty with respect to the z 3D distance
$\dot{x}$	x velocity in state estimate
$\dot{y}$	y velocity in state estimate
ϵ_y	refractive angle
∂n	change in refractive index

∂x	change in the horizontal spatial component
∂y	change in the vertical spatial component
∂z	change in the depth of the refraction object
σ_x	x measurement error standard deviation
σ_y	y measurement error standard deviation
$\hat{\mathbf{P}}'$	reprojected image position coordinate in camera 2
$\hat{\mathbf{P}}$	reprojected image position coordinate in camera 1
A_{px}	pixel area
C_c	Camera Coordinate System
C_s	spatial calibration scale
$d(*,*)$	Euclidean distance between two points
D'	diameter of field of view at position of refractive object
d_e	equivalent spherical diameter in mm
d_{px}	equivalent spherical diameter in mm
e'	epipole in opposite camera
h_i	image height
h_o	object height
l'	epipolar line in opposite camera
L'	length from refractive object to camera
L_i	object image
L_o	object distance
L_{total}	length from lens to camera
O_A	Optical center of the camera A
O_B	Optical center of the camera B
O_c	Optical center of a camera
O_w	Optical center for world coordinate system

P_c	Coordinate of a 3D point in the camera coordinate system
P_w	Coordinate of a 3D point in the world coordinate system
v_{3D}	3D velocity
W_c	World Coordinate System
x_1	An x coordinate for the Camera 1 image
x_2	An x coordinate for the Camera 1 image
x_A	2D vector position point in the camera A
x_k	x position in state estimate
x_s	scale distance in mm
y_1	An x coordinate for the Camera 1 image
y_2	An x coordinate for the Camera 1 image
y_k	y position in state estimate
C	reprojection error cost function
D	diameter of lens
e	epipole
F	Fundamental Matrix
f	focal length
k	time step
l	epipolar line
L	length from lens to refractive object
m	magnification
n	refractive index
p	distance in pixels
R	rotation matrix
S	refractive object
T	translation matrix
x	x direction in schlieren diagram
y	y direction in schlieren diagram

CHAPTER 1

INTRODUCTION

1.1 Reactive Materials and Fragmentation Behavior

Traditional munition cases are made of steel which is fragmented and accelerated outward during detonation. Although these steel fragments can deliver kinetic energy and impulse to a target, there is a desire to increase the energy delivered to the target. One method to enhance the energy on target is to replace the steel case with a material that will impact a target and then combust. Reactive materials (RMs) is a modern term for these sort of combustible materials that can enhance energy delivery in munition systems.

Reactive materials are consolidated powder specimens that are pressed into spheres, cubes, or cylinders to be used as projectiles. The projectiles may be comprised of a single metal such as aluminum [1, 2] or be a composite consisting in multiple metals such as bi-metallic composites of aluminum and tungsten[3] or other composites such as Al:PTFE[4, 5] for example. Most published studies have focused on single component consolidated powder RM specimens, including aluminum spheres [1, 2, 6] and cylinders [7] and zinc cylinders [8].

Fragmentation behavior of reactive materials is an active area of research to improve combustion and kinetic energy of small fragments [9, 10, 11]. Most studies explore fragmentation from high velocity impact tests [1, 2, 6, 8]. Many of these studies are performed via high velocity impacts of RM projectiles on thin metallic plates [1, 2, 6, 7, 8] or an anvil [7]. RM specimens are frequently accelerated to high velocities using via gun-launch using either gas guns [6, 7, 8] or powder guns [1, 2, 4]. During high-velocity impact, strain rates often induce brittle material behavior in the reactive materials due to dynamic loading [1, 7, 8, 12].

1.2 Schlieren and shadowgraph imaging

Refractive imaging techniques are utilized for visualization of the refraction of light rays through a medium. Schlieren imaging is a refractive imaging technique used to visualize the refractive index gradient of a medium [13]. Figure 1.1 is a diagram of a typical focused-schlieren setup in which a collimated light beam

Light ray curvatures are defined by

$$\frac{\partial^2 x}{\partial z^2} = \frac{1}{n}\frac{\partial n}{\partial z} \tag{1.1}$$

in which n is the refractive index, x is a horizontal spatial component, y is a vertical spatial component, and z is the depth of the refraction object in the optical test section. The integrated form represents the schlieren visualization of the first derivative of the refractive index

$$\epsilon_x = \frac{1}{n}\int \frac{\partial n}{\partial x}\partial z \tag{1.2}$$

where ϵ_y is the refractive angle in x direction, pictured in Figure 1.1. Traditional focused shadowgraph and schlieren imaging utilizes a parabolic lens to collimate light from a point source which is then refocused with a second parabolic lens to a focal point. The collimated light beam constitutes the optical test section in which a schlieren object causes changes in a refractive index field. Schlieren images visualize refractive index gradients using a knife edge placed at the focal point such that the light is bent toward the high refractive index or higher density in the test section. Schlieren images visualize variations in refractive index as variations of grayscale intensity in the final image.

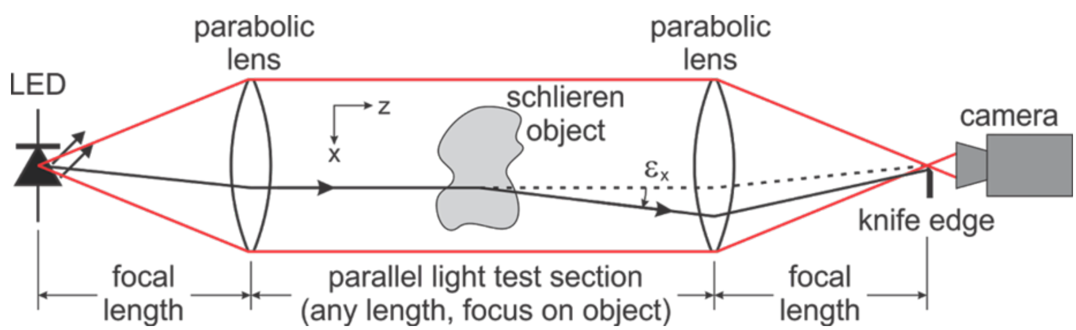


Figure 1.1: A diagram of the traditional focused-schlieren setup using two lenses to achieve a collimated light beam.

Shadowgraph imaging is a refractive imaging technique used to visualize the Laplacian, or the second derivative, of the refractive index field in a medium. This form of refractive imaging is useful for visualizing sharp disturbances and changes in the refractive index, such as shockwaves and turbulent structures, and gas discontinuities in the test section. A typical focused-shadowgraph system is the same as a schlieren imaging system setup but without the knife-edge. The parallel light in the test section of a traditional focused-shadowgraph system allows for true projections of for size analysis of objects via spatial calibration of the optical diagnostics section. The parallel light, however, does preclude three-dimensional (3D) positional measurement of the the same objects.

Non-parallel light refractive imaging techniques are desirable for obtaining 3D positions of the objects in the test section. Existing projective refractive imaging techniques include retroreflective shadowgraphy and background oriented schlieren [14], which are generally better suited to larger scale, far field fluid flow visualization than focused-shadograph and focused-schlieren techniques. Background oriented-schlieren (BOS) is a form of schlieren that is ideal for refractive imaging at significantly larger scales compared to the previously mentioned shadowgraph and schlieren techniques. BOS utilizes image processing, including background subtraction and image correlation techniques, to visualize refractive disturbances via their distortion to a background pattern. Three-dimensional reconstruction of shockwaves has been successfully performed via background-oriented schlieren imaging [15]. Although BOS allows the reconstruction of three-dimensional position, the limited pixel resolution of high-speed cameras and image processing needs of BOS makes it insufficient for the fragment measurements desired here.

Retro-reflective shadowgraphy differs from focused-shadowgraphy in that it does not utilize a collimated light beam. Instead it has a light source aligned with the optical axis of camera placed a distance from a retroreflective screen, taking advantage of the diverging light rays of the light source to project shadows of the refractive objects between the light source and the screen. The same setup can be achieved with a single parabolic lens, where the light source and camera are each positioned at twice the lens focal length. This single lens setup is shown in Figure 1.2 and allows better illumination efficiency that typical retroreflective shadowgraphy. The reconstruction of turbulent gases using stereo focused-schlieren (dual-lens) and retro-reflective shadowgraph imaging has been successfully implemented [16], which motivates the work of applying a stereo-shadowgraph (single-lens) imaging technique for determining the 3D positions of fragments with a narrower field of view.

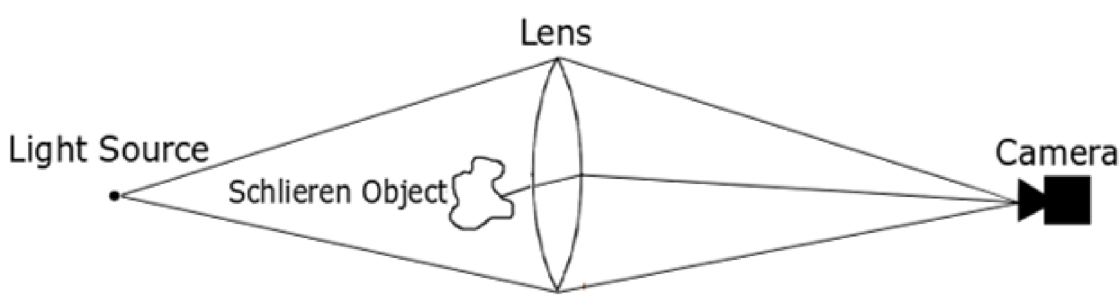


Figure 1.2: A diagram of the single-lens shadowgraph setup with diverging and converging light sections.

1.3 Digital Image Processing

Images are captured on the high-speed CMOS (Complementary Metal Oxide Semiconductor) camera sensor in the shadowgraph imaging system. The number of photons that impact the sensor yields the pixel intensity. Color images are created using a Bayer filter pattern over an image sensor [17]. Images taken by color cameras are constructed from three color planes, or multiple pixel intensity matrices. A Bayer filter is an arrangement of individual pixel color filters over a grid of photosensors that construct the CMOS sensor. Each color filter registers a pixel intensity count that only allows a small range of light bandwidths to pass through, in either red, green, and blue light frequencies. Color images must interpolate each color plane and combine them to construct a desired color image due to the arrangement of color filter mosaic. The exposure of an image is set by the exposure time, or the time that a camera sensor is active. In the case of high speed cameras, this is usually an electronic equivalent to the mechanical shutter of traditional cameras. Proper image exposure is applied in high-speed imaging techniques to mitigate motion blur of objects.

Digital images are matrices filled with intensity values associated with each pixel in the camera sensor. The intensity values are quantized (or digitized) by the camera in the sampling process. Digital image processing techniques are utilized to obtain quantititative and qualitative features within the image and act on matrices of pixel intensities. Threshold-based image segmentation techniques may use global thresholds or varying thresholds based on windows of nearby pixels or statistical methods to apply thresholds when binarizing a grayscale image [18]. Binarized images can label clusters of pixel regions with unique morphological parameters, including pixel area, centroids, bounding boxes, etc [18]. Applying spatial calibration techniques of measuring the pixel width of an object of a known size within the optical diagnostics section allows for size analysis of other objects travelling within the same test section.

1.4 Computer Vision and Three-Dimensional Reconstruction

To obtain depth information of objects, an optical system requires diverging light rays and multiple cameras. Reconstruction of shapes requires several cameras, or camera views, to achieve tomographic reconstruction or via structure from motion [19]. Stereo calibration of two cameras would allow for epipolar geometry as well as extrinsic and intrinsic properties that define the camera setup in relation to the objects being imaged. Transformations between the camera coordinate system C_c and the world coordinate system W_c are defined via the projective geometry described by the extrinsic parameters of the translation T of the optical center from the origin of the world coordinates and the rotation R of the image plane [20]. They define the location and orientation of the camera with respect to the world frame. This gives a position of the focal plane in the world coordinate system. For a point P in 3D space, the camera coordinate system is represented by

$$P_c = R(P_w - T), \tag{1.3}$$

where the external world coordinate system defines the point P_w, and is defined by a rotation matrix

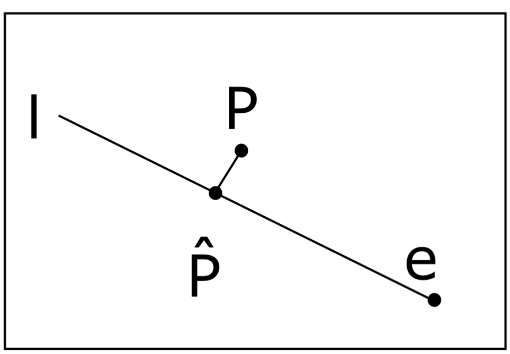
$$R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \tag{1.4}$$

and the translation matrix

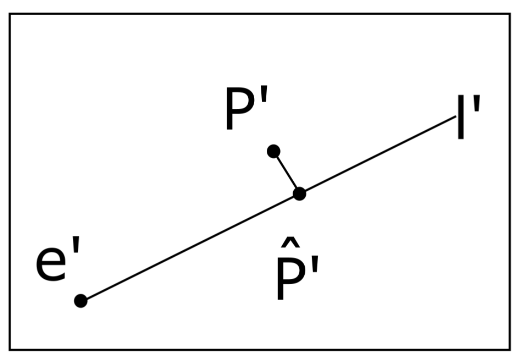
$$T = O_w - O_c = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} \tag{1.5}$$

which are determined from the stereo calibration process [20].

The intrinsic parameters are necessary for performing the transformation between the camera coordinate system C_c and pixel coordinates in the image frame and include the the focal length f, the principle point, pixel sizes [20]. The epipolar geometry defined by the stereo calibrated cameras is described by the epipoles e and e' as well as the epipolar lines l and l', which all lie in the same plane, the epipolar plane [19]. These can be visualized in the image planes associated with each camera, as shown in Figure 1.3. The epipoles are points of projection of camera center into the plane of the opposite camera. The epipolar lines are lines in an image plane corresponding to points in the other plane aligned with the optical center [19].



(a)



(b)

Figure 1.3: Reprojection error minimization is based on the distances of measured and estimated reprojected points in (a) camera view 1 and (b) camera view 2.

The epipolar geometry shows the relationship of points in one image and their respective epipolar lines from the points in the other camera using the equation

$$l' = F\boldsymbol{P} \tag{1.6}$$

where F is the fundamental matrix determined from the stereo calibration process, l' is the epipolar line in one camera, and P is the a point representing an image point in the other camera view [19] with a zero value in the third coordinate place. The fundamental matrix F is a 3×3 matrix which maps corresponding points between stereo images and is determined via the eight-point algorithm. The eight-point algorithm is a computer vision algorithm used to perform stereo calibration based on a set of correspondences, or uniquely identifiable points between two images [21]. The multiplication of the point and the fundamental matrix result in a vector often extrapolated to a line.

The triangulation of an object's three-dimensonal position is performed by minimizing the projected error while satisfying

$$\hat{\boldsymbol{P}}' F \hat{\boldsymbol{P}} = 0 \tag{1.7}$$

as described by Hartley and Zisserman [19] by minimzing the respective distances of the measured positions and estimated positions as shown in Figure 1.3. The reprojection error is the calculated via the geometric error error cost function

$$\mathrm{C}(X) = d(\boldsymbol{P}, \hat{\boldsymbol{P}})^2 + d(\boldsymbol{P}', \hat{\boldsymbol{P}}')^2 \tag{1.8}$$

in which $d(*, *)$ is the Euclidean distance operation applied to the measured image position coordinates $\boldsymbol{P}$ and $\boldsymbol{P}'$ with the reprojected image position coordinates $\hat{\boldsymbol{P}}$ and $\hat{\boldsymbol{P}'}$, as described by Hartley and Zisserman [19] and shown in 1.3. It is minimized by providing numerous paired images of unique orientations of a calibration target such that a sufficient number of correspondences can be made.

1.5 Velocimetry Techniques

Comparisons of the existing velocimetry techniques influenced the approach to tracking fragments in this work. Particle Tracking Velocimetry (PTV) techniques differ from Particle Image Velocimetry (PIV) techniques in that the former is focused on measuring fragment velocities via tracking individual, discrete particles in motion [22]. The latter is focused on estimating the displacement of clusters of particles from correlated displaced windows in pairs of images [22]. A typical methodology for measuring residual velocities of fragments for reactive materials penetrating through thin impact plates is via high-speed video [7, 8]. Efforts to apply in-situ optical diagnostics to perform fragmentation studies for explosive casings [23] and other high velocity impacts in non-RM related studies are active areas of research. Guildenbecher et al. [23] performed 3D optical diagnostics on warhead casings of a known thickness to aid sizing and utilized stereo digital image correlation (DIC) techniques for performing tracking of fragments. Shadowgraph and Kalman-Filter based PTV efforts have been performed [3, 4, 5] for the explosive launch of RMs.

PTV techniques favor relatively larger particle sizes than those observed in PIV; hence the applicability of each favors discrete particles or bulk particle movements of tracer particles, respectively. A typical PIV setup requires a laser sheet and a camera capable of capturing pairs of images with a desired time interval between pulses to perform post-processing techniques and track bulk displacements of particles illuminated by the laser sheet intersecting the flow field. Similar PTV techniques have been performed with a laser sheet as well [24, 25]. Other forms of PIV, including volumetric PIV, requires at least three cameras to measure 3D velocity fields of particles.

1.6 Kalman Filtering

Kalman filtering has been previously applied to track individual objects [26] and for multiple object tracking methods [27, 28]. Linear Kalman filters are used in this work for their simplicity in execution but also because the alternative particle filters, or extended Kalman filters are more appropriate for estimating or predicting non-linear behavior [26]. The Kalman filter generates optimal estimates for state variables of a system by iteratively comparing estimates to measurements with the assumption of Gaussian noise for each object state [26, 29]. Trajectories and velocities of individual fragments can be estimated by iteratively comparing estimates of positions of the fragment from an equation of motion to the observed centroids of fragment projections from the digital image process. Multiple object tracking methods and velocimetry methods have applied Kalman Filters to perform the tracking of individual fragments [26, 27, 30, 31]. Assignment algorithms, explored by Kuhn and Munkres[32, 33], have also been applied to the process of multiple object tracking by assigning the each observed object in sequential time increments to their nearest existing trajectories following the object [30].

In the case of tracking in image space, multivariate Gaussian assumptions are used to describe that an object exists at a location represented as a mean and an associated variance around its 2D position in the form of a multivariate Gaussian. To estimate the location of fragments in the 2D case (image plane) and 3D case (world coordinate system W), the following system of equations describes each object's position, velocity, and acceleration:

$$\begin{aligned} x_k &= x_{k-1} + \dot{x}_{k-1}\Delta t + \tfrac{1}{2}\ddot{x}_{k-1}\Delta t^2 \\ y_k &= y_{k-1} + \dot{y}_{k-1}\Delta t + \tfrac{1}{2}\ddot{y}_{k-1}\Delta t^2 \\ z_k &= z_{k-1} + \dot{z}_{k-1}\Delta t + \tfrac{1}{2}\ddot{z}_{k-1}\Delta t^2 \end{aligned} \tag{1.9}$$

$$\begin{aligned} \dot{x}_k &= \dot{x}_{k-1} + \ddot{x}_{k-1}\Delta t \\ \dot{y}_k &= \dot{y}_{k-1} + \ddot{y}_{k-1}\Delta t \\ \dot{z}_k &= \dot{z}_{k-1} + \ddot{z}_{k-1}\Delta t \end{aligned} \tag{1.10}$$

$$\begin{aligned}\ddot{x}_k &= \ddot{x}_{k-1}\\ \ddot{y}_k &= \ddot{y}_{k-1}\\ \ddot{z}_k &= \ddot{z}_{k-1}\end{aligned} \tag{1.11}$$

The system of equations in the 2D velocity case is then defined in matrix form:

$$\boldsymbol{x_k} = \begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \end{bmatrix} \tag{1.12}$$

The same 2D velocity case can be represented by:

$$\boldsymbol{x_k} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \dot{x}_{k-1} \\ \dot{y}_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}(\Delta t)^2 & 0 \\ 0 & \frac{1}{2}(\Delta t)^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} \ddot{x}_{k-1} \\ \ddot{y}_{k-1} \end{bmatrix} \tag{1.13}$$

in terms of the transition **A** and control **B** matrices:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.14}$$

$$\boldsymbol{B} = \begin{bmatrix} \frac{1}{2}(\Delta t)^2 & 0 \\ 0 & \frac{1}{2}(\Delta t)^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \tag{1.15}$$

The control input

$$\boldsymbol{a_{k-1}} = \begin{bmatrix} \ddot{x}_{k-1} \\ \ddot{y}_{k-1} \end{bmatrix} \tag{1.16}$$

which is used as an acceleration controlling parameter in Kalman filter [26].

The governing multivariate mean state prediction equation

$$\boldsymbol{x_k} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{x_{k-1}} + \begin{bmatrix} \frac{1}{2}(\Delta t)^2 & 0 \\ 0 & \frac{1}{2}(\Delta t)^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \boldsymbol{a_{k-1}} \tag{1.17}$$

can then be simplified to

$$\boldsymbol{x_k} = \boldsymbol{A}\boldsymbol{x_{k-1}} + \boldsymbol{B}\boldsymbol{a_{k-1}} \tag{1.18}$$

in terms of the introduced matrices. For the process of predicting the covariance update equation, it follows the form of

$$\boldsymbol{P_k} = \boldsymbol{A P_{k-1} A^T} + \boldsymbol{Q} \tag{1.19}$$

which is a function of an assumed value for $\mathbf{P_{k-1}}$ as well as the process noise matrix Q is defined as

$$\boldsymbol{Q} = \begin{bmatrix} \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} & 0 \\ 0 & \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & 0 & \Delta t^2 & 0 \\ 0 & \frac{\Delta t^3}{2} & 0 & \Delta t^2 \end{bmatrix} \sigma_a^2 \tag{1.20}$$

using a discrete noise model and dependent on time intervals between measurements and the random noise variance due to acceleration [26]. It is also assumed that the spatial direction measurements of x and y are uncorrelated [26].

The state mean update equation

$$\boldsymbol{x_k} = \boldsymbol{x_{k-1}} + K\boldsymbol{z_{res}} \tag{1.21}$$

corrects the predicted state using the Kalman gain factor K and the measurement residual $\mathbf{z_{res}}$. The measurement residual

$$\boldsymbol{z_{res}} = \boldsymbol{x_{meas}} - \boldsymbol{H x_k} \tag{1.22}$$

is used to determine how to calculate the state correction based on the difference of the measured position $\mathbf{x_{meas}}$ and the predicted state. The observation matrix **H**

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{1.23}$$

represents the observed quantities, including the measured coordinates in the image. The Kalman Gain factor

$$\boldsymbol{K} = \boldsymbol{P_{k-1} H^T} (\boldsymbol{S})^{-1} \tag{1.24}$$

is the correction factor in terms of the covariance, observation matrix, and the innovation covariance matrix

$$\boldsymbol{S_k} = (\boldsymbol{H P H^T} + \boldsymbol{R_k}) \tag{1.25}$$

. The measurement noise covariance matrix

$$\boldsymbol{R_k} = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \tag{1.26}$$

is defined such that the assumed constant measurement uncertainty values associated in each of the directions are uncorrelated and independent [26]. The state mean update covariance equation

$$\boldsymbol{P_k} = (\boldsymbol{I} - \boldsymbol{KH})\boldsymbol{P_{k-1}} \tag{1.27}$$

uses an identity matrix $\boldsymbol{I}$ with the same dimensions as the previous state mean covariance.

The Kalman gain factor always weighs the correction such that $0 \leq K \leq 1$ [26]. The Kalman gain serves as a correction factor for estimations, such that, for increasing K, the correction weighs in favor of the current measurement value as opposed than the predicted value. This process iterates through each time step, updating the predicted trajectory based on the previous position and corrected by measurement of the current measured position of the object. An alternative way to describe the Kalman gain factor is

$$\boldsymbol{K} = \frac{\text{Estimate Uncertainty}}{\text{Estimate Uncertainty + Measurement Uncertainty}} = \frac{\boldsymbol{P_{k+1,k}}}{\boldsymbol{P_{k+1,k}} + \boldsymbol{R_k}} \tag{1.28}$$

which shows that when the measurement uncertainty is small relative to the estimate uncertainty, the Kalman gain is high, or close to 1, and applies a large correction [26]. The Kalman gain is small when the measurement uncertainty is relatively large compared to the estimate uncertainty which is observed with slow convergence after many time steps [26]. The design of the Kalman filter ensures that with each iteration the estimate uncertainty decreases since $0 \leq K \leq 1$ when applied to the state covariance matrix.

Uncertainty covariances are defined as the uncertainties in the state variables for a each fragment and initially assumed to be on the order of ones of pixels and pixels per frame for the position and velocity estimate uncertainties, respectively. The general form for a 2D state prediction covariance is a function of an assumed estimate uncertainty values associated in each of the directions for position and velocities are uncorrelated and independent [26]:

$$\boldsymbol{P_k} = \begin{bmatrix} P_x & P_{x\dot{x}} & P_{x\ddot{x}} & P_{xy} & P_{x\dot{y}} & P_{x\ddot{y}} \\ P_{\dot{x}x} & P_{\dot{x}} & P_{\dot{x}\ddot{x}} & P_{\dot{x}y} & P_{\dot{x}\dot{y}} & P_{\dot{x}\ddot{y}} \\ P_{\ddot{x}x} & P_{\ddot{x}\dot{x}} & P_{\ddot{x}} & P_{\ddot{x}y} & P_{\ddot{x}\dot{y}} & P_{\ddot{x}\ddot{y}} \\ P_{yx} & P_{y\dot{x}} & P_{y\ddot{x}} & P_y & P_{y\dot{y}} & P_{y\ddot{y}} \\ P_{\dot{y}x} & P_{\dot{y}\dot{x}} & P_{\dot{y}\ddot{x}} & P_{\dot{y}y} & P_{\dot{y}} & P_{\dot{y}\ddot{y}} \\ P_{\ddot{y}x} & P_{\ddot{y}\dot{x}} & P_{\ddot{y}\ddot{x}} & P_{\ddot{y}y} & P_{\ddot{y}\dot{y}} & P_{\ddot{y}} \end{bmatrix} \tag{1.29}$$

1.7 Assignment algorithms

An assignment algorithm is one that attempts to solve the assignment problem or transportation problem, which is traditionally the issue of assigning a worker to at most one job for the overall minimum cost [32, 33]. Assignment algorithms operate on cost matrices that divide "workers" and "tasks" into rows and columns, and each element of the matrix is a cost associated with assigning a worker to a particular task [33]. Here in the cost matrix for fragment tracking, rows represent existing trajectories and columns are detected fragments, which are analogous to workers and tasks, respectively. Euclidean distances between measurements and predictions serve as the cost associated with each combination of existing trajectories and detected fragment positions.

Assignment algorithms also ensure that each task has a unique worker and no worker performs more than one task. Row reduction is performed by updating a new matrix such that the smallest value in each row is subtracted to ensure that each row has at least one zero [32, 33]. Column reduction follows the same procedure for each column [32, 33]. A process of "covering" zeroes in the matrix with lines (rows or columns) with a form of temporary marking indicated with stars and primes on potential assignments is performed [32, 33]. This ensures that all zeros in the matrix are now covered with a minimal number of rows and columns[32, 33]. Subtraction of the smallest element in the matrix from every unstarred or unprimed matrix cost element is applied followed by the addition of it to every element at the intersection of two covered lines (rows or columns) [33]. The above process is repeated until the minimum number lines covering each zero is equal to the number of workers or tasks assigned, where the position of the assigned zeros correspond to the positions of the original cost matrix to determined the minimum cost assignment matrix [33].

1.8 Optical Size Estimation

Size estimation of fragments using an optical diagnostics requires a consideration of geometric optics and image processing applied to fragment projections for successfully tracked fragments. The geometric optics associated with the use of schlieren lenses obeys the thin lens equation

$$\frac{1}{f} = \frac{1}{L_i} + \frac{1}{L_o} \tag{1.30}$$

such that the focal point lies a focal length distance f from the lens center [34] and L_o and L_i are the distances of the objects and the image respectively. The non parallel light in the light cones from the lenses result of magnification

$$M = \frac{h_i}{h_o} = \frac{L_i}{L_o} \tag{1.31}$$

of the objects depending on their distances from the center of the lens [34] where h_o and h_i are the heights of the object and image respectively. When considering the size of the objects optically, a consideration of the image plane, focal plane, and the calibration plane are essential. The image plane is the plane in which the image is formed and is typically the sensor of a camera. The focal plane is the plane where objects would appear in focus since the converging rays from a convex lens lie on the same plane at a distance f. It is perpendicular to the optical axis passing through the focal point. The calibration plane is the plane in which the calibration target lies in the test section.

Grady and Kipp applied 2D projection methodology for in-situ sizing methodology directly to fragment sizes while using an equivalent spherical diameter assumption [35, 36]

$$d_e = \sqrt{\frac{4A_{px}}{\pi}} \tag{1.32}$$

where A_{px} is the pixel area of an object in the image.

This assumption is also applied generally in dynamic image analysis (DIA) [37] used to quantitatively estimate the size of sands and small grains in geological studies. DIA studies typically average the observed equivalent diameters, but also employ other sizing and shape factors, including minimum Feret diameter-based sizing [37].

A general size estimation approach that considers the tumbling of fragments is desirable, especially since other works performing in-situ optical sizing[23] rely on determining the normal to the flat face of fragments to estimate sizes or single area projection measurements [35, 36]. The size estimation approach inspired by the DIA in computerized particle size analyzers, which operate on observing numerous projections of fragments and particles via a high-frame rate camera, is taken in this body of work.

1.9 Research Objectives

The objective of this thesis is to develop methodologies for performing in-situ optical diagnostics for tracking RM fragments resulting from ballistic impacts. This thesis will explore shadowgraph imaging techniques, digital image processing and dynamic image analysis, stereo camera epipolar geometry, Kalman-Filtering applications, velocimetry techniques, and in-situ fragment size analysis. The goal of this work is to develop, apply, and validate stereo single-lens shadowgraph systems to visualize, track, and determine trajectories, velocities, and sizes of tumbling RM fragments in-situ.

CHAPTER 2

EXPERIMENTAL METHODS

2.1 Experimental Setup

A test series of impact experiments was conducted at the Naval Surface Warfare Center Indian Head Division. A two-camera stereo shadowgraph system was implemented and 18 individual tests were performed. The ballistic impact experimental setup consisted of a .50 inch (12.7 mm) caliber gun firing RM samples at a steel plate. The RM projectiles consisted of Cylindrical RM samples of Al, Al/W composites, and Al/Bi2O3 composites. The Al samples analyzed in this work are listed in Table 2.1, which summarises the dimensions and properties of the RM specimens pressed by the Naval Surface Warfare Center Indian Head Division. Eight of the tests were performed for 3D reconstruction of trajectories and 3D velocity measurement.

The cylindrical RM projectiles were held in 3D printed sabots made from ASA filament for gun-launch and were aerodynamically removed via a sabot stripper before the projectile reaches the steel plate. An external TTL signal is sent from a break screen located at the muzzle and via Standford Research Systems DG535 digital delay generator when the projectile exited the barrel. The DG535 then provided TTL signals to trigger the cameras and laser illumination source.

Table 2.1: A comparison of RM specimens used as projectiles for the impact fragmentation tests.

Shot #	Pellet Material	Mass (g)	Height (cm)	Density (g/cc)	% TMD
9	Aluminum	0.488	0.594	2.593	96
10	Aluminum	0.502	0.612	2.590	96
11	Aluminum	0.473	0.587	2.545	94
13	Aluminum	0.485	0.589	2.598	96
14	Aluminum	0.476	0.577	2.607	97

The aluminum (Al) specimens were composed of Valiment H-60 aluminum with a 60 micron particle size and pressed to 206.8 MPa to 275.8 MPa (30,000 to 40,000 psi). The aluminum-tungsten (Al/W) specimens were composed of 25% Al and 75% W by weight, using H-2 aluminum with a 2 micron particle size and tungsten with a 44 micron particle size, pressed between 206.8 MPa to 275.8 MPa

. The Al/Bi_2O_3 specimens were composed of 25% Al and 75% Bi_2O_3 by weight, using H-2 aluminum with a 2 micron particle size and also pressed between 206.8 MPa to 275.8 MPa.

The optical diagnostics setup is shown in Figure 2.1. The shadowgraphy technique here is used, though not directly for the purpose of refractive imaging, for the purpose of extracting the desired fragment area projections since the technique is useful for visualizing sharp disturbances in the optical test section. The back-illumination of the fragments and objects that travel through the optical test section will provide projections of the fragment areas which is necessary for determining their sizes. The projected shadows of the fragments can be more easily extracted from the digital image process than that of direct high-speed video of the ballistic impact event. The converging light section is desirable for the extraction of the 3D depth information of the objects as well, which is necessary for reconstructing the 3D trajectories of the fragments and incident projectile.

The setup consists of a two intersecting single lens shadowgraph systems, as shown in Figure 2.1. Each individual shadowgraph system utilizes a point-like light source that is refocused by a large lens to a camera placed at the focal point. Each individual shadowgraph system is angled such that each is observing a different orientation of the test section. The test section is placed in the converging-light side of the lenses. The stereo shadowgraph setup utilizes the angle between the individual shadowgraph systems and the non-parallel light of single-lenses to meet conditions to perform 3D reconstruction of fragment trajectories. For the estimation of object sizes, the plane the objects are in must be considered as discussed in Section 1.2. A dual-lens parallel light stereo setup would not be capable of 3D reconstruction of fragment trajectories because no depth information can be extracted from parallel light. Effectively, only 2D velocity information could be constructed from 2D trajectories in each 2D plane of each camera; however, fragment sizes could be estimated without a need to know what plane the objects are in.

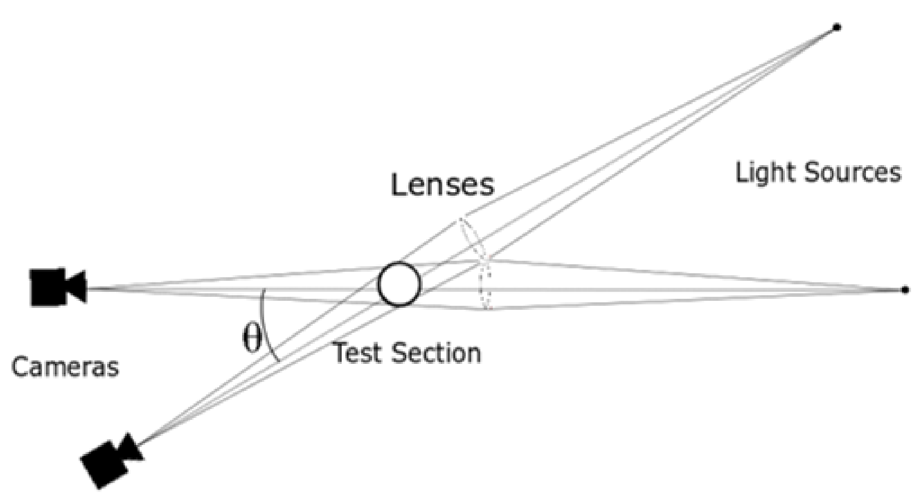


Figure 2.1: A diagram of the stereo single-lens shadowgraph setup.

Two intersecting shadowgraph imaging systems were set up at the impact point of the projectile on the steel plate, as shown Figure 2.2. This camera position allows imaging of the projectile before the impact of the plate and the subsequent fragmentation behavior of the RM samples. Phantom v2012 and v1212 color cameras were used to image the experiments. Each was placed on the optical rail screwed onto the stereo shadowgraph support scaffolding constructed from 80/20 aluminum T-slot structural framing. The cameras are oriented and elevated appropriately on the support scaffolding to image the same test section i.e. the impact point on the steel plate.

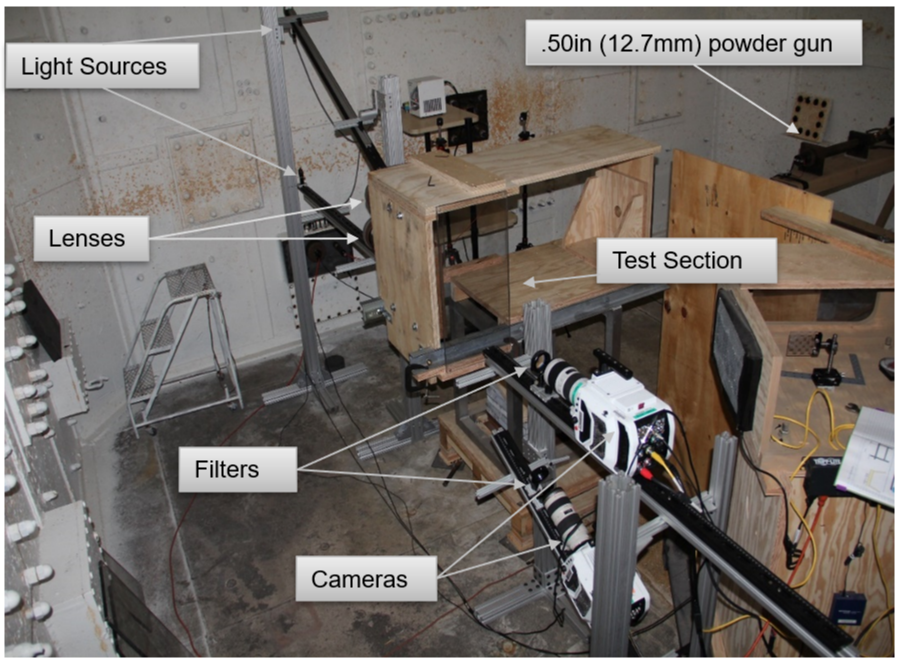


Figure 2.2: An annotated stereo shadowgraph Setup for performing optical diagnostics of ballistic impact plate experiments.

The stereo shadowgraph support scaffolding was constructed from 80/20 Aluminum T-slot structural framing in two separate isolated frames. The first frame used two "poles" of 7.6 cm x 7.6 cm (3 in x 3 in) of hollow Quad rail with lengths of 1.2 m and 1.8 m (4 and 6 feet) whereas the second frame used 1.8 m and 2.4 m (6 and 8 feet) respectively. Supports for the quad rails holding optical rails consist of, on either side of the frame, a single Four Slot Rail 7.6 cm x 7.6 cm (1 in x 1 in) screwed onto an inline/perpendicular pivot connected to a sleeve bearing carriage or mount flange bearing with hand brakes. This support setup allows the optical rails to be elevated and oriented as appropriate for the stereo shadowgraph setup to be centered on the desired test section. Each 1.5 m (5 ft) long 45 mm Square Hollow 4-Slot rail act as a beam to support two 750 mm long THOR LABS dovetail rails for mounting all the cameras and optics.

For the dual lens setup, the length from the first camera to the parabolic lens was approximately 59.1 cm (23.3 in) for both the horizontal and diagonal systems. The distance between the cameras vertically was 57.8 cm (22.8 in), measured from the ends of the center of the camera lenses. The distance from the centers of the schlieren lenses was approximately 1.23 m (4.04 ft) for the horizontal system and 1.38 m (4.54 ft) for the diagonal system. The distance from the second schlieren lens to the light source was 0.67 m (2.19 ft) for both the horizontal and diagonal systems. The angle was approximately 24-25 $^{\circ}$ oriented between the shadowgraph systems.

For the single lens setup, the distance between the camera lens to the schlieren lens was 1.41 m (4.63 ft) for the horizontal setup and the 1.49 m (4.49 ft) for the diagonal system. The distance from the light source to the second schlieren lens was 1.28 m (4.2 ft) for the both horizonal system and diagonal system. The distance from the lenses on the light-source structural frames to the impact point of the test section was approximately 0.44 m (1.44 ft) for both systems. The angle was approximately 24-24.5 $^{\circ}$ oriented between the shadowgraph systems.

The cameras recorded at 50,000 frames per second (fps), which resulted in two different image sizes because of the cameras used. The v2012 recorded at a frame size of 592x640 and the v1212 recorded at 432x384 pixels. The light source, a SI-LUX 640 nm spoiled coherence laser, provided a 20 ns pulse width. The pulse width is the effective exposure time for each image which mitigated the effects of motion blur of fragments to allow for accurate projected areas of fragments. The use of the shadowgraph technique takes advantage of focusing the laser to maximize the intensity of imaging through the combustion environment, aiding in the ease of back-illumination of the desired fragments and their projected area extraction.

The effective camera resolution is determined by removing the interpolation of the color cameras created in the Bayer filter process. The Phantom camera Bayer filter has a "gbrg" pattern. The Phantom camera CMOS sensor thus only imaged on the red filtered pixels while using a red laser for illumination. The red laser light reduces the effective resolution to a quarter of the original resolution, from 432x384 to 216x192 and 592x640 to 296x320 for each camera view, respectively.

The lenses attached to the cameras were 80-200 mm lenses. The schlieren lenses used for the shadowgraph setups were 127 mm in diameter each, with 700 mm focal lengths. 50mm square absorptive neutral density filters were used to reduce the intensity of the light to aid the visualization of the shadowgraph images during testing. 640 nm bandpass filters (50 mm diameter, OD 4.0) were used to isolate the light reaching the camera to only that in the laser wavelength such that the much of the direct light from the RM combustion is filtered from the imaging process. Lexan was placed around the steel plate to protect the optics from the fragments produced after impact while still being optically clear for the shadowgraph imaging.

When applying the shadowgraph technique to extract the fragment area projections, the refractive properties may not be desirable since the visualization of the product gas clouds may partially obscure the fragment areas. A minimization of the sensitivity of the shadowgraph technique is therefore desirable to mitigate the occlusion of fragment area projections in the product gas clouds. The sensitivity of the shadowgraph technique employed, i.e. the minimum resolvable refraction angle, is a function of the optical geometry of the system. The shadowgraph sensitivity is influenced by the distance L from the focusing parabolic lens from the refractive object S and the distance L_{total} from the focusing parabolic lens to the camera, as shown in Figure 2.3. The refracting object in non-parallel light forms of shadowgraphy will typically have the highest sensitivity halfway between the camera and the parabolic lens used to focus the light. The sensitivity cannot be minimized by decreasing the length of the system since it is restricted to the focal length by the focusing parabolic lens. Alternatively, decreasing the ratio of the distance from the refractive object S to the camera L' and the distance of the parabolic lens from the camera L_{total} would reduce the shadowgraph sensitivity; however, this would also undesirably reduce the field of view D to D' for imaging the ballistic impact event, as shown in Figure 2.3.

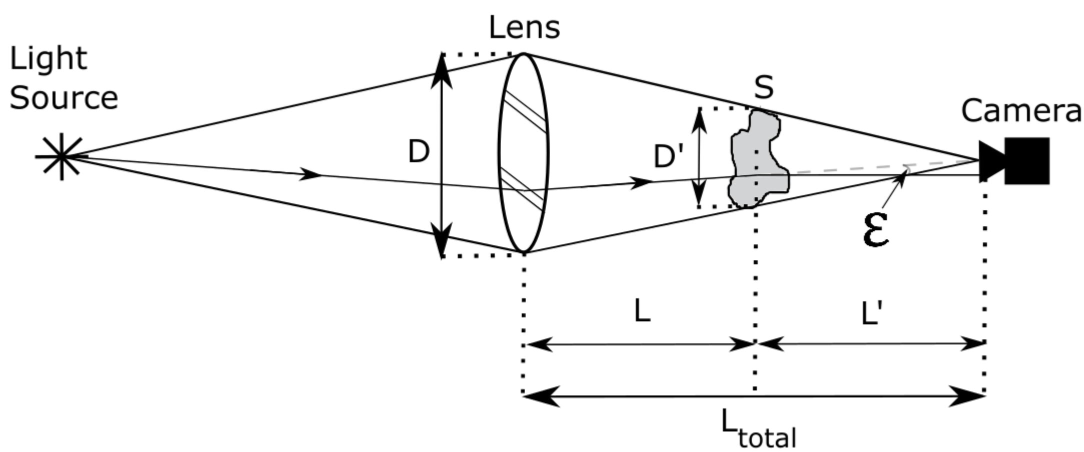
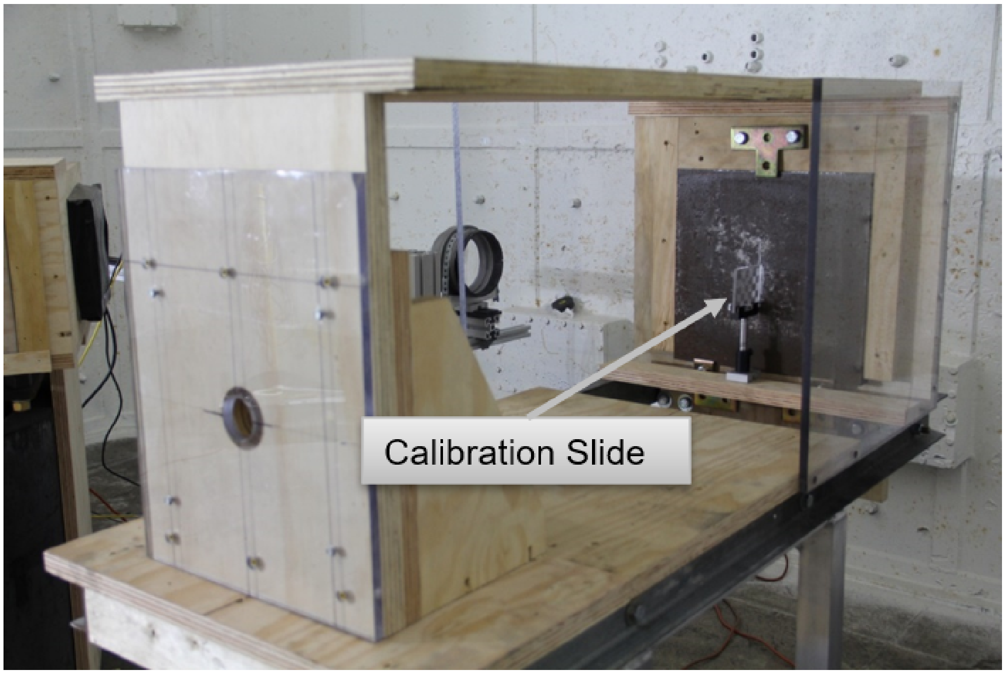


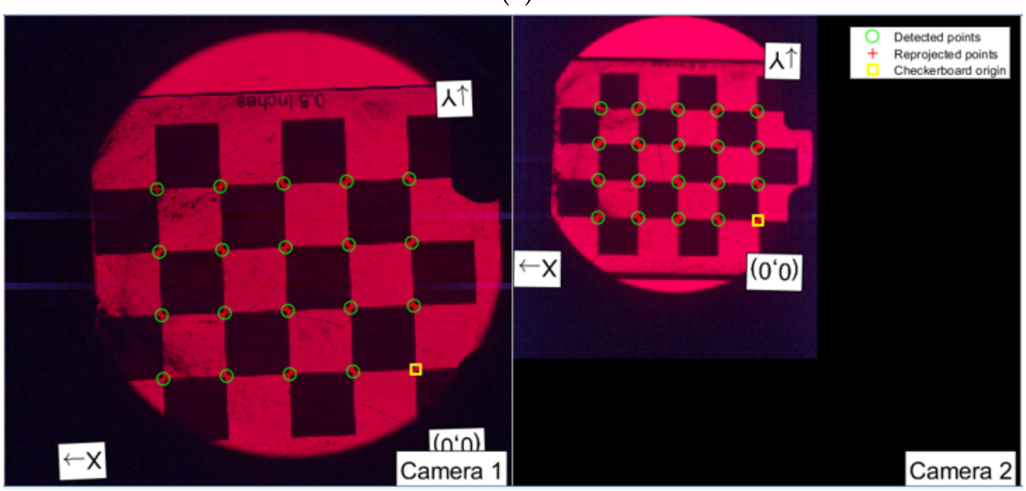
Figure 2.3: An annotated shadowgraph setup with a refractive object in the converging light test section.

2.2 Stereo Camera Calibration

The stereo camera calibration process consists of taking on the order of 20 image pairs of a calibration target simultaneously in each camera field of view, as shown in Figure 2.4. The calibration target used was an asymmetrical checkerboard with a grid resolution of 12.7 mm (0.5 inches).



(a)

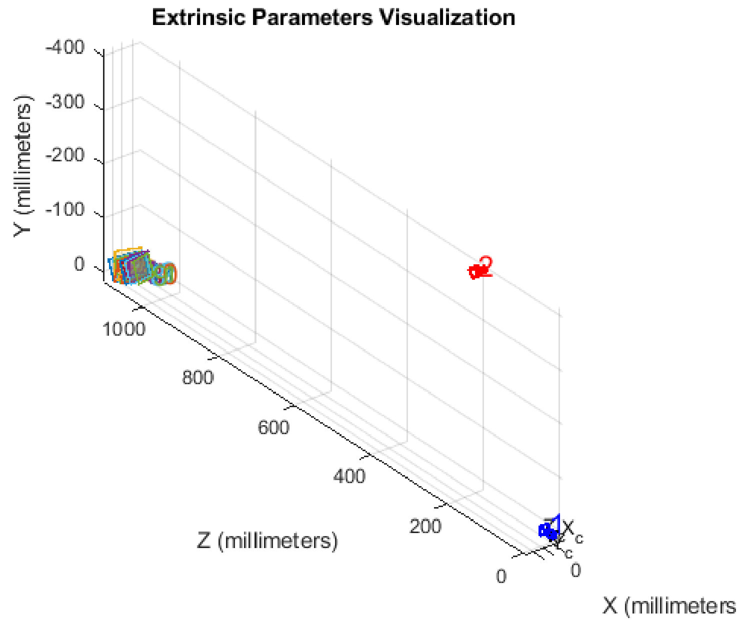


(b)

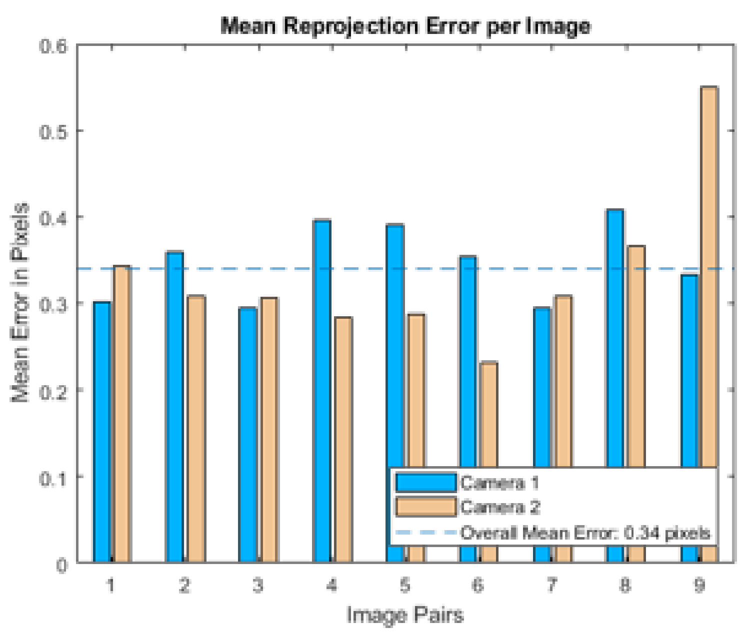
Figure 2.4: (a) A calibration target placed within the test section of the stereo shadowgraph optical setup and (b) Side-by-side views of same checkerboard in each camera's field of view with right image zero-padded with the detected and reprojected points.

The calibration target is rotated and translated slightly with each image pair to capture a variety of unique orientations of the calibration target. An 8-point algorithm via MATLAB's Stereo Camera Calibrator application is applied to the pairs of stereo calibration images. The algorithm seeks to relate the vertices of the checkerboard from one image representing the first camera view with another simultaneous image taken from the other camera view for several paired images. Using MATLAB's Stereo Camera Calibrator application, the internal and external parameters are estimated for the camera system, the latter of which is visualized in Figure 2.5.

MATLAB calculates the position of the target from each pair, and the final output is the relative position of the two cameras and their fields of view. Additionally, the application allows for the minimization of the reprojection error for the estimated fundamental matrix via the removal of poor image pairs such that the errors are subpixel for each stereo calibration performed for this test series. The average reprojection error is less than 0.4 pixels, as shown in Figure 2.5b. The two cameras had different resolutions, each image must be zero-padded to the same size for the MATLAB reconstruction algorithms, shown in Figure 2.4. Zero padding was performed such that the black pixels were appended to the right and bottom of the original image. Zero padding symmetrically such that the original image is centered in the padded image and using these images in the stereo calibration process did not change the average reprojection error or the estimated extrinsic stereo camera parameters, shown in Figure 2.5. For the sake of simplicity, zero padding to the right and the bottom of the image was preferable to avoid consideration of position offsets in further analysis of the digital images.



(a)



(b)

Figure 2.5: The mean reprojection error bar chart for a stereo calibrated shadowgraph setup using 9 successful image pairs, generated by MATLAB's Stereo Camera Calibrator application.

2.3 Digital Image Processing

The goal of digital image processing is to uniquely identify fragments for the purpose of tracking positions and determining fragment sizes. After acquiring and saving the high-speed video of a ballistic-impact test the images to be used in the digital image processing process are exported from the camera software. The Bayer pattern is disabled by ensuring the "Color Interpolation" option is set to "OFF" in the Phantom Camera Control (PCC) software. The PCC software exports the images as a digital negative file with a .DNG file extension, which are then converted into .tiff images in MATLAB.

One of the challenges of fragment detection is the separation of fragments from within fine particle clouds and product gases as a result of the combustion of the RM specimen upon impact. The use of MATLAB's *adapthresh* allows for greater local separation of fragments from the backgrounds that otherwise could not be extract by Otsu's method, since Otsu's method assumes relatively invariant background intensity changes and effectively a pure bimodal pixel intensity histogram for a given image [18]. Figure 2.6 shows the modified background image resulting from *adapthresh* to find appropriate threshold values at different regions in the background of the observed image series .

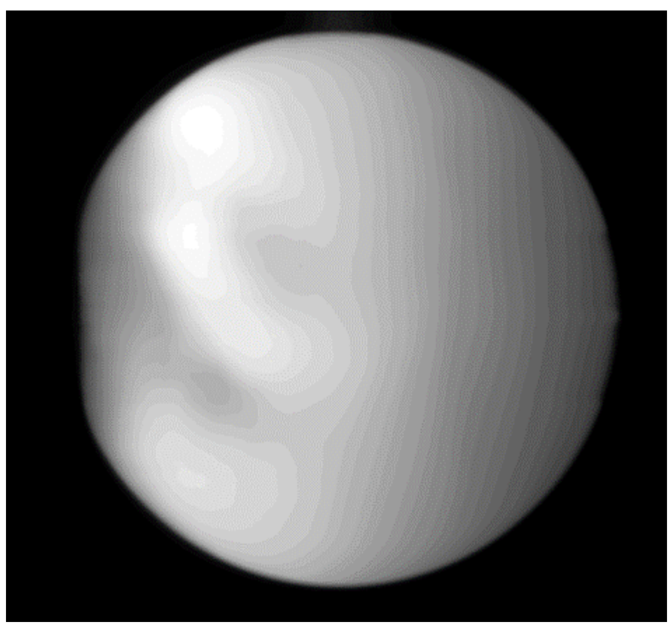
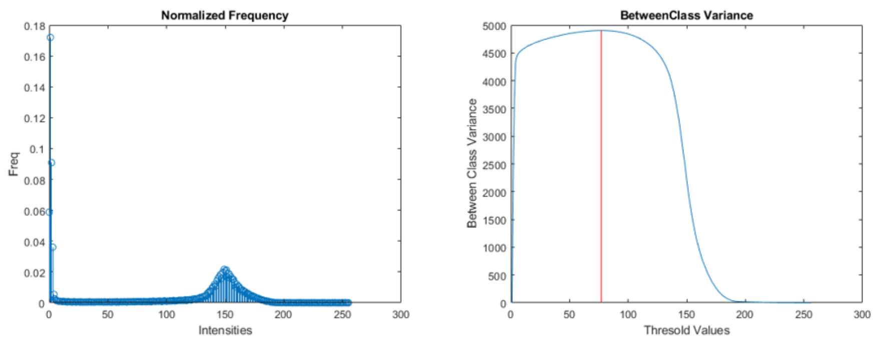


Figure 2.6: An image of the effective background for improved isolation of fragments from a background with significantly varying illumination due to product gas clouds and fine particle clouds.

Using Otsu's method via MATLAB's *graythresh* and *imbinarize* functions to determine a global intensity threshold, the image is binarized to separate fragment regions from the background with noisy light intensities. Otsu's method determines the threshold to maximize the between-class variance, as shown in Figure 2.7a, to optimally separate the fragments in the foreground from the background of the image [18]. Otsu's method effectively determines the threshold that separates the bimodal pixel intensity histogram for an image to optimally binarize the image. Figure 2.7 describes the use of Otsu's method and observed frequency of pixel intensities for an image of the entire field of view.



(a) (b)

Figure 2.7: (a) Normalized Frequency and (b) Between Class Variance

The image segmentation [18, 38] is performed to identify and extract the pixel area, and centroids via MATLAB's *regionprops* function. The application of a global pixel intensity threshold to binarize the image separates fragment regions from the background. Clusters of connected pixels in the morphological image are automatically labelled to identify unique fragments and obtain a pixel area or pixel count. Centroids of these fragments are extracted for each uniquely identified binarized fragment, as shown in Figure 2.8. Further detail on estimating fragment size in terms of pixel area and the associated pixel area error is described in Section 2.4.

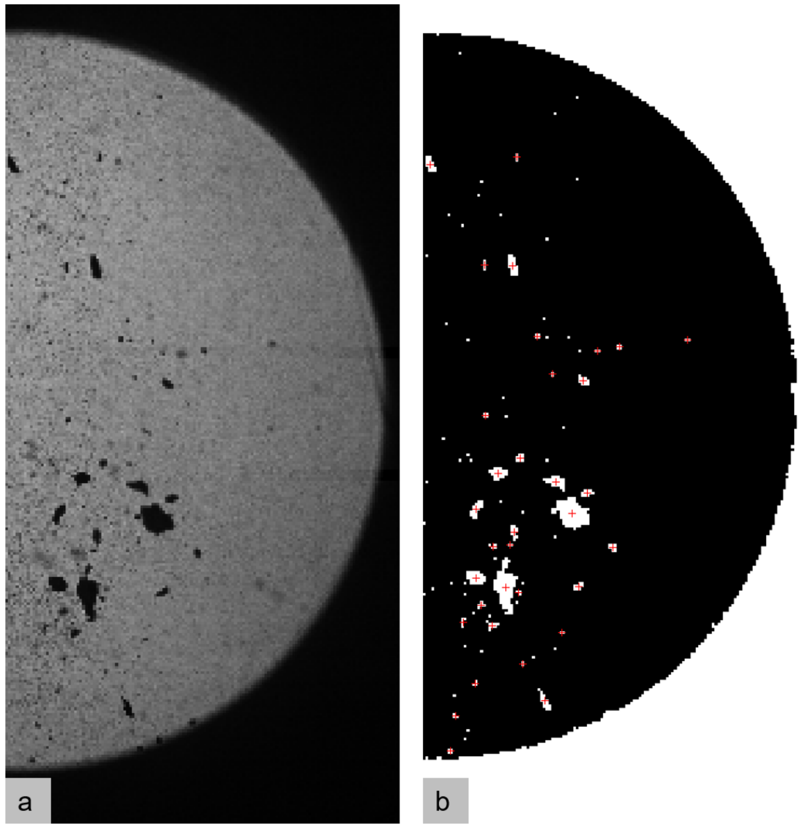


Figure 2.8: Centroids of fragments denoted by small red crosses.

2.4 Fragment Size Estimation

From the digital image process described in Section 2.3, MATLAB's *regionprops* function is used to determine centroids of fragments and extract the bounding boxes of the fragments. Bounding boxes, as shown in Figure 2.9, are constituted of the smallest rectangle that encapsulates each fragment area. In order to obtain a more precise fragment area with associated error bounds, additional digital image processing is performed using Otsu's between-class variance thresholding. While *regionprops* could be used to extract areas of fragments in the binarized image directly, bounding boxes are extracted for the fragments instead to more easily determine the uncertainty of area measurements via Otsu's method.

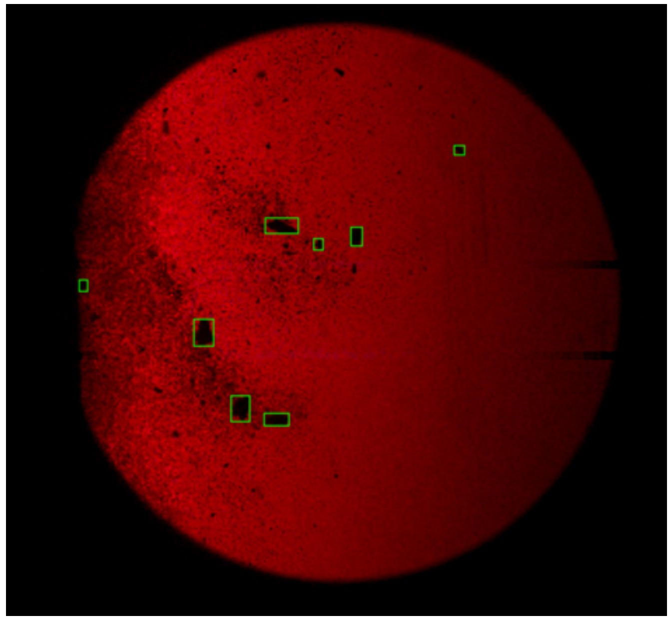


Figure 2.9: Bounding boxes obtained boundaries of fragment areas from the fragment detection process.

The optimal pixel intensity threshold is associated with the peak between-class variance, shown in red in Figure 2.10. However, there is often a short range of values that results in the same between-class variance, shown in red in Figure 2.11, which is the range that correlates the the same pixel intensity threshold associated with the same between-class variance curve shown in 2.10. Similar to the procedure applied by Watson[24], the size of each fragment is found by iteratively thresholding the region of interest, in this case a bounding box, and using the peak in the pixel area gradient to determine the range of possible pixel area values for a single frame. Figure 2.11 shows the area of a fragment for varying pixel intensity threshold values as well as the gradient of the area, demonstrating that the range of optimal threshold values does not necessarily correlate to jumps in pixel area gradient unlike what is observed by Watson to determine optimal fragment contours [24]. The local minimum in gradient observed in Figure 2.11 does not necessarily correlate to consistency in extracting fragment projected areas. The gradient of pixel area for extracted fragment projected areas could vary significantly and local maxima and minima for several fragment area measurements, including for different fragments entirely, did not necessarily lie in the threshold range obtained.

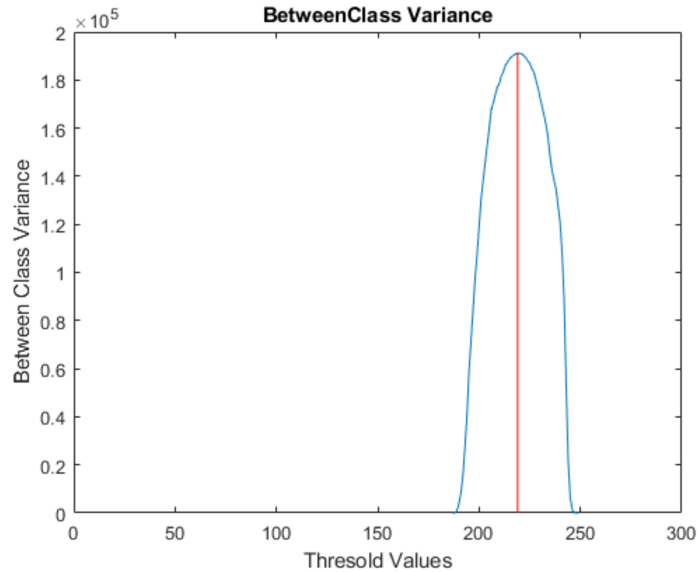


Figure 2.10: The Between-Class Variance of the extracted image associated with the bounding box of a fragment.

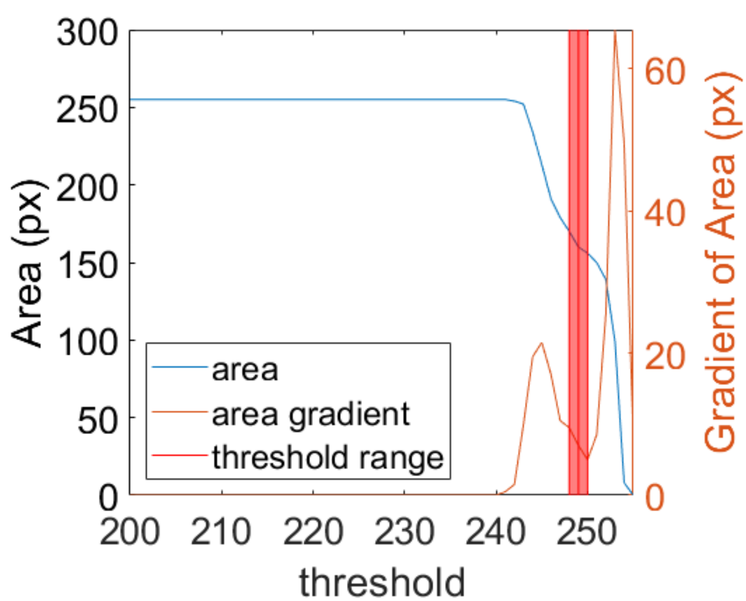
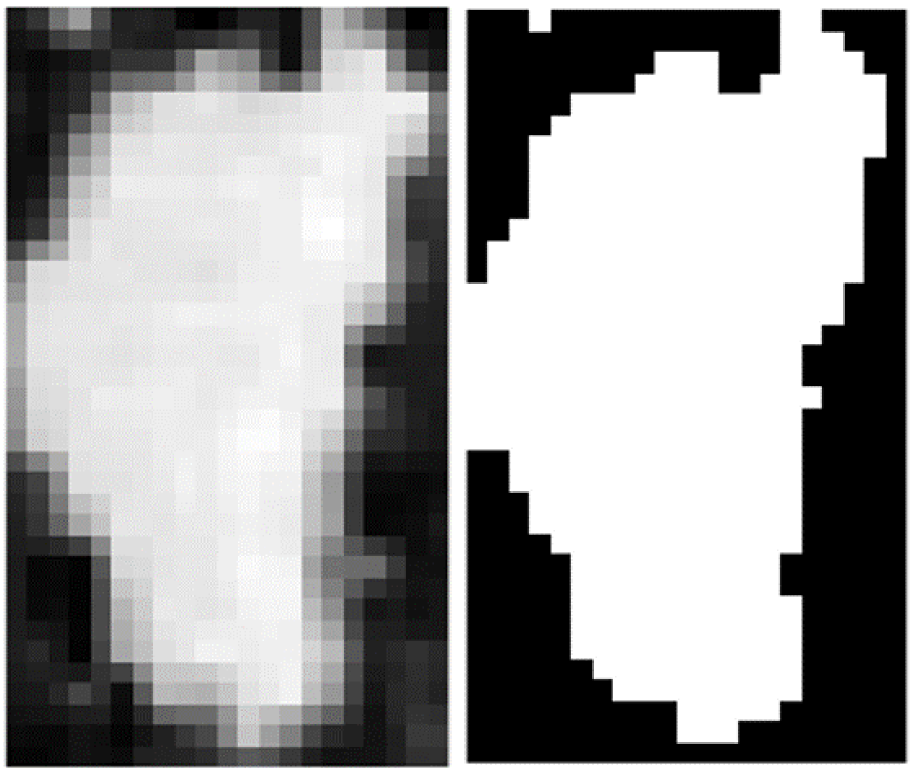


Figure 2.11: The pixel area and gradient of pixel area of the extracted image associated with the bounding box of a fragment.

The result of Otsu's method applied to a grayscale image of the boundary box containing a given fragment, visualized in Figure 2.12a, is the binary image from which MATLAB's *regionprops* isolates the largest fragment and determines the pixel area in Figure 2.12b. The exact boundary can be extracted via MATLAB's *bwboundaries* as shown in Figure 2.13. The threshold range of maximum values of between-class variance for the bounding box image corresponds to the threshold range of pixel areas to determine the pixel area error bounds surrounding the pixel area determined via Otsu's method directly using Matlab's *graythresh*, *imbinarize*, and *regionprops*.



(a) (b)

Figure 2.12: (a) Grayscale image of extracted boundary box of fragment and the (b) resulting binary image of the extracted boundary box of the same fragment.

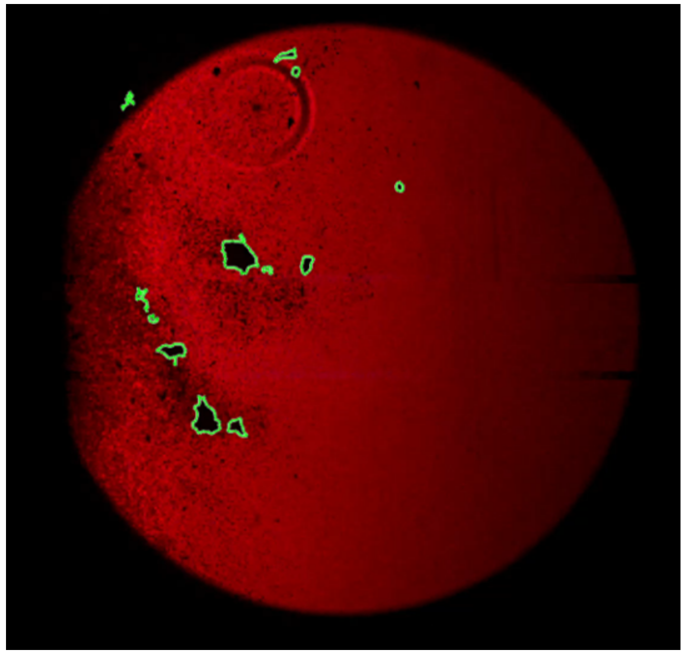


Figure 2.13: Obtained boundaries of fragment areas from fragment detection process.

The 2D projection methodology described by Grady and Kipp [35, 36] in their studies of high-velocity impact-fragmentation of bulk metals using high-speed x-ray imaging can be applied generally to in-situ optical diagnostics for obtaining fragment sizes. An equivalent spherical particle assumption is used here for each fragment tracked. Size estimates are performed by taking the existing tracked fragments and measuring the pixel area, or counting of the number of pixels, of a fragment region and converting to an equivalent diameter in pixel space, then applying a pixel-to-width conversion using a spatial calibration measured in the region image before tests. The equivalent spherical particle assumptions has been applied in high-rate dynamic loading seen in hypervelocity impacts [24]to obtain estimates of the particle sizes.

Consideration of the magnification of objects due to the lens in the shadowgraph system is accounted for as a scaling that considers the distance of the object from the camera and the distance of the calibration plane from the camera. This is determined by using the triangulation process described in Section 2.6 and applying Matlab's *norm* function to calculate the normalized distance of the fragment triangulated in 3D space to Camera 1, defined as the camera from which the camera-to-world coordinate transformation is performed via the stereo calibration process.

2.5 2D Kalman Filtering

The goal of the application of Kalman filtering over a sequence of images is to automate the process of tracking multiple fragments for a given test. A linear Kalman filter model is selected as an appropriate algorithm for tracking fragments and obtaining velocity measurements because the in-flight fragments are assumed to be non-maneuvering objects [26] with a four-dimensional state vector:

$$\boldsymbol{x_k} = \begin{bmatrix} x_k & y_k & \dot{x_k} & \dot{y_k} \end{bmatrix}^T \tag{2.1}$$

The state vector for fragments in a 2D image is dependent on the positions (x_k and y_k) and velocities ($\dot{x_k}$ and $\dot{y_k}$) in each direction of the image space. The following diagram 2.14 is a summary of the Kalman Filtering process described in Section 1.6. This process uses a known prior state to make a prediction of a future state that is then corrected after a comparison of the prediction and a measurement. The corrected prediction is the output and now serves as the "prior" state in a feedback loop iterating on time step k as shown in 2.14. The intial state is assumed to have the initial measured centroid position for the first occurrance of a detected fragment.

The Kalman filter is implemented following the procedure described by [26, 27, 29], which begins with calculation of the predicted mean $\mathbf{x_k}$ and covariance matrix $\mathbf{P_k}$ of the state variables. The transition matrix $\mathbf{A}$ represents the system dynamics for the x and y directions in the 2D image plane with respect to time interval Δt. The Kalman filter then calculates the innovation covariance matrix $\mathbf{S_k}$ and the Kalman gain factor K where the observation matrix $\mathbf{H}$ represents the observed quantities, the x_k and y_k positions in the image space at time step k. The Kalman filter will then calculate the a posteriori mean: $\mathbf{x_k}$ and covariance matrix $\mathbf{P_k}$ by taking account the Kalman gain as a correction factor. The initialized a priori covariance matrix $\mathbf{P_k}$ which is initially assumed to have a value of 1 pixel. The process noise matrix $\mathbf{Q}$ has an assumed initial random acceleration uncertainty that is assumed to be on the order of 1 pixel per unit time (frame) squared in terms of the image space.

For automating the process of tracking multiple fragments in each frame, an assignment algorithm [32, 33] is utilized to assign fragment detections from the digital image process to existing fragment trajectories generated by the Kalman filter, or create new ones. An assignment algorithm operates on cost matrices generated for each frame. The cost matrix for a given frame represents the difference or Euclidean distance between the estimated position and the measured positions of newly detected fragments. A cost matrix is generated by first defining a matrix of the estimated position coordinates for each fragment and the measured position coordinates of fragments in the image at the a priori time step. The Euclidean distance between all possible pairwise coordinates is placed into a pairwise distance matrix and made into a square matrix using MATLAB's *squareform* function.

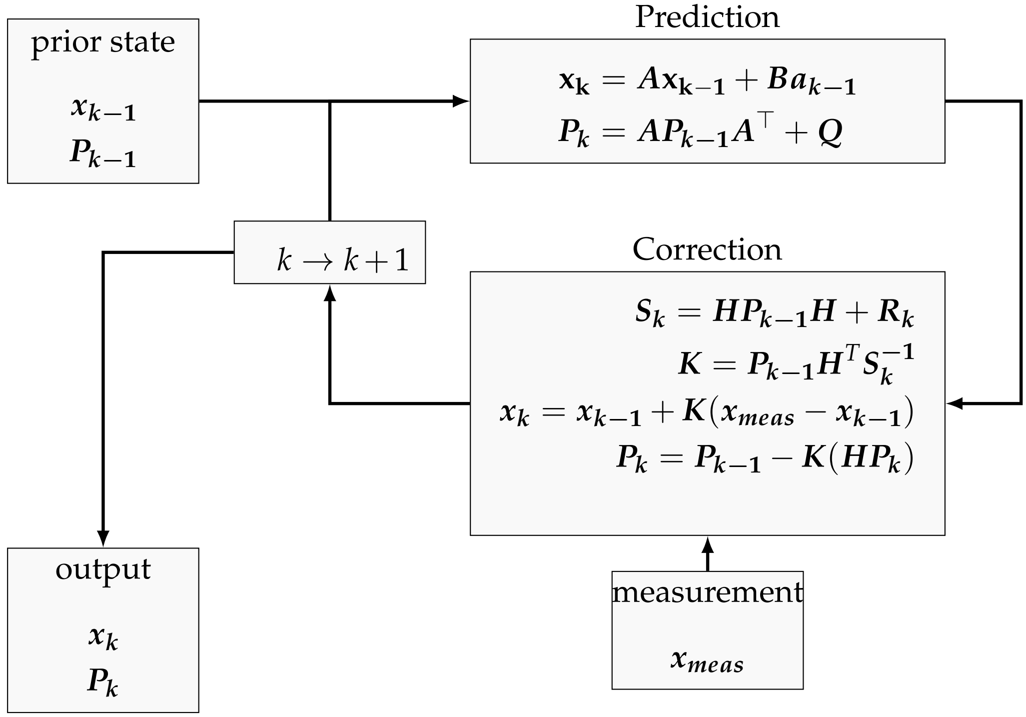
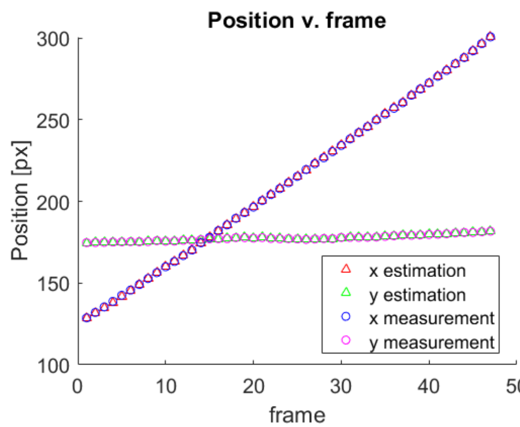
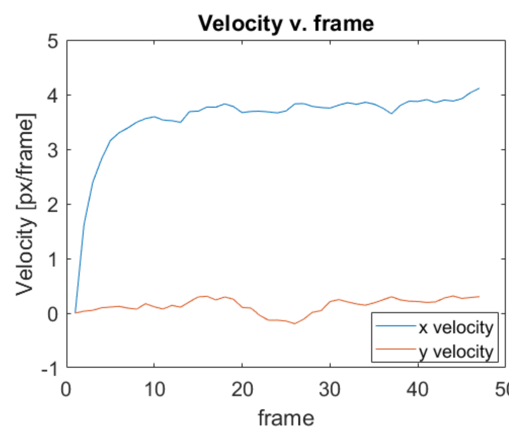


Figure 2.14: A diagram summarizing the Kalman Filter feedback loop that takes a prior state, makes a prediction for a future state, uses a measured state to compare with the prediction to apply a correction, outputs the corrected state which is used in future predictions.

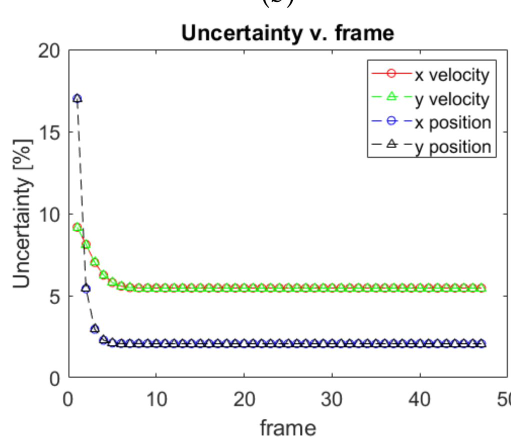
An existing Matlab implementation of the generalized Munkres-based optimal assignment algorithm [39] is applied before applying the Kalman gain correction step in the Kalman filtering process. An individual fragment is tracked manually to obtain order of magnitude velocity information in the horizontal direction in the 2D image plane to inform the general initial conditions for automatically tracking multiple fragments. Position estimates for new fragments are initialized to be the unassigned fragment centroids. An assignment algorithm [32, 33] is also utilized to allow for the assignment of fragment detections from the digital image process to existing fragment trajectories generated by the Kalman filter, or create new ones.



(a)



(b)



(c)

Figure 2.15: An example of position and velocities with respect to time (frames) for a single representative fragment.

An example of a tracked fragment with an estimated position from the Kalman filtering process that matches well with the measured centroid positions is shown in Figure 2.15a. Without assuming a non-zero velocity to initialize Kalman filters for newly detected fragments in the field of view, velocity corrections are large and occur in the early time of tracking an individual fragment, as shown in Figure 2.15b. Figure 2.15c demonstrates that the uncertainty associated with the positions are large in the early time of the tracking process but converge to a small uncertainty rapidly. For the case of the fragment described in Figure 2.15a, there is a final position uncertainty of approximately 2.2%.

2.6 Fragment matching and 3D reconstruction via triangulation

Identifying the same fragment in both cameras is needed to reconstruct the 3D trajectory from the individual trajectories of the fragment in each camera. The process utilizes the epipolar geometry defined from the stereo calibration process. As described in Section 1.4, an epipole is a point of projection from the center of a camera into the epipolar plane connecting the epipole of the other camera in the stereo calibrated camera pair [19]. Epipolar lines are lines in an image plane in one camera view that corresponds to a point in the plane in-line with the optical center of the other camera.

According to epipolar geometry, a point x has a corresponding epipolar line l' that can be constructed via

$$l' = FP \tag{2.2}$$

in which the fundamental matrix F is the mapping between the two cameras. This is because in one camera, an object in 3D space may appear to be only a point, reflected in the singular point x in Figure 2.16 for the on Camera A's image when aligned with the optical center of the camera. However, along the ray between the optical center of Camera A and the point x and its projection into 3 space, the object represented as an epipole in Camera may correspond to many depths from Camera A but when observed by Camera B may correspond to many points on the image, represented by the green line in Figure 2.16. This is because the optical centers of each camera, epipoles, and epipolar lines all lie in the same plane, denoted in blue as the epipolar plane in Figure 2.16 .

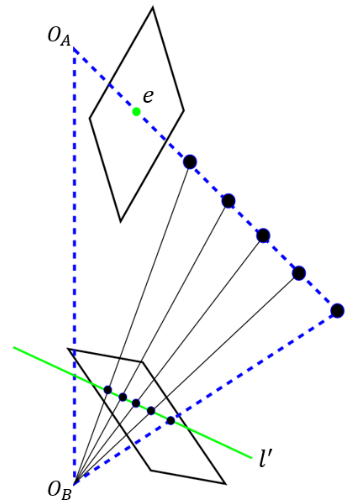
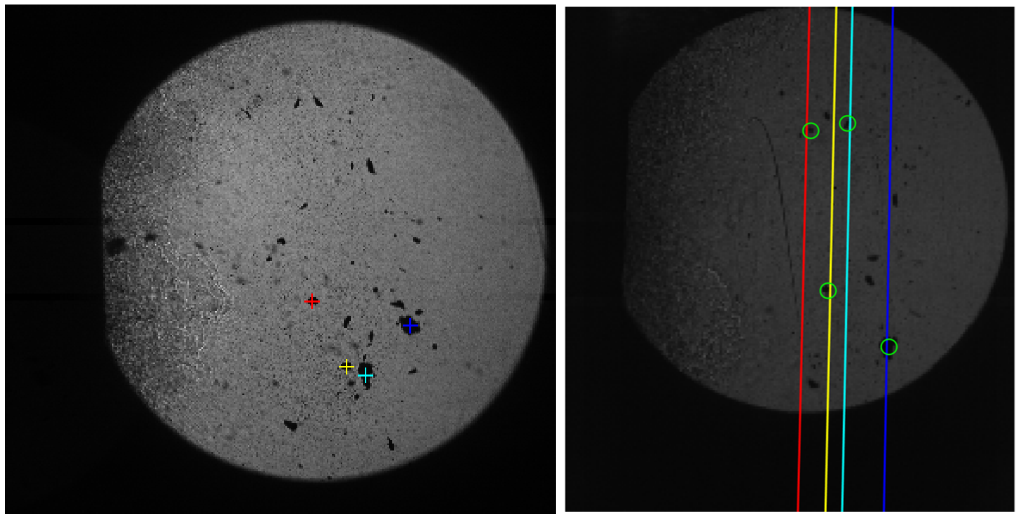


Figure 2.16: A diagram of the epipolar geometry defined by the stereo shadowgraph setup.

As seen in Figure 2.17, fragments are labelled in order to identify the same fragment with the track that describes its 2D trajectory in each image plane. The centroids, denoted with colored crosses that highlight tracked fragments in one camera view correspond to the same colored epipolar lines drawn in the other which overlap with the centroids of tracked fragments denoted with green circles in the other camera. If a line intersects multiple possible tracked fragments, each case is examined at different frames i.e. 20 frames apart and the line should intersect with a unique track in either camera due to differences in velocities of the fragments. Additionally, size can be used as another form of uniqueness when fragment matching.



(a) (b)

Figure 2.17: (a) Color coded centroids of fragments and epipolar lines associated with the corresponding centroids in (b) with fragments observed in the camera with reduced resolution

As described in Section 1.4, the triangulation of a point in 3D space is determined after finding two corresponding points in each stereo calibrated camera view. The triangulation process works by attempting to minimize the projected error as discussed in Section 1.4. This was performed by the MATLAB Stereo Calibration application and after determining the fundamental matrix F from the process, as described in Section 2.2, then using MATLAB's *triangulate* function.

2.7 3D Velocity Estimation

3D velocity estimation is performed by implementing a 3D Kalman filtering technique in 3D space using the reconstructed trajectory of each fragment. The choice to implement a simple 3D Kalman filter is made to aid in the process of determining the velocity associated with the reconstructed 3D velocities, without simply fitting a line in 3D space. The 3D Kalman filter is useful for smoothing out the noise associated with the reconstruction but also quantifying the uncertainty of the 3D trajectory and associated velocity estimated using the feedback loop of system state predictions and corrections. This Kalman filter uncertainty will be compared with the velocity uncertainty estimated via the propagated uncertainty of the 3D distances and time measurements in Section 2.8.

3D Kalman filtering utilizes the same governing equations for state prediction and state correction discussed in section 1.6 but uses matrices extended to account for the third spatial dimension. The 3D Kalman filter utilizes a transition matrix:

$$A = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

which describes a constant velocity projectile motion assumption for the tracked object for the duration observed. An acceleration model is not applied in this Kalman Filter model of fragment motion since the fragment is only observed for a short distance while in frame and significant velocity changes are not observed.

The observation matrix:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.4}$$

describes that three spatial positions are measured when applying the Kalman Filter process to the 3D trajectory. This matrix reflects that only position measurements are made directly and does not include direct velocity measurements at each discrete step in the Kalman filter for each fragment.

The 3D noise measurement matrix:

$$R = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_z^2 \end{bmatrix} \tag{2.5}$$

assumes uncorrelated and independent uncertainties associated with each spatial direction. The process noise matrix:

$$Q = \begin{bmatrix} \frac{\Delta t^4}{4} & 0 & 0 & \frac{\Delta t^3}{2} & 0 & 0 \\ 0 & \frac{\Delta t^4}{4} & 0 & 0 & \frac{\Delta t^3}{2} & 0 \\ 0 & 0 & \frac{\Delta t^4}{4} & 0 & 0 & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & 0 & 0 & \Delta t^2 & 0 & 0 \\ 0 & \frac{\Delta t^3}{2} & 0 & 0 & \Delta t^2 & 0 \\ 0 & 0 & \frac{\Delta t^3}{2} & 0 & 0 & \Delta t^2 \end{bmatrix} \tag{2.6}$$

also assumes uncorrelated and independent uncertainties with dependencies on time duration between discrete time steps.

A unique 3D Kalman filter is applied to each reconstructed trajectory using initial conditions of assuming the first predicted position is the first measured position. The assumed initial velocity that describes the motion of the fragment in

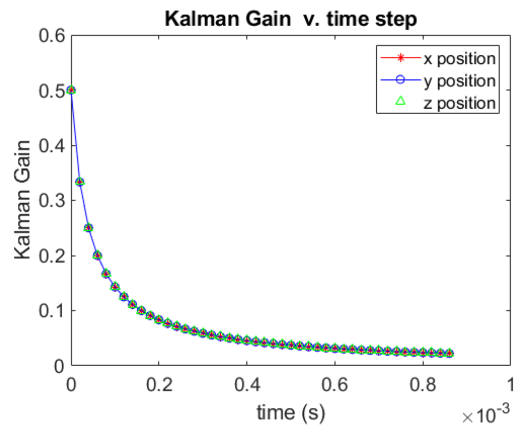
3D space is the mean of velocities estimated from the spatial gradient of the measured reconstructed points divided by the temporal difference of 20 microseconds between measurements in 2D space. Similar to the Kalman filter process observed in the two dimensional case, the uncertainty of the spatial positions in the 3D space converges to 2.2 % as seen in Figure 2.18b. The uncertainty for each triangulated point is determined by comparing the range of 3D positions in each direction obtained by calculating the triangulated point using each coordinate pair combination of image positions in the first stereo camera

$$\begin{gathered}(x_1 \pm \delta r_{px}, y_1) \\ (x_1, y_1 \pm \delta r_{px}) \\ (x_1 \pm \delta r_{px}, y_1 \pm \delta r_{px})\end{gathered} \tag{2.7}$$

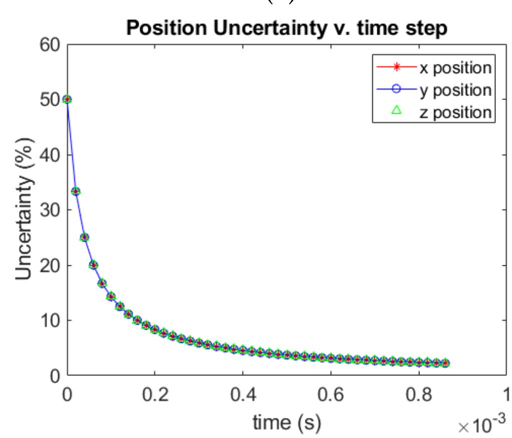
and positions in the second stereo camera

$$\begin{gathered}(x_2 \pm \delta r_{px}, y_2) \\ (x_2, y_2 \pm \delta r_{px}) \\ (x_2 \pm \delta r_{px}, y_2 \pm \delta r_{px})\end{gathered} \tag{2.8}$$

in terms of the reprojection error δr_{px}.



(a)



(b)

Figure 2.18: (a) The Kalman gain correction factor and (b) the uncertainty associated with each spatial direction demonstrating a convergence to a minimum value for a particular fragment.

The 3D Kalman filter applied to the triangulated trajectories starts with a balanced Kalman gain factor of 0.5 that quickly converges to values approaching zero, demonstrated in Figure 2.18a, for each respective coordinate direction. This convergence is reflected in the uncertainty in Kalman filtered position estimates for each coordinate direction, as shown in Figure 2.18b.The convergence of fragment Kalman filter state equation for the same fragment was observed to be within 10% after 10 sequential frames, displayed in Figure 2.18b. After 44 sequential frames of being tracked, the minimum value was approximately 2%, also displayed in Figure 2.18b. For the duration that a given fragment is tracked, measured trajectory (reconstructed from 2D Kalman filters) and the trajectory estimated via the 3D Kalman filter were within a 1% relative difference as shown in Figure 2.19.

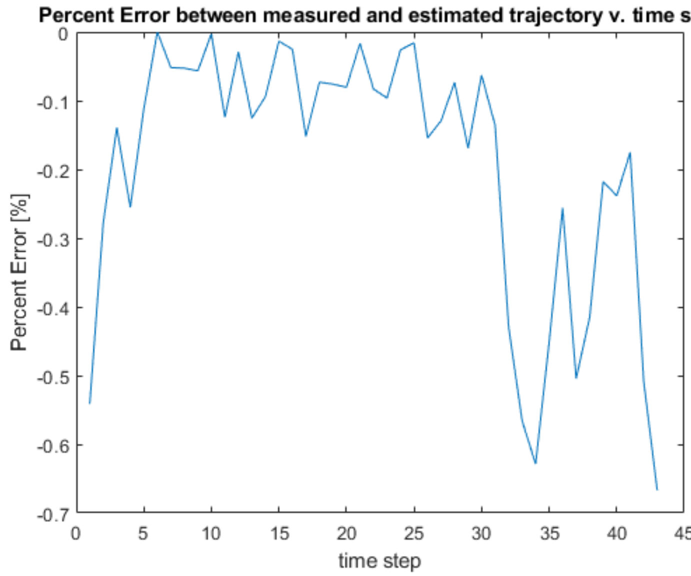


Figure 2.19: The percent difference between the measured trajectory and the trajectory estimated via the 3D Kalman filter process.

2.8 Uncertainty propagation

Potential sources of uncertainty in the measurements are associated with the spatial calibration scale, timing jitter in the laser pulse, temporal resolution of the high-speed cameras, the magnification in the non-parallel light section, as well as the pixel area. The uncertainty propagation of measurements are performed under linear error assumptions and independent uncertainties for the measured quantities. The following uncertainties in physical measurements follow the rules for uncertainty propagation described by Taylor [40]. The spatial calibration scale uncertainty is defined by the generalized uncertainty equation

$$\delta C_s = \sqrt{(\frac{\partial C_s}{\partial x_s}\delta x_s)^2 + (\frac{\partial C_s}{\partial p}\delta p)^2} \tag{2.9}$$

such that C_s in the calibration scale in mm/px, $\mathrm{C_s}$, x_s is the scale distance in mm, p is the distance in pixels, and δ is the uncertainty associated with the respective variable. The generalized uncertainty equation simplifies to:

$$\frac{\delta C_s}{C_s} = \sqrt{(\frac{\delta x_s}{x_s})^2 + (\frac{\delta p}{p})^2} \tag{2.10}$$

where the discretization uncertainty δp is found by assuming the pixels on the imaging sensor represent linear scale graduations, similar to that of a ruler. As such, the uncertainty is taken as half the graduation spacing, which is half a pixel. The fractional uncertainty of $\frac{\delta p}{p}$ is the controlling parameter for the spatial calibration uncertainty since it is much larger than the fractional uncertainty with the respect to the measurement uncertainty when measuring the calibration object, either via caliper as shown in the following section 2.9 or via calibration target grid tolerance. After performing spatial calibration to determine the size of objects, the observed calibration scales, or resolutions, for the two cameras with respect to the calibration plane were 0.20 ± 0.03 mm/px and 0.36 ± 0.03 mm/px respectively.

The fractional uncertainty of the equivalent diameter in mm is represented by:

$$\frac{\delta d_e}{d_e} = \sqrt{(\frac{1}{2}\frac{\delta A_{px}}{A_{px}})^2 + (\frac{\delta C_s}{C_s})^2} \tag{2.11}$$

such that the spatial calibration scale C_s uncertainty is accounted for. The derivation of this general uncertainty equation is described in Appendix B. When accounting for the size estimation uncertainty of fragments or other objects, the controlling parameter is the fractional uncertainty $\frac{\delta A_{px}}{A_{px}}$ because the method in which the pixel area is counted is dependent on the between-class variance peak and the varying background intensity in which a fragment may be travelling through including through fine particle clouds, product gas clouds, and other changes to background illumination. Additionally, the uncertainties with respect to the calibration scale are between 1-2%. The fractional uncertainties with respect to equivalent diameter for the smallest fragments was approximately 16% (0.1mm) for smaller fragments and 18% (0.5mm) for larger fragments, as shown in Table 2.2. The controlling parameter for both small and large fragments was the uncertainty due to the calibration scales; however, for the smaller fragments, the uncertainty in pixel area did have a non-negligible contribution to the overall equivalent diameter uncertainty.

The fractional uncertainty of magnification of object sizes in the shadowgraph system is given by:

$$\frac{\delta h_o}{h_o} = \sqrt{(\frac{\delta L_i}{L_i})^2 + (\frac{\delta L_o}{L_o})^2 + (\frac{\delta h_i}{h_i})^2} \tag{2.12}$$

such that h_o and L_o are the object size and distances and h_i and L_i are the image size and distances respectively. The distance uncertainties are essentially the triangulation uncertainties of the object from Camera 1 as well as the calibration plane to Camera 1. The uncertainty for each triangulated point is determined by comparing the range of 3D positions in each direction obtained by calculating the triangulated point using each coordinate pair combination of image positions in the first stereo camera via equations 2.7 and 2.8, in terms of the reprojection error δr_{px}, as described in Section 2.7.

Table 2.2: Limits for uncertainty of equivalent diameter with respect to pixel area uncertainty for the smallest and largest fragments observed.

Area A_{px} (px)	δA_{px} (px)	eq. diam. d_e (mm)	δd_e (mm)
5	$\pm$1	.55	$\pm$0.1 (18%)
177	$\pm$10	3.1	$\pm$0.5 (16 %)

Generally, the uncertainty in the out of plane direction (z coordinates) was larger than the uncertainties in x and y coordinates in 3D space. The largest determined fractional uncertainties for $\frac{\delta x}{x}$, $\frac{\delta y}{y}$, and $\frac{\delta z}{z}$ were 1%,1%, and 3%, respectively. This would suggest that δL_i and δL_o were approximately 3.5%. The controlling parameter was therefore δh_i, since it is the equivalent diameter uncertainty which ranged between 16% (0.1mm) for smaller fragments and 18% (0.5mm) for larger fragments.

The fractional 3D velocity uncertainty is given by

$$\frac{\delta v_{3D}}{v_{3D}} = \sqrt{(\frac{\delta x_{3D}}{x_{3D}})^2 + (\frac{\delta y_{3D}}{y_{3D}})^2 + (\frac{\delta z_{3D}}{z_{3D}})^2 + (\frac{\delta t}{t})^2} \tag{2.13}$$

which is dependent on the uncertainty of each distance component from the 3D reconstruction. The uncertainty of each position component is determined from the 3D Kalman filter process discussed in Section 2.7. This includes the measurement uncertainty from the triangulation process for each triangulated point using each coordinate pair combination between the two cameras in terms of the reprojection error δr_{px}. The jitter time of the SI-LUX 640 nm spoiled coherent laser was reported to be less than 5 ns, which is four orders of magnitude less than the interframe time of the recorded images. The triangulation uncertainties with respect to the 3D spatial coordinates were therefore controlling parameters of the uncertainties of the 3D velocities.

The larger velocities have a larger uncertainty than smaller velocities since the distances x_{3D}, y_{3D}, and z_{3D} measured are smaller than the trajectories of longer trajectories associated with slower fragments. This is observed despite the distances being significantly larger than their respective uncertainties, which were determined using equations 2.7 and 2.8, in terms of the reprojection error δr_{px}, as described in Section 2.7. Using this method to determine the uncertainty in velocities, the fastest fragments had a fractional uncertainty in velocity of 2% (7 m/s) and the slower fragments had a fractional uncertainty of 1% (0.2 m/s), as shown in Table 2.3. When comparing this uncertainty to the uncertainty obtained from the 3D Kalman filter, for the system converging to position uncertainties of approximately 1-2% for each spatial direction, the 3D Kalman filtered velocity was approximately 2.5%, especially for fragments tracked over many frames such as in the case seen in Figure 2.18b.

Table 2.3: Limits for uncertainty of 3D velocities for the slowest and fastest fragments.

x_{3D} (m)	δx_{3D} (m)	y_{3D} (m)	δy_{3D} (m)	z_{3D} (m)	δz_{3D} (m)
0.019	$\pm 1e-4$	0.023	$\pm 1e-4$	0.011	$\pm 2e-4$
0.012	$\pm 1e-4$	0.01	$\pm 1e-4$	0.006	$\pm 2e-4$
t (s)	δt (s)	v_{3D} (m/s)	δv_{3D} (m/s)		
1.85e-3	$\pm 5e-9$	17	$\pm 0.2(1\%)$		
5e-5	$\pm 5e-9$	334	$\pm 7(2\%)$		

2.9 Experimental Validation

Experimental validation was performed using a stereo-calibrated shadowgraph system to demonstrate that object sizes and dimensions can be determined from their projections in each camera view and are comparable to other measurement methods. The validation setup is shown in Figure 2.20 in which a pair of Phantom V711 cameras in stereo shadowgraph systems are stereo calibrated in the same plane between two optical tables such that each shadowgraph system intersects with an approximate 30 ° angle between them.

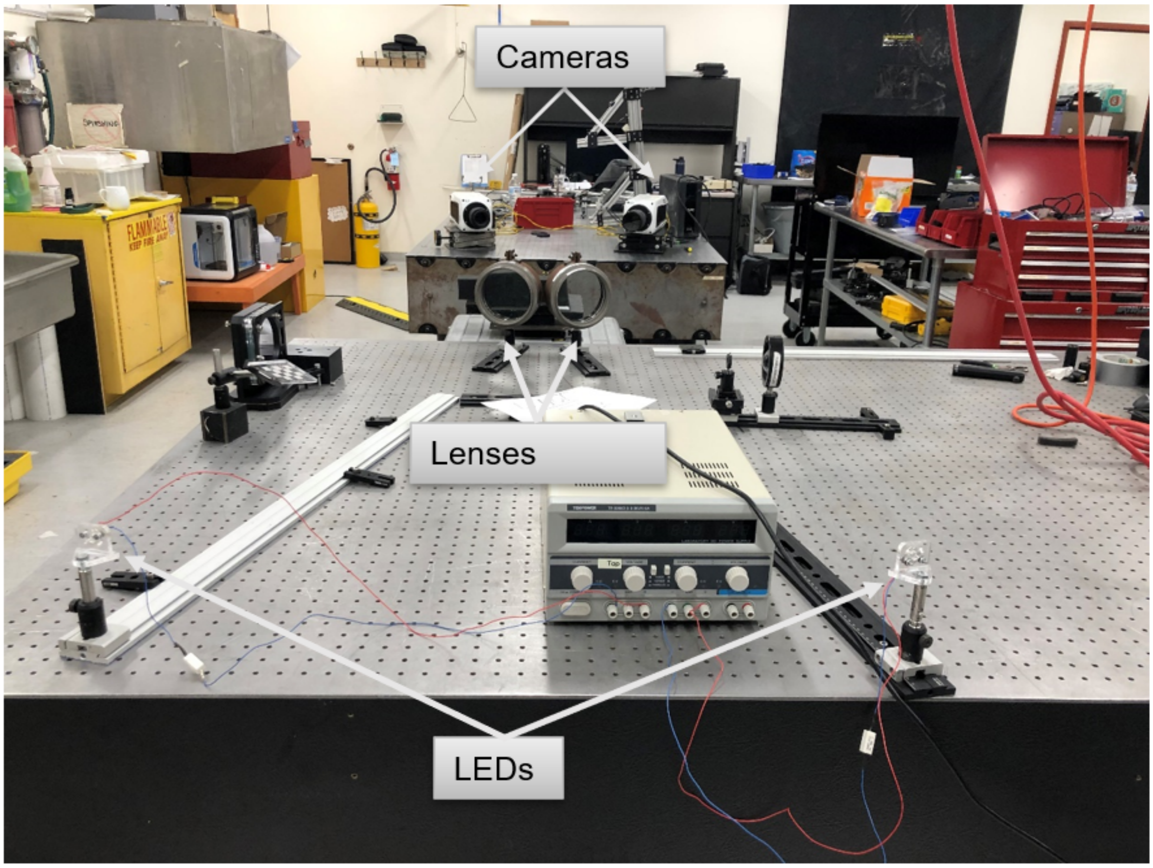
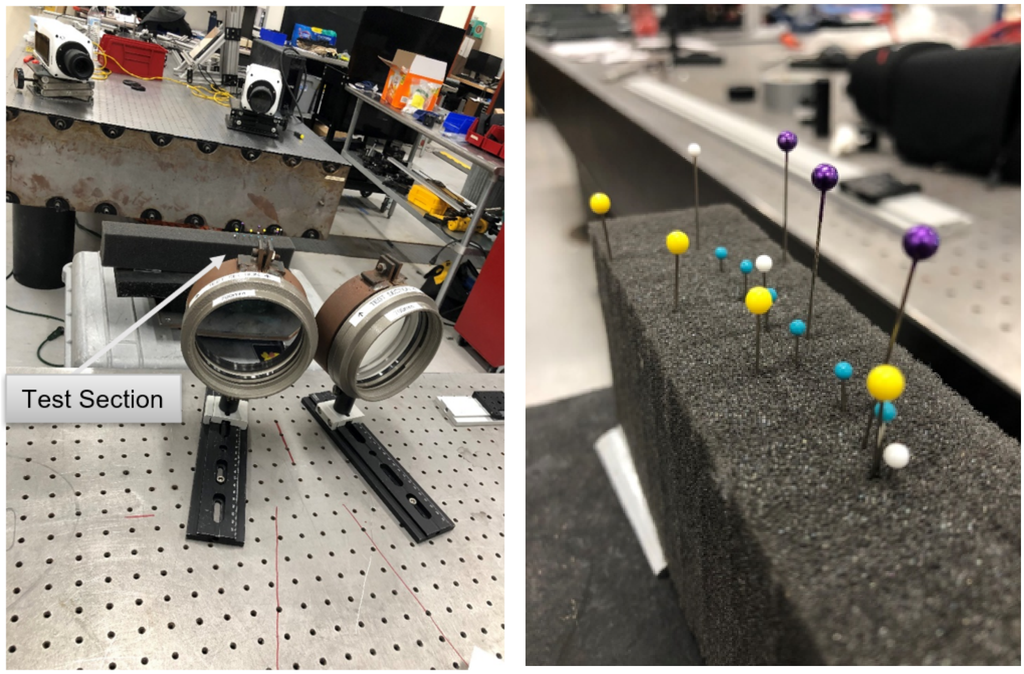


Figure 2.20: An experimental validation setup constituted by LEDs, lenses, and Phantom V711 cameras to implement stereo shadowgraphy.

The test section between the two optical tables is shown in Figure 2.21a in which a series of pinheads stuck into a piece of foam was placed in the test section, visualized in 2.21b. This stereo shadowgraph setup is intended for quantifying the ball end pin diameters optically, via the equivalent spherical assumption and extracted pixel areas, and comparing the resulting diameter with measurements performed via caliper.



(a) (b)

Figure 2.21: Test section with pinhead setup.

A combination of two sizes of ball end pins and varying distances from each camera are placed in the test section to demonstrate that the methodology presented in this work is appropriate. The varying distances of the objects from the camera are visualized using MATLAB's *viscircles* in Figure 2.22. The pixel areas of the ball end pins are extracted followed by a calculation of the equivalent spherical diameter, a spatial calibration conversion from pixels to millimeters, and an account for the object size from the magnification due to the lens and distance of the objects from the camera as well as the distance of the calibration plane from the camera, as done for all ballistic tests. The distance measurements were performed optically by determining the normalized distance of the 3D position for each ball end pin from the camera sensor. This position was triangulated using the centroid of each circle observed in each stereo camera.

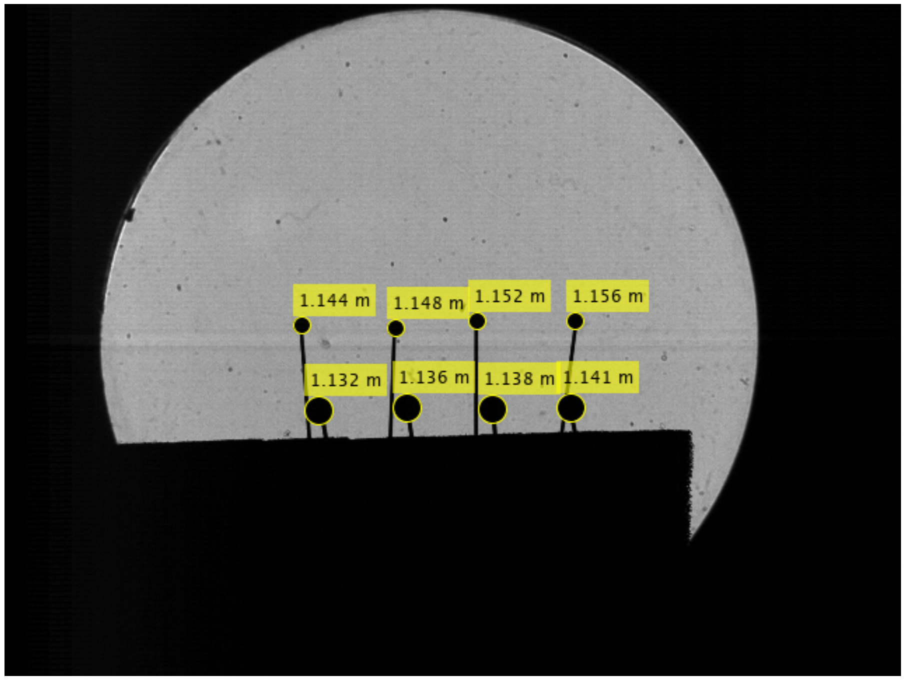


Figure 2.22: Ball end pins with their diameters denoted by yellow circles and annotated with their respective distances from one of the cameras.

A tabulated summary of the diameter of the pin heads is shown in Table 2.4 such that the associated uncertainties of the measurements optically are determined from the fragment size uncertainty propagation equations discussed in Section 2.8.

Table 2.4: Comparison of diameters dimensions of Ball End pins measured via caliper and optically.

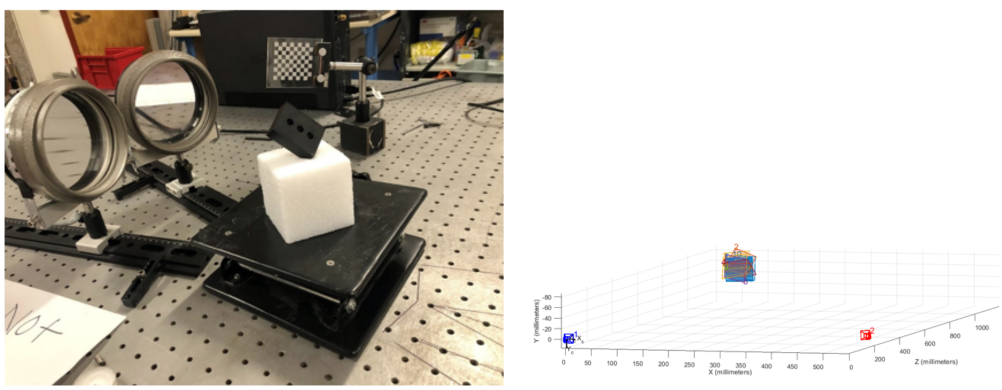
	Caliper	Optically Measured Size
blue pin 1	2.49 ± 0.01 mm	2.5 ± 0.2 mm
blue pin 2	2.45 ± 0.01 mm	2.6 ± 0.2 mm
blue pin 3	2.48 ± 0.01 mm	2.5 ± 0.2 mm
blue pin 4	2.45 ± 0.01 mm	2.5 ± 0.2 mm
purple pin 1	4.06 ± 0.01 mm	4.2 ± 0.2 mm
purple pin 2	4.10 ± 0.01 mm	4.1 ± 0.2 mm
purple pin 3	4.04 ± 0.01 mm	4.1 ± 0.2 mm
purple pin 4	4.06 ± 0.01 mm	4.2 ± 0.2 mm

Table 2.4 shows that with consideration of the uncertainties of the measurements, the methods produce equivalent diameter measurements. Following the validation of ball end pin sizes to mimic the fragment size estimation process, a validation of reconstruction of triangulated points using objects with known dimensions in 3D space is performed using a block and a stereo shadowgraph system. Table 2.5 show that the methods, via measuring tape and optically, produce

Table 2.5: Comparison of triangulated distances of Ball End pins and distances measured via caliper of the same block.

	Measuring Tape	Optically Measured Distance
blue pin 1	1.143 ± 0.003 m	1.144 ± 0.002 m
blue pin 2	1.147 ± 0.003 m	1.148 ± 0.002 m
blue pin 3	1.151 ± 0.003 m	1.152 ± 0.002 m
blue pin 4	1.156 ± 0.003 m	1.156 ± 0.002 m
purple pin 1	1.132 ± 0.004 m	1.132 ± 0.002 m
purple pin 2	1.135 ± 0.004 m	1.136 ± 0.002 m
purple pin 3	1.139 ± 0.004 m	1.138 ± 0.002 m
purple pin 4	1.142 ± 0.004 m	1.141 ± 0.002 m

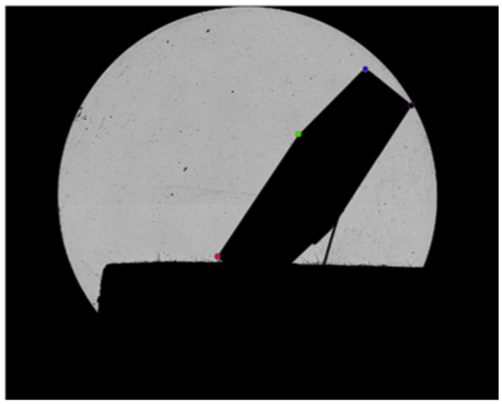
equivalent distance measurements of the ball end pins from the camera. The uncertainty in the measurement of the distance via measuring tape was larger that the optical method in this case since the measurement required placement of the measuring tape atop the ball end pins. The physical size of the ball end pins was on the order of tenths of a centimeter, with the conservative uncertainty being approximately the diameter of the ball end pins.



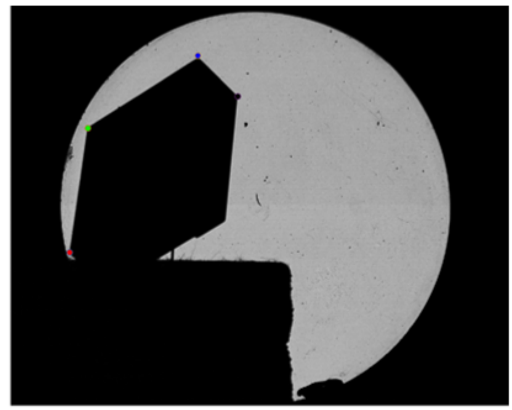
(a) (b)

Figure 2.23: A block placed in a stereo-calibrated shadowgraph system serves as a validation object for 3D reconstruction.

A separate test to quantitatively validate the position reconstruction and distance measurements optically by placing a block in the test section of a stereo shadowgraph system, shown in Figure 2.23a. A stereo calibration is applied to construct the extrinsic properties of the stereo shadowgraph setup associated with the block dimension reconstruction validation test is observed in Figure 2.23b. The goal of this validation test is to measurement of the dimensions of a block with known length, width, and height optically and compare with the dimensions measured via caliper. The block has shadowgraph projections in each camera view and it is oriented such that the 3 unique dimensions of the block are visible. The unique vertices are identified to be the same corner of the block, as shown in Figure 2.24a and 2.24b, and is triangulated in 3D space with respect to one of the cameras.



(a)



(b)

Figure 2.24: Unique vertices of block are identified and denoted with corresponding colored points to indicate the same vertex seen in each camera view.

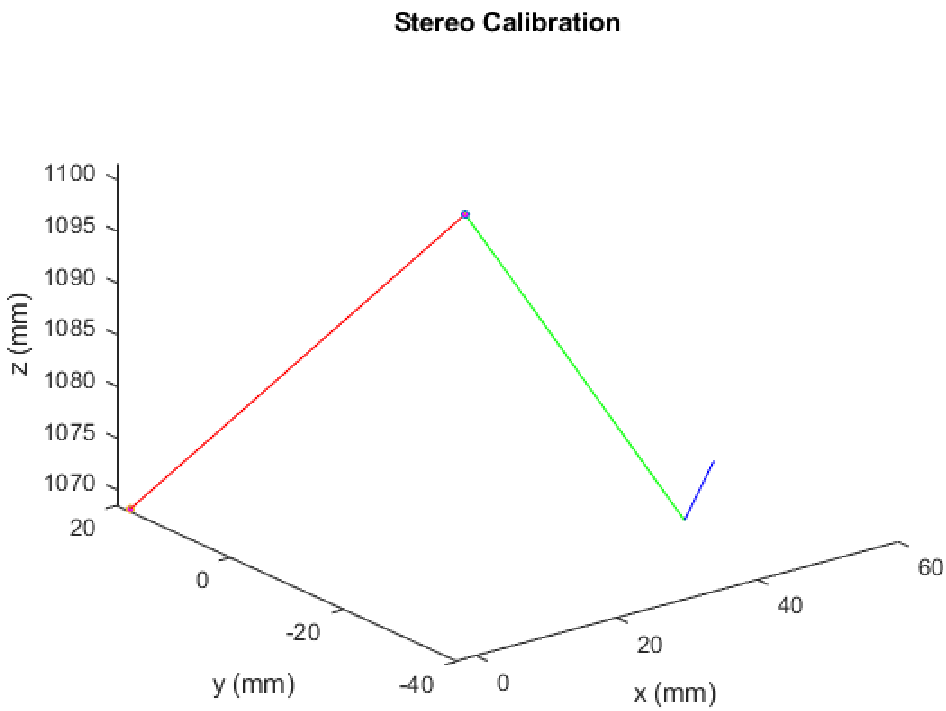


Figure 2.25: The reconstruction of the block dimensions with red, green, and blue lines corresponding to the length, width, and thickness dimensions of the block.

Connecting the points in 3D space allow for the reconstruction of the dimensions of the block optically as shown in Figure 2.25. A tabulated summary of the dimensions of the block is shown in Table 2.6 such that the associated uncertainties of the measurements optically are determined from the triangulation sensitivity for each reconstructed point. The table shows that with consideration of the uncertainties of the measurements, the methods produce equivalent measurements of lengths. Generally, the uncertainty in the out of plane direction (z coordinates) was larger than the uncertainties in x and y coordinates. The measured dimensions for the block all show similar uncertainties because the block was oriented at an arbitrary angle accounting for all of these differences.

Table 2.6: Comparison of triangulated dimensions of block and dimensions measured via caliper of the same block.

Dimension	Caliper	Triangulated
length	50.84 ± 0.01 mm	50.7 ± 0.3 mm
width	38.03 ± 0.01 mm	37.5 ± 0.4 mm
thickness	15.69 ± 0.01 mm	15.8 ± 0.2 mm

CHAPTER 3

EXPERIMENTAL RESULTS

3.1 3D trajectory reconstruction and velocity estimation

For several ballistic impact-fragmentation tests, numerous large fragments were observed rebounding off the steel anvil through fine particle clouds. From the automated tracking procedure, fragments were tracked for each stereo camera, as shown in Figure 3.1 with colored 2D trajectories for each camera. Figure 3.1 shows that for some example frames 200 microseconds apart, the velocities assigned to the fragment 2D trajectories correspond to the 3D velocities obtained from the 3D Kalman filters. Generally, the majority of observed fragments travelled slowly at velocities below 50 m/s, with only a few fragments travelling faster with yellow or green color tracks, as noted by the color bar in Figure 3.1.

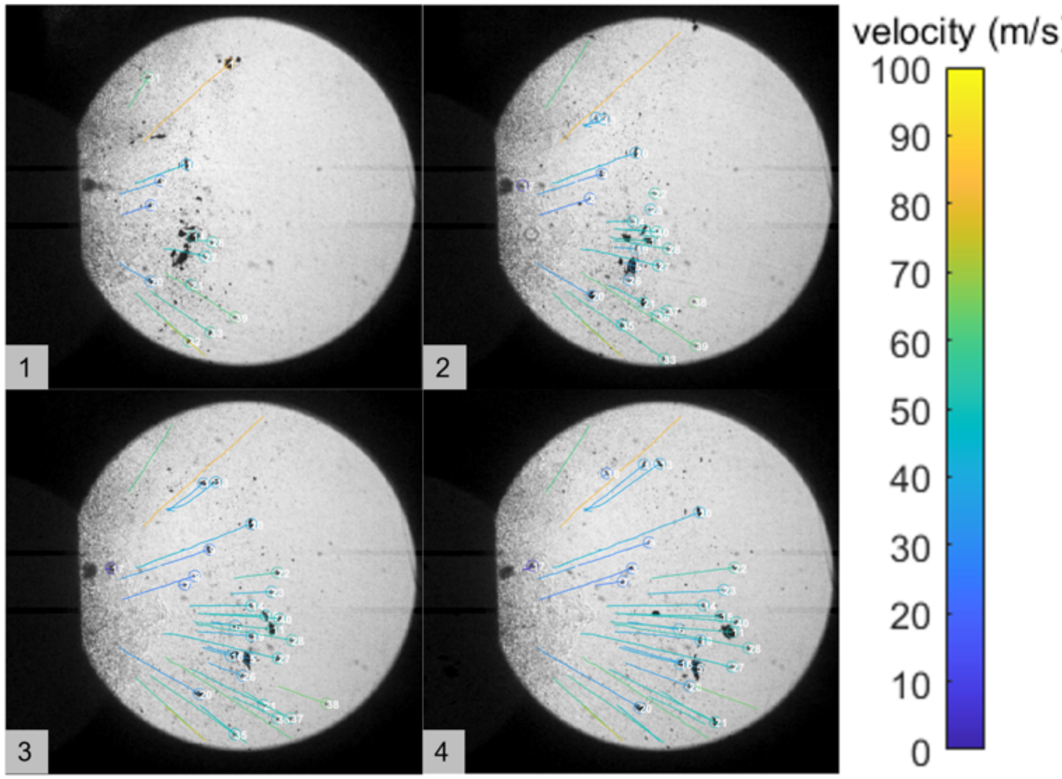


Figure 3.1: 4 non-sequential frames of tracked fragments in the high-fidelity camera 200 microseconds apart.

From the automated tracking algorithm, the 3D Kalman filtering process was applied to each reconstructed trajectory, as shown in Figure 3.2 for a single particle track. The smooth Kalman filtered trajectory is denoted in blue whereas the positions from the stereo reconstruction is denoted in green. The percent errors between the Kalman filtered trajectory and reconstructed trajectory in the position for this trajectory was approximately within 2%.

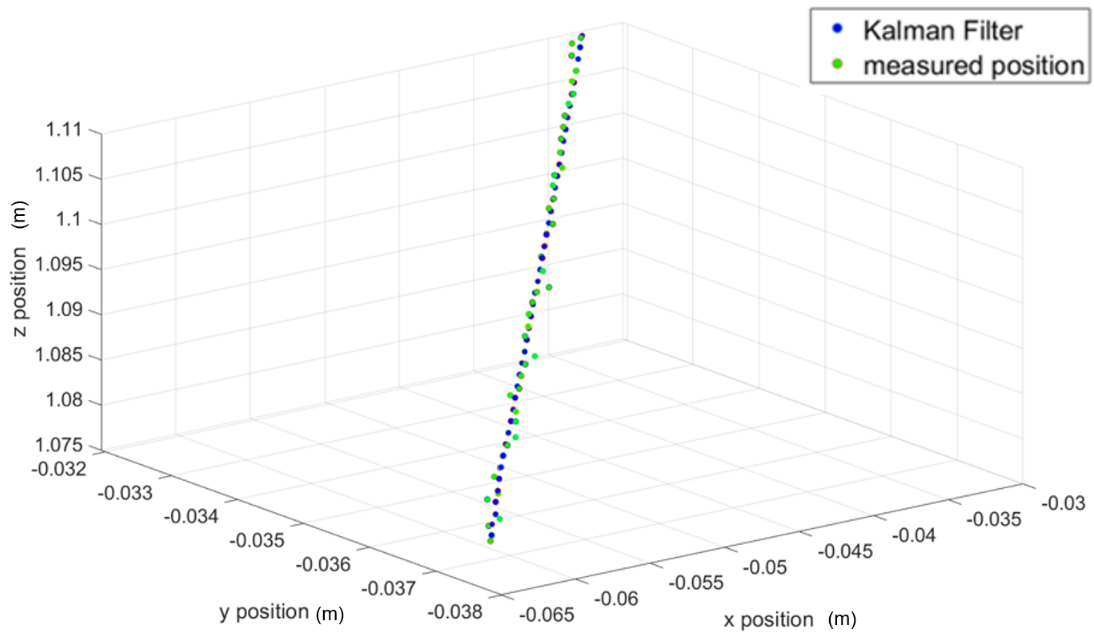


Figure 3.2: Comparison of measured and estimated 3D trajectory for a representative fragment.

For a series of impact tests, Figure 3.3 shows 3D reconstructions for many fragments travelling away from the point of contact and the 3D path of the incident projectile denoted with black data points. The fragment trajectories are colored corresponding to the color bar mapping their individual velocities. The number of reconstructed fragment trajectories varied such that each test shown in Figure 3.3 (a-e) had 43, 43, 37, 32, and 20 reconstructions, respectively. The range in velocities of the same fragments was from 17 to 334 m/s. Generally, the fragments with large angles from the x-axis had a tendency to travel at larger velocities. The back-projected trajectories of the fragments approximately showed an origin near the impact point of the plate coinciding with the trajectory of the incident projectile.

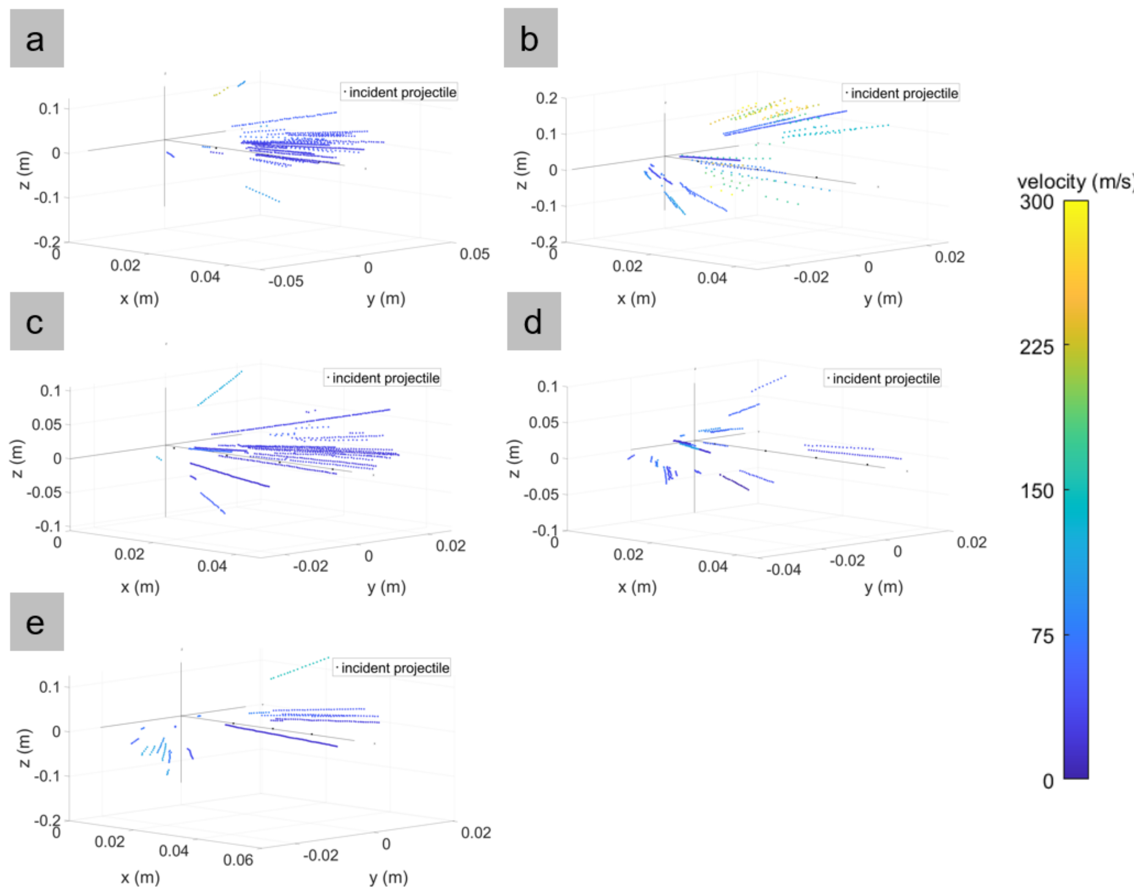


Figure 3.3: Reconstructions for multiple impact tests and their associated velocities.

The impact-fragmentation tests performed had a wide variation of observed fragmentation behavior due to the variations in powder loads for launching the projectiles, unintended induced angles of attack, as well as the occasional pre-impact fragmentation from inside the barrel or after exiting the sabot stripper. The impact fragmentation resulted in large fragments as well as fine particle clouds. The overlap of fragment projections, especially in the early time of tracking fragments also mitigated the number of successful reconstructions. The successful reconstructions were observed to have been the largest fragments in the shared optical test section region. The mismatched image resolutions for the cameras effectively reduced the amount of successful reconstructions. From individual cameras, the smallest fragment size successfully 3D reconstructed had a pixel area of 2-5 pixels. The reliability of this small size tracking was significantly limited by the different camera resolutions which were also effectively reduced due to the Bayer filter and monochromatic illumination.

Generally, incident projectiles with apparent unintended induced angles of attack resulted in fragments with a tendency to have large velocity z components. Fragments with significant angles from the x-axis travelled within the field of view for a significantly shorter time than fragments with smaller angles, limiting the ability to reconstruct their trajectories due to limited visibility.

Uncertainty propagation for the reported 3D velocities was performed by accounting for the spatial uncertainties in the trajectory reconstruction process. The propagation of the reprojection error in either camera had a mean value of δr_{px} = 0.34 px which is applied to each coordinate pair combination between the two cameras to determine the error associated with the triangulation process for each triangulated point, as described by equations 2.7 and 2.8 in Section 2.7. The largest determined fractional uncertainties for $\frac{\delta x_{3D}}{x_{3D}}$, $\frac{\delta y_{3D}}{y_{3D}}$, and $\frac{\delta z_{3D}}{z_{3D}}$ were 1%,1%, and 2%, respectively. The jitter time of the SI-LUX 640 nm spoiled coherence laser was reported to be less than 5 ns. The uncertainty of the velocity obtained from the 3D Kalman filter process and the uncertainty propagation of the spatial coordinates and time was thus approximately 1-2% for the velocity field shown in Figure 3.3, as discussed in Section 2.8.

3.2 Fragment Size Estimation and Bivariate Histograms

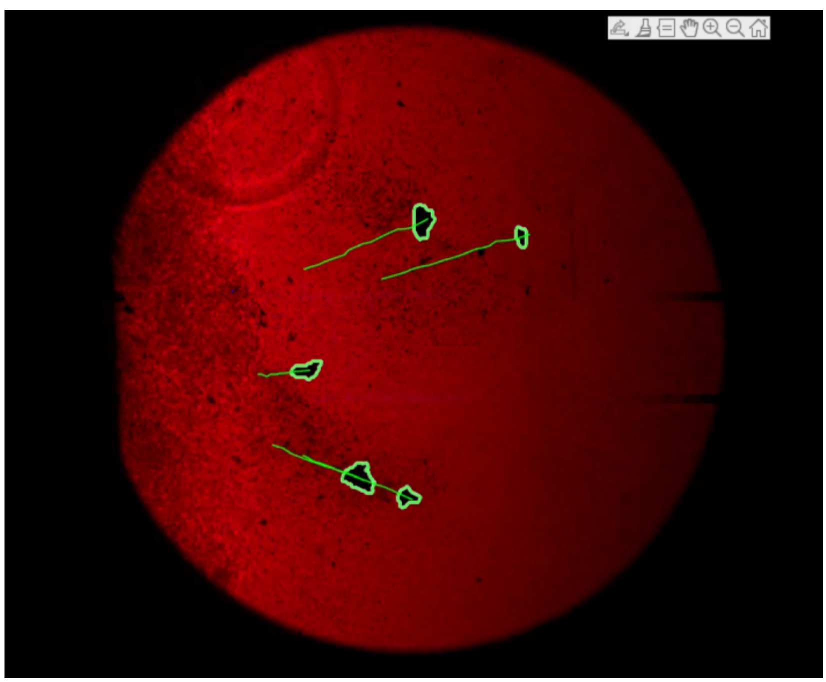


Figure 3.4: Boundaries highlighting the measured areas of the 5 largest fragments indicated in green.

The non-parallel light of the shadowgraph systems affects the projected size of the objects as they pass through the test section due to the magnification of the objects. The fragment projection sizes are determined by finding the magnified size with respect to the calibration plane. A similar triangle geometry is assumed using the distance from the camera to the objects and the distance of the camera to the calibration plane to account for magnification. The depth of the objects, or distance from the primary camera, are visualized in Figure 3.5.

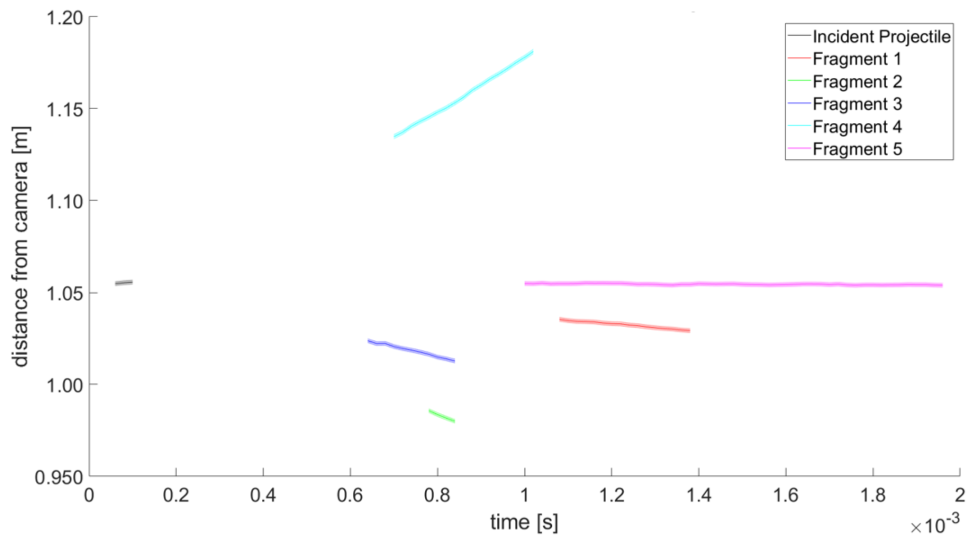


Figure 3.5: Distance of the incident projectile and the fragments shown in figure 3.4 from the camera as a function of time.

For each fragment, the uncertainty in the pixel area is determined by finding the range of maximum between-class variance values associated with the range of threshold values in Otsu's method. This range of thresholds is used to observe the range in pixel area possible using Otsu's method of binarization to obtain the pixel area error δA_{px}.

The area for each fragment is determined over the sequence of frames they are tracked through. Measured fragment pixel areas are presented in Figure 3.6 where the uncertainty is shown as the light colored region around solid line measurements. When plotted as a function of frame number, it is observed that particles have an oscillating pixel area. This is attributed to the particles having a non-spherical shape and rotating about multiple axes. For a sequence of images, the equivalent spherical assumption is utilized then averaged over the time that a representative fragment is tracked simultaneously in each camera. The conversion from pixel area to equivalent spherical diameter for a representative fragment, and the associated time-averaged equivalent spherical diameter, is shown in Figure 3.7. Motion blur did not affect particle sizes since for the largest fragment velocities observed, the distance travelled would have been approximately on the order of microns. This distance is significantly smaller the resolution of the cameras while using the 20 ns pulse width of laser illumination.

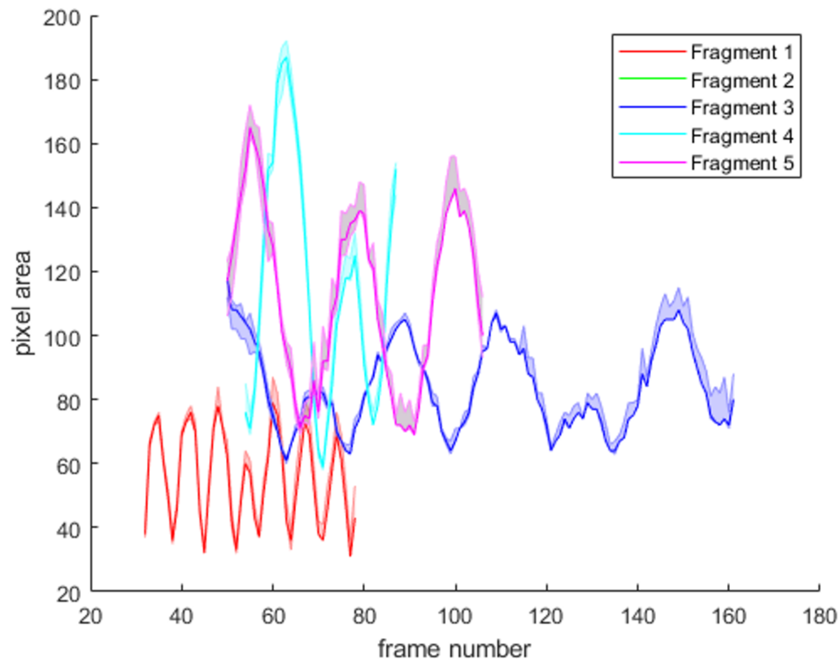
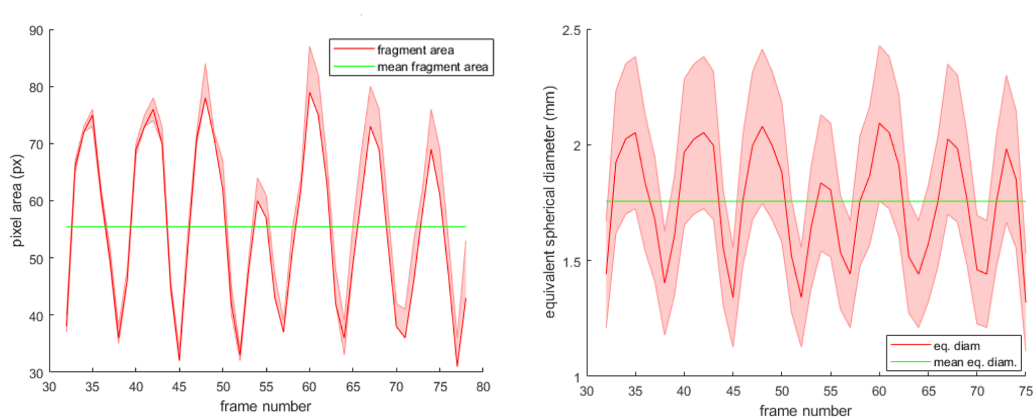


Figure 3.6: The pixel area and associated uncertainty with time for 2D projections of 5 representative tracked fragments.

Examples of multiple fragment equivalent spherical diameters with respect to time are visualized for 5 fragments tumbling at different rates in Figure 3.8. The difference in the size of peaks and valleys of the observed oscillatory behavior reflect the rate of tumbling on different axes.



(a) (b)

Figure 3.7: (a) Pixel area for a fragment and (b) equivalent diameter for a representative fragment with respect to frame number.

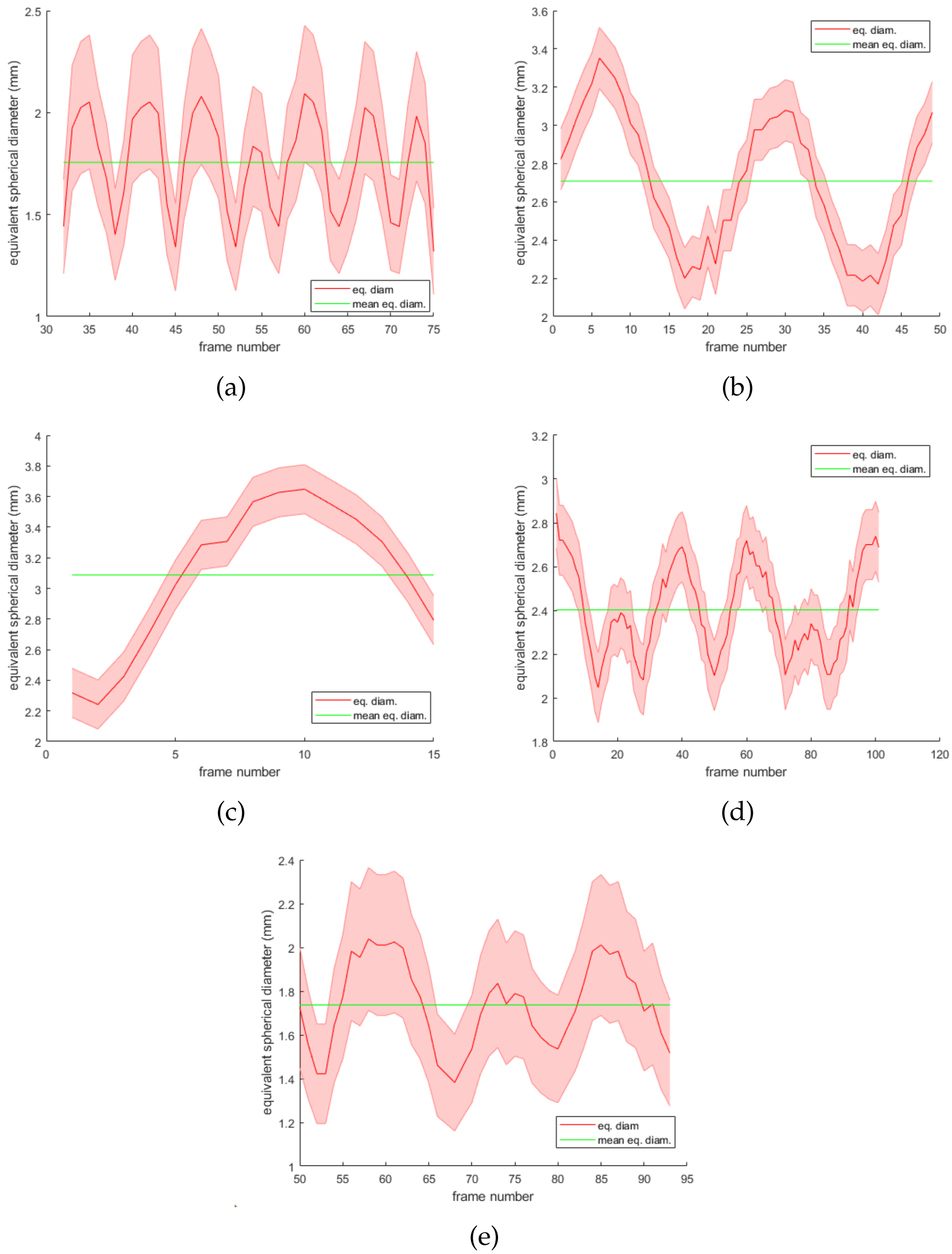


Figure 3.8: The equivalent spherical diameters of the same fragments in Figure 3.4 tracked simultaneously in each camera

For the same impact fragmentation tests shown in Figure 3.3, bivariate histograms describe the frequency of reconstructed fragment trajectories with respect to the fragment velocities and the mean equivalent spherical diameter. Simultaneously tracked fragments and their respective sizes and velocities are visualized in bivariate histograms in Figure 3.9. For the data in Figure 3.9, the mean estimated equivalent spherical diameter sizes ranged from approximately 0.5 to 3.1 mm in size. The majority of fragments across multiple impact-fragmentation tests had a size between 0.5 and 1.5 mm. The majority of fragments travelled at velocities between 17 to 100 m/s. While the fragments larger than 1.5 mm tended to travel at speeds between 17 to 50 m/s and the smaller fragments between 0.5 to 1.5 mm in size travelled between the full range of observed velocities (17 to 340 m/s).

Table 3.1 summarizes the mass and velocities of the ballistically launched cylindrical aluminum projectiles, and their respective momentum and kinetic energy. For comparison, after impact the images were analyzed to measure the resulting mass, momentum, and kinetic energy of the fragments. The mass for each fragment

$$m = \rho \frac{\pi}{6} d_e^3 \tag{3.1}$$

is comprised of the uniform density ρ assumption and the spherical volume calculation in terms of the equivalent spherical diameter d_e. Momentum for each fragment is calculated using

$$p = m v_{3D} = (\rho \frac{\pi}{6} d_e^3) v_{3D} \tag{3.2}$$

in terms of mass and the 3D velocity of the fragment, v_{3D}. The kinetic energy for each fragment is calculated by

$$T = \frac{1}{2} m v_{3D}^2 = \frac{1}{2} (\rho \frac{\pi}{6} d_e^3) v_{3D}^2 \tag{3.3}$$

with the same assumptions. The 2D fragment projections are used to calculate a total mass of the three-dimensionally tracked fragments after impact, which is calculated by summing the masses of each fragment. Similarly, the total momentum of the three-dimensionally tracked fragments is calculated by summing the momentum of each fragment and likewise for total kinetic energy.

A mass ratio is calculated from the division of the total mass of the tracked fragments by the known mass of the projectile. Similarly, the momentum ratio is calculated from the division of the total momentum of the tracked fragments by the momentum of the incident projectile. The kinetic energy ratio is calculated from the total kinetic energy of the tracked fragments by the kinetic energy of the incident projectile. Table 3.2 summarizes the optically recovered mass, momentum, and kinetic energy ratios for fragments for each impact test compared to the incident projectile's properties in Table 3.1. The total mass of the fragments with successfully reconstructed trajectories was compared to the mass of the incident projectile in reported mass ratios, ranging from 21% to 45%. After comparing the estimated mass from 2D projections of all detected fragments in one camera, the mass ratio was approximately 95% of the incident projectile's mass. The mismatched resolutions of the cameras significantly affected the number of successful reconstructions and the ability to resolve smaller matching fragments in both cameras simultaneously. The ratio of the total momentum of the fragments with successfully reconstructed trajectories to the momentum of the incident projectile ranged from 1.3% to 8.2%. The ratio of the total kinetic energy of the fragments with successfully reconstructed trajectories to the kinetic energy of the incident projectile ranged from 0.08% to 2.1%.

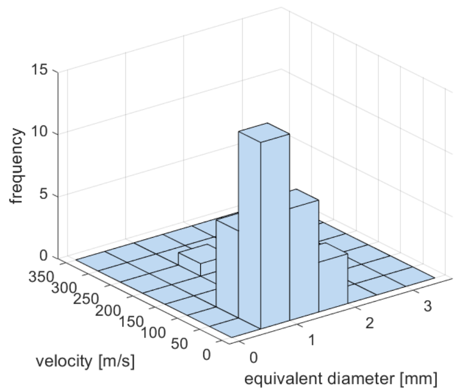
The resolution played a contributing factor in the ability to resolve the same fragments in each camera, which significantly impact the number of reconstructed fragment trajectories and the associated mass ratios, which were much smaller than the mass ratio of 95% of fragments tracked in a single camera. Similarly, the reported momentum and kinetic energy ratios are dependent on the masses of fragments with successfully reconstructed fragment trajectories and therefore the ratios reported in Table 3.2 are smaller than expected since the mass ratios reflects the reduced number of successful trajectories.

Table 3.1: A summary of the properties for each incident projectile.

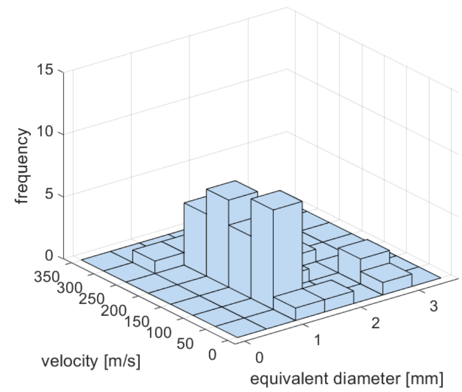
Test #	Mass (g)	Density (g/cm^3)	Velocity (m/s)	Momentum (kg m /s)	Kinetic Energy (J)
1	0.488	2.604	640	0.31	98.7
2	0.502	2.590	690	0.35	119
3	0.472	2.545	630	0.30	93.7
4	0.485	2.598	630	0.30	96.0
5	0.476	2.607	603	0.29	86.5

Table 3.2: A summary of the ratios of the fragment mass, momentum, and kinetic energy for fragments with successfully reconstructed 3D trajectories to the incident projectile.

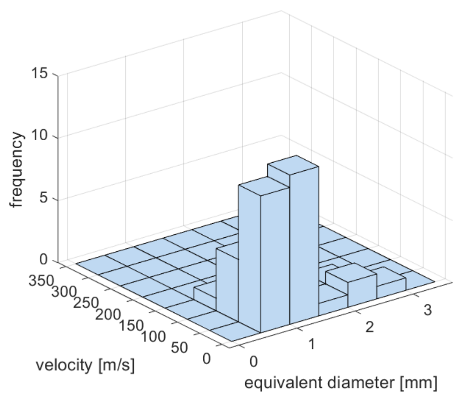
Test #	Mass Ratio	Momentum Ratio	Kinetic Energy Ratio
1	21%	1.4%	0.12%
2	45%	8.2%	2.1%
3	25%	1.3%	0.08%
4	41%	3.3%	0.36%
5	29%	2.4%	0.3%



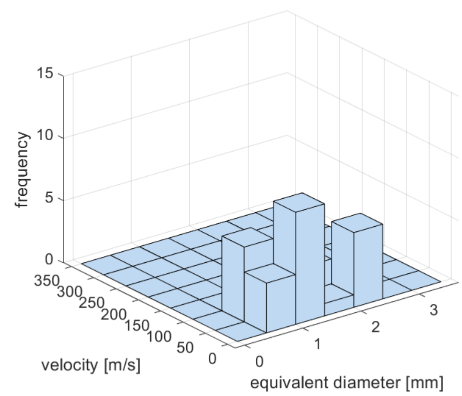
(a)



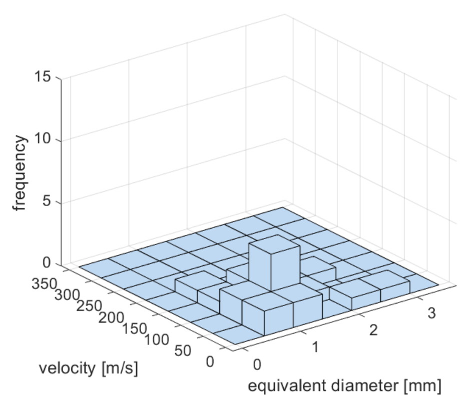
(b)



(c)



(d)



(e)

Figure 3.9: Bivariate histograms of the series of impact fragmentation tests.

CHAPTER 4

CONCLUSION

4.1 Conclusion

The methodology for simultaneous in-situ trajectory, velocity, and size estimates have been successfully developed for fragments resulting from ballistically-launched impacts of reactive material specimens over multiple tests. This work utilized stereo shadowgraph imaging to obtain 3D information of in-flight fragments after steel plate impacts. Digital image processing techniques, including image segmentation, were applied to high-speed shadowgraph images to obtain the location and projected area of fragments. Kalman filter-based tracking was implemented for each camera view and 3D trajectories were reconstructed from the triangulation of fragments, uniquely identified using the epipolar geometry of the stereo shadowgraph setup. Velocities of fragments were determined using 3D Kalman filters applied to the reconstructions. Successfully tracked fragments allowed for the measurement of the fragment sizes at each time step and visualization of the fragments rotating on multiple axes. After comparing the uncertainty of the velocity measurements via the 3D Kalman filters with the uncertainty propagation calculations, the former approached the latter fractional uncertainties, serving as a comparison tool that suggested the 3D velocities could be obtained solely from the trajectory reconstruction. For the simplicity in performing the uncertainty propagation calculations, estimation of the velocities directly from the reconstructed trajectory is preferred.

Some limitations of the optical diagnostics methodology explored in this work are significant differences in resolution between the two mismatched cameras used in the stereo calibration process as well as due to the color camera Bayer filter on resolution; however, this body of work suggests that mismatched cameras can still perform in-situ optical diagnostics despite limiting the minimum size of fragments tracked simultaneously in each camera. The difference in resolutions between the two cameras also contributed to the reduced number of successful 3D trajectory reconstructions.

The stereo shadowgraph system implemented in this work was successful in the goal of obtaining fragment area projections as well as depth information of fragments. The benefits of this intersecting configuration of single-lens focused shadowgraph systems includes the simplicity of the experimental setup and adjustments. The challenges of this system is the limitation of the field of view and the size of overall optical test section comprised from the intersecting converging light cones, both restricted by the size of the lenses utilized. Additionally, the occlusion of fragments was a source of impediment for tracking despite using two single-lens focused-shadowgraph orientations.

The unique measurements performed in this work includes the area measurement and the resolution of fragments rotating on multiple axes. This was achieved using the combination of the tracking algorithms and the precise area extraction via image segmentation techniques. Mean equivalent spherical diameters of fragments were determined with the consideration of the depth of objects in the test section, the magnification, and the tumbling of area projections.

Simultaneous velocities and fragment sizes for fragments are visualized in bivariate histograms to represent the result of the fragmentation behavior of RM samples undergoing high-velocity impacts. They provided insight into the most frequent size and velocities represented by fragments observed for an individual high velocity impact. The bivariate histograms were also useful for comparing the range of observed fragment sizes and velocities, from test to test, with varying incident projectile velocities.

4.2 Future Work

In future work, the optical diagnostic techniques can be applied to other studies of fragmentation behavior including explosive casings, warheads, ballistics of other compositions, etc. Additionally, future work includes the application of the PTV techniques discussed in this work using other shadowgraph techniques, including projective shadowgraphy in stereo, for larger scale field tests of fragmentation behavior. When applying tracking techniques to larger fields of view, future work could also apply an acceleration model of motion and quantify the force of drag on fragments, or use non-linear Kalman Filtering techniques. Comparisons of Kalman filter-based tracking to DIC techniques should also be explored.

Future work should consider applying different geometric assumptions when performing size analysis of fragments such as other polygonal geometries. The continuous measurement of fragment projection pixel area akin to a computerized particle size analyzer approach allows for observed oscillations in area that could be used in conjunction with other shape factors and geometric assumptions for size estimation of fragments. Improvements to the tracked size estimation approach could include the reconstruction of the 3D volume via a convex hull or a true shape from the rotation for fragments. The next steps for obtaining bivariate histograms from the size and velocity measurements of fragments as a result of high velocity impacts could include the true shape of fragments from the observed rotations.

REFERENCES

[1] Joseph P. Hooper. Impact fragmentation of aluminum reactive materials. *Journal of Applied Physics*, 112(4):043508, 2012. doi: 10.1063/1.4746788. URL `https://doi.org/10.1063/1.4746788`.

[2] Joseph P. Hooper, Christoper L. Milby, Richard J. Lee, and R. Jason Jouet. High-velocity impact fragmentation of brittle, granular aluminum spheres. *Procedia Engineering. Proceedings of the 12th Hypervelocity Impact Symposium*, 58: 663–671, 2013. ISSN 1877-7058. doi: https://doi.org/10.1016/j.proeng.2013.05.076.

[3] Michael J. Hargather, Jamie Kimberley, and Steven G. Thoma. Failure and fragmentation of pressed bi-metallic composites. In *AIP Conference Proceedings*, 2018. doi: 10.1063/1.5044974.

[4] S. H. Youngblood, B. Miller, C. Schmittle, M. J. Hargather, and J. Kimberley. Study of reactive material fragmentation behavior in gun- and explosive-launched systems. 2019.

[5] S.H. Youngblood, S. Palmer, J. Kimberley, and M. J. Hargather. In situ measurement diagnostics of the fragmentation behavior of powder composite reactive materials subjected to high-rate dynamic loading. 2021.

[6] S. C. Thuot, J. Wilkinson, R. J. Lee, J. R. Carney, J. Hooper, J. M. Lightstone, J. R. Jouet, J. G. Rogerson, Mark Elert, Michael D. Furnish, William W. Anderson, William G. Proud, and William T. Butler. Impact fragmentation and ballistics of pressed aluminum powder projectiles. In *AIP Conference Proceedings [AIP SHOCK COMPRESSION OF CONDENSED MATTER 2009: Proceedings of the American Physical Society Topical Group on Shock Compression of Condensed Matter*. AIP, 2009. doi: 10.1063/1.3294970.

[7] Jacob Kline and Joseph P. Hooper. The effect of annealing on the impact fragmentation of a pure aluminum reactive material. *Journal of Applied Physics*, 125(20):205901, may 2019. doi: 10.1063/1.5094444.

[8] Megan Tang and Joseph P. Hooper. Impact fragmentation of a brittle metal compact. *Journal of Applied Physics*, 123(17):175901, 2018. doi: 10.1063/1.5026711.

[9] Vitali F. Nesterenko, Po-Hsun Chiu, C.H. Braithwaite, Adam Collins, David Martin Williamson, Karl L. Olney, David Benson, and Francesca McKenzie. Dynamic behavior of particulate/porous energetic materials. In *AIP Conference Proceedings*. AIP, 2012. doi: 10.1063/1.3686334.

[10] William H. Wilson, Fan Zhang, and Kibong Kim. Fine fragmentation distribution from structural reactive material casings under explosive loading. In *AIP Conference Proceedings*. Author(s), 2017. doi: 10.1063/1.4971531.

[11] Edward L. Dreizin. Metal-based reactive nanomaterials. *Progress in Energy and Combustion Science*, 35(2):141–167, apr 2009. doi: 10.1016/j.pecs.2008.09.001.

[12] Po-Hsun Chiu and V. F. Nesterenko. Dynamic behavior and fracture of granular composite al-w. In *DYMAT 2009 - 9th International Conferences on the Mechanical and Physical Behaviour of Materials under Dynamic Loading*. EDP Sciences, 2009. doi: 10.1051/dymat/2009133.

[13] G. S. Settles. *Schlieren and Shadowgraph Techniques : Visualizing Phenomena in Transparent Media*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. ISBN 9783642566400.

[14] Gary S Settles and Michael J Hargather. A review of recent developments in schlieren and shadowgraph techniques. *Measurement Science and Technology*, 28(4):042001, 2017. doi: 10.1088/1361-6501/aa5748.

[15] Kyle O. Winter and Michael J. Hargather. Three-dimensional shock wave reconstruction using multiple high-speed digital cameras and background-oriented schlieren imaging. *Experiments in Fluids*, 60(6), may 2019. doi: 10.1007/s00348-019-2738-x.

[16] Kyle Jeffrey Benalil. Three-dimensional reconstruction of turbulent gases using stereo schlieren and shadowgraph techniques. Master's thesis, New Mexico Institute of Mining and Technology, 2018.

[17] Mark R. Shortis, James W. Seager, Euan S. Harvey, and Stuart Robson. Influence of bayer filters on the quality of photogrammetric measurement. In J.-Angelo Beraldin, Sabry F. El-Hakim, Armin Gruen, and James S. Walton, editors, *SPIE Proceedings*. SPIE, jan 2005. doi: 10.1117/12.588217.

[18] Rafael Gonzalez. *Digital image processing*. Prentice Hall, Upper Saddle River, N.J, 2008. ISBN 9780131687288.

[19] Andrew Zisserman Richard Hartley. *Multiple View Geom Comp Vision 2ed*. Cambridge University Press, February 2003. ISBN 0521540518. URL `https://www.ebook.de/de/product/3267382/richard_hartley_andrew_zisserman_multiple_view_geom_comp_vision_2ed.html`.

[20] J. Paul Siebert Boguslaw Cyganek. *An Introduction to 3D Computer Vision Techniques and Algorithms*. Paperbackshop UK Import, 2009. ISBN 047001704X.

[21] R.I. Hartley. In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):580–593, 1997. doi: 10.1109/34.601246.

[22] Charles Pecora. Particle tracking velocimetry: A review. mathesis, University of Washington, 2018.

[23] Daniel R. Guildenbecher, Elizabeth M.C. Jones, Elise M. Hall, Phillip L. Reu, Timothy J. Miller, Francisco Perez, Andrew D. Thompson, and James Patrick Ball. 3d optical diagnostics for explosively driven deformation and fragmentation. *International Journal of Impact Engineering*, 162:104142, 2022. doi: 10.1016/j.ijimpeng.2021.104142.

[24] Erkai Watson, Max Gulde, and Stefan Hiermaier. Fragment tracking in hypervelocity impact experiments. *Procedia Engineering*, 204:170–177, 2017. doi: 10.1016/j.proeng.2017.09.770.

[25] Erkai Watson, Hans-Gerd Maas, Frank Schäfer, and Stefan Hiermaier. Trajectory based 3d fragment tracking in hypervelocity impact experiments. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2:1175–1181, 2018. doi: 10.5194/isprs-archives-XLII-2-1175-2018.

[26] Tbd Subhash Challa. *Fundamentals of Object Tracking*. Cambridge University Press, July 2011. ISBN 0521876281. URL `https://www.ebook.de/de/product/13782883/subhash_challa_tbd_fundamentals_of_object_tracking.html`.

[27] Xin Li, Kejun Wang, Wei Wang, and Yang Li. A multiple object tracking method using kalman filter. In *The 2010 IEEE International Conference on Information and Automation*. IEEE, 2010. doi: 10.1109/icinfa.2010.5512258.

[28] Sawsen Rezig, Rosario Toscano, Gilles Rusaouën, and Vincent Lozano. Fuzzy kalman filter for 3d lagrangian particle tracking using multi-scale bubble detection. *Energy Procedia*, 78:3078–3083, 2015. doi: 10.1016/j.egypro.2015.11.760.

[29] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960. doi: 10.1115/1.3662552.

[30] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, and Tae-Kyun Kim. Multiple object tracking: A literature review. *Artificial Intelligence*, 293:103448, 2021. doi: 10.1016/j.artint.2020.103448.

[31] Kakuji Ogawara, Sei ichi Iida, Takuya Noborizato, and Kenji Asuwa. Application of kalman filter-type PTV to flow field with density change. *TRANSACTIONS OF THE JAPAN SOCIETY OF MECHANICAL ENGINEERS Series B*, 61(590):3679–3683, 1995. doi: 10.1299/kikaib.61.3679.

[32] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. doi: 10.1002/nav.3800020109.

[33] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957. doi: 10.1137/0105003.

[34] F.L. Pedrotti, L.S. Pedrotti, and L. M. Pedrotti. Introduction to optics (3rd ed.). *Pearson Prentice Hall. Pearson Education Inc.*, 2007.

[35] Dennis E. Grady and Marlin E. Kipp. Fragmentation properties of metals. *International Journal of Impact Engineering*, 20(1-5):293–308, jan 1997. doi: 10.1016/s0734-743x(97)87502-1.

[36] M. E. Kipp D. E. Grady. Impact failure and fragmentation properties of metals. techreport Technical Report SAND98-0387, Sandia National Laboratories, Sandia National Laboratories, Albuquerque, NM, USA,, 1998.

[37] Linzhu Li and Magued Iskander. Comparison of 2d and 3d dynamic image analysis for characterization of natural sands. *Engineering Geology*, 290: 106052, 2021. doi: 10.1016/j.enggeo.2021.106052.

[38] Richard Szeliski. *Computer Vision*. Springer London, 2011. doi: 10.1007/978-1-84882-935-0.

[39] Markus Buehren. Functions for the rectangular assignment problem. MATLAB Central File Exchange., 2022. URL `https://www.mathworks.com/matlabcentral/fileexchange/6543-functions-for-the-rectangular-assignment-problem`.

[40] John R. Taylor. *Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements*. Univ Science Books, 1997. ISBN 0935702423.

[41] Béla Sokoray-Varga and János Józsa. Particle tracking velocimetry (PTV) and its application to analyse free surface flows in laboratory scale models. *Periodica Polytechnica Civil Engineering*, 52(2):63, 2008. doi: 10.3311/pp.ci.2008-2.02.

[42] K. Ogawara, S. Iida, T. Noborizato, and K. Asuwa. A numerical study on kalman filtering PTV for variable density flows. In *American Institute of Aeronautics and Astronautics*. American Institute of Aeronautics and Astronautics, 1996. doi: 10.2514/6.1996-36.

[43] David Forsyth. *Computer vision : a modern approach*. Pearson, Boston, 2012. ISBN 9780136085928.

[44] S. Palmer and M. J. Hargather. Three-dimensional particle tracking velocimetry and size estimation using stereo schlieren systems. *APS Division of Fluid Dynamics Meeting Abstracts (pp. P20-004).*, 2021.

[45] Martin Kainrath. 3d high speed filming. *Bachelor Thesis. University of Applied Sciences Upper Austria.*, 2018.

[46] Michael Hargather and Gary Settles. Recent developments in schlieren and shadowgraphy. In *27th AIAA Aerodynamic Measurement Technology and Ground Testing Conference*. American Institute of Aeronautics and Astronautics, 2010. doi: 10.2514/6.2010-4206.

[47] M.M. Biss, G.S. Settles, M.J. Hargather, L.J. Dodson, and J.D. Miller. High-speed digital shadowgraphy of shock waves from explosions and gunshots. In *Shock Waves*, pages 91–96. Springer Berlin Heidelberg, 2009. doi: 10.1007/978-3-540-85168-4_13.

[48] Michael J. Hargather and Gary S. Settles. Retroreflective shadowgraph technique for large-scale flow visualization. *Applied Optics*, 48(22):4449, jul 2009. doi: 10.1364/ao.48.004449.

[49] Markus Raffel. Background-oriented schlieren (BOS) techniques. *Experiments in Fluids*, 56(3), mar 2015. doi: 10.1007/s00348-015-1927-5.

[50] Samuel J. Grauer, Andreas Unterberger, Andreas Rittler, Kyle J. Daun, Andreas M. Kempf, and Khadijeh Mohri. Instantaneous 3d flame imaging by background-oriented schlieren tomography. *Combustion and Flame*, 196:284–299, oct 2018. doi: 10.1016/j.combustflame.2018.06.022.

[51] F. Nicolas, V. Todoroff, A. Plyer, G. Le Besnerais, D. Donjat, F. Micheli, F. Champagnat, P. Cornic, and Y. Le Sant. A direct approach for instantaneous 3d density field reconstruction from background-oriented schlieren (BOS) measurements. *Experiments in Fluids*, 57(1), dec 2015. doi: 10.1007/s00348-015-2100-x.

APPENDIX A

MATLAB CODE

A.1 Automated Fragment Detection

```
%% Sean Palmer, New Mexico Tech, SGDL 2022
% AutoFragDetection.m
% A code for detecting and extracting fragment positions to ...
    input into
% Kalman filter-based tracking codes.


%% clear everything
clc
clear all
close all

%% load preset variables from Fragment Detection Preview Codes
% filePath = 'D:\Sean\Documents\20210903_THOR_Testing\shot 4\';
% cd(filePath)
% load('savedVariablesShot4_2000to50v2.mat');

%% load camera images
currentFolder = pwd;
% pathname = 'D:\Sean\Documents\20210903_THOR_Testing\shot ...
    4\Test1\20210903_THOR_Shot4_Phantom v711';
pathname = strcat(currentFolder,...
    '\input files\stereo sample files\test 11 25736\actual');
cd(pathname);
Cam = dir('*tiff'); % list of all tif images

%% initialize or override variables to begin automated detection ...
    of frag
startFrame = 45; % user defined input of when to start process
finalFrame = 100; % user defined input of when to end process
Nfiles = finalFrame - startFrame + 1; % number of files
maximumSize = 200; % the maximum pixel area to observe
minimumSize = 5; % the minimum pixel area to observe
X = cell(1,Nfiles); %detection X coordinate indices
Y = cell(1,Nfiles);  %detection Y coordinate indices
```

```
xoffset = 0; % controls the window of where to accept detections ...
    of frag

%% detect fragments in images
for i = startFrame: finalFrame
    %% read in images
    I1 = imread(Cam(i).name); % first image in processing sequence
%     background = imread(Cam(1).name); % background image
%     subtracted = (background - I1); % perform image ...
    subtraction if desired
%
    %% if desired, can use Otsu's method (threshold based image ...
        segment.)
%     level = graythresh(I1); %
%     bw = ¬imbinarize(I1, level);
%     figure, imshow(bw,[])
    %% image segmentation/binarization
    bw = segmentImage2(I1); % perform image segmentation
    labeled = bwlabel(bw); % labels regions


    % preview images if desired
%     figure, imshow(bw, [])
%     title('cam 1 bw')
%     figure, imshow(labeled, [])
%     title('cam 1 label')

    %% look at all detected regions and extract properties
    % all detected regions, or particles
    properties = regionprops(labeled, 'all');
    fragmentAreas = [properties.Area]; % save pixel areas of regions
    desiredSizes = (fragmentAreas ≤ maximumSize) & ...
        (fragmentAreas ≥ minimumSize); % desired sizes of fragments
    desired_indices = find(desiredSizes); % labels of desired ...
        fragments

    %% isolate desired detected regions
    if ¬isempty(desired_indices)
        % isolate desired labels on labeled image
        desiredFragmentImage = ismember(labeled, desired_indices);
        % Label each regions for future measurements
        labeledDesiredImage = bwlabel(desiredFragmentImage);
        % figure, imshow(labeledDesiredImage)
        % title('labeledDesiredImage')

        %% obtain properties of isolated regions
        properties2 = regionprops(labeledDesiredImage, 'all');

        % sort set number of particles by area, largest to ...
            smallest if
```

```
        % desired
        get_particles = [properties2.Area];
        % [get_particles, j] = sort(get_particles, 'descend');
        % largeToSmall = sort(properties2, 'descend');

        % save centroids and other properties in variables
        centroids2 = [properties2.Centroid];
        equivDiam2 = [properties2.EquivDiameter];
        boundBox = [properties2.BoundingBox];
        numberOfBoxes = length(equivDiam2);
        boxes = reshape(boundBox,[],numberOfBoxes);
        x_cent = centroids2(1:2:end-1);
        y_cent = centroids2(2:2:end);
        imshow(I1,[])
        if i == startFrame
            h = drawcircle('Color','g','FaceAlpha',0.0);
            x0 = h.Center(1);
            y0 = h.Center(2);
            r = h.Radius;
        else
            h = ...
                drawcircle('Center',[x0,y0],'Radius',r,'Color','g', ...
                'FaceAlpha',0.0);
        end
        hold on

        for n = 1:length(x_cent)
            if (x_cent(n)-x0)^2 + (y_cent(n)-y0)^2 ≤ r^2
                xCent(n) = x_cent(n);
                yCent(n) = y_cent(n);
            end
        end

        %save centroids into cells
        x_centroid{i - startFrame + 1} = xCent;
        y_centroid{i - startFrame + 1} = yCent;
        equivalentDiameter{i - startFrame + 1} = equivDiam2;



        % plot centroids
        for k = 1 : length(xCent)
%              if x_cent(k) > xoffset % show only centroids of ...
    past offset
                plot(xCent(k), yCent(k), 'r+');
%              end
        end
        drawnow();
    end
end
```

```
%% save detections into a directory of user's choice
detectionsDirectory = strcat(currentFolder,...
    '\saved files');
mkdir(detectionsDirectory)
cd(detectionsDirectory)
save('detections_stereo_test11_size5to100.mat', 'x_centroid', ...
    'y_centroid')



%% image segmentation functions

% image segmentation of color images: flood fill-based
function [BW,maskedImage] = segmentImage(RGB)
%segmentImage Segment image using auto-generated code from ...
    imageSegmenter app
%  [BW,MASKEDIMAGE] = segmentImage(RGB) segments image RGB using
%  auto-generated code from the imageSegmenter app. The final ...
    segmentation
%  is returned in BW, and a masked image is returned in MASKEDIMAGE.

% Auto-generated by imageSegmenter app on 21-Mar-2022
%----------------------------------------------------


% Convert RGB image into L*a*b* color space.
X = RGB;

% Auto clustering
s = rng;
rng('default');
L = imsegkmeans(single(X),2,'NumAttempts',2);
rng(s);
BW = L == 2;

% Flood fill
row = 170;
column = 542;
tolerance = 5.000000e-03;
normX = sum((X - X(row,column,:)).^2,3);
normX = mat2gray(normX);
addedRegion = grayconnected(normX, row, column, tolerance);
BW = BW | addedRegion;

% Create masked image.
maskedImage = RGB;
maskedImage(repmat(¬BW,[1 1 3])) = 0;
end

```

```
% image segmentation of grayscale images: adaptive thresholding
function [BW,maskedImage] = segmentImage2(X)
%segmentImage Segment image using auto-generated code from ...
    imageSegmenter app
%   [BW,MASKEDIMAGE] = segmentImage(X) segments image X using ...
    auto-generated
%   code from the imageSegmenter app. The final segmentation is ...
    returned in
%   BW, and a masked image is returned in MASKEDIMAGE.

% Auto-generated by imageSegmenter app on 26-Mar-2022
%----------------------------------------------------


% Adjust data to span data range.
X = imadjust(X);

% Threshold image - adaptive threshold
BW = imbinarize(X, 'adaptive', 'Sensitivity', 1.0, ...
    'ForegroundPolarity', 'bright');

% Invert mask
BW = imcomplement(BW);

% Create masked image.
maskedImage = X;
maskedImage(¬BW) = 0;
end
```

A.2 Multi Fragment Kalman Filter (constant velocity)

```
%% Sean Palmer, New Mexico Tech, SGDL 2022
% KalmanTracking.m
% A code for detecting and extracting fragment positions to ...
    input into
% Kalman filter-based tracking codes.
%% clear everything
clc
clear all
close all
%%
%                        References
% Buehren, Markus (2020). Functions for the rectangular assignment
% problem version 1.5.0.0. assignmentoptimal source code
% ...
    https://www.mathworks.com/matlabcentral/fileexchange/6543-functions-for-the-
```

```
% MATLAB Central File Exchange. Retrieved October 28, 2020.

%% user defined initial conditions
% startFrame = 40; % THOR
% finalFrame = 60; % THOR

% IndianHead data
startFrame = 45; % user defined input of when to start process
finalFrame = 100; % user defined input of when to end process
range1 = finalFrame - startFrame + 1; %restrict number of frames ...
    of interest
num_particles = 100; %number of particles
updateInitialize = range1; %startFrame - 37 + 1;

t = 1; % time increment in frames
%THOR: 1
max_dist = 4; % THOR: 25
min_dist = 0; % THOR: 0
initial_vel = 3;
std_dev_x = 1; % error in x position measurement 100
std_dev_y = 1; % error in y position measurement 100
accel = 0; % initialize acceleration in pixels/frame^2
std_dev_accel = 1; % error in acceleration 100

poi = 1; %particle of interest

enablePreview = 1; % Y = 1, N = 2
%% load detection mat file and image files
currentFolder = pwd;
filePath = strcat(currentFolder,...
    '\saved files');
cd(filePath);
load('detections_stereo_test11_size5to100.mat')

pathname = strcat(currentFolder,...
    '\input files\stereo sample files\test 11 25736\actual');
cd(pathname);
images = dir('*tiff'); % list of all tif images

%% add the contents of subfolders with necessary files to run
filePath = strcat(currentFolder,...
    '\Important Non-SGDL Codes');
addpath filePath

%% initialize the coefficent matrices
A = [1 t; 0 1];
B = [t^2/2; t];
C = [1 0];
%% assumed initial conditions (user defined)
% initialize x and y positions and measurements
```

```
num_meas = length(x_centroid{updateInitialize}); %initize number ...
    of measurements
state_xpos = x_centroid{updateInitialize}; % first x estimate = ...
    x position measurement
state_ypos = y_centroid{updateInitialize}; % first y estimate = ...
    y position measurement
state_xvel = zeros(1, num_meas);
state_yvel = zeros(1, num_meas);



for n = 1:num_meas
     state_xvel(n) = initial_vel;
     state_yvel(n) = 0;
end

state_x = [state_xpos; state_xvel]; % estimate of x direction states
state_y = [state_ypos; state_yvel]; % estimate of y direction states
size(state_xpos)
size(state_xvel)

preset = 9000;  %pick large enough size to prevent tracks from ...
    starting at same origin
est_posX = nan(preset);
est_posY = nan(preset);
est_velX = nan(preset);
est_velY = nan(preset);


% initialize error in estimates of x and y positions
var_a = std_dev_accel ^ 2; % variance of acceleration
dep1 = (t^2/2)^2; % dependencies in time
dep2 = t*(t^2/2);
dep3 = t^2;
E_predX = [dep1 dep2; dep2 dep3] * var_a;
E_predY = [dep1 dep2; dep2 dep3] * var_a;

%intialize error measurement covariance matrices
E_measX = std_dev_x ^ 2;  % error in x position measurement
E_measY = std_dev_y ^ 2;  % error in y position measurement
strk_trks = zeros(1,2000);  %counter of how many strikes a track ...
    has gotten

%% Kalman filter-based tracking
disp('tracking...')
tic
for image = 1:range1
     %Update state estimation prediction (Kalman step 1)
     %testA = (A .* state_x(1,:))
     %testB = (accel * B)
```

```
    for n = 1:length(state_x)
        state_x(:, n) = (A * state_x(:,n)) + (accel * B);
        state_y(:, n) = (A * state_y(:,n)) + (accel * B);
    end

    % generate the cost matrix
    measured_position = [x_centroid{image}; y_centroid{image}]';
    estimated_position = [state_x(1,1:num_meas); ...
       state_y(1,1:num_meas)]';
    pair_wise = [estimated_position; measured_position];
    vec_sum = sum(power(pair_wise.', 2),1);
    term1 = vec_sum.' + vec_sum;
    term2 = 2 * pair_wise * pair_wise.';
    eulerian_distance = sqrt(term1 - term2);
    cost_matrix = eulerian_distance(1:num_meas,num_meas+1:end);

    %predict next covariance (step 2)
    E_predX = A * E_predX * A' + E_predX;
    E_predY = A * E_predY * A' + E_predY;

    % Calculate the Kalman Gain factor (step 3)
    Kx = E_predX * C' * inv(C * E_predX * C' + E_measX);
    Ky = E_predY * C' * inv(C * E_predY * C' + E_measY);

    % Buehren, Markus (2020). Functions for the rectangular ...
       assignment
    % problem version 1.5.0.0.
    % ...
       https://www.mathworks.com/matlabcentral/fileexchange/6543-functions-for-t
    % MATLAB Central File Exchange. Retrieved October 28, 2020.
    munkres = assignmentoptimal(cost_matrix);

    % check whether assigment is reasonable given user defined ...
       distance
    check = zeros(1,num_meas);
    cost = zeros(1, num_meas);
    for n = 1:num_meas
        selection = munkres(n);
        if selection ≠ 0
            cost(n) = cost_matrix(n, selection);
            check(n) = cost(n)  < max_dist && cost(n) > min_dist;
        end
    end

    %apply the assignment to the step of updating the state ...
       estimates
    munkres = times(munkres', check);
    for n = 1:length(munkres)
        if munkres(n) > 0 && munkres(n) ≤ num_particles
            selection = munkres(n);
```

```
            % Kalman step 4
            Z_x = x_centroid{image}(selection);
            Z_y = y_centroid{image}(selection);
            updateX = (Z_x - C * state_x(:,n));
            updateY = (Z_y - C * state_y(:,n));

            test = updateX * Kx;
            state_x(:, n) =  state_x(:, n) + updateX * Kx;
            state_y(:, n) =  state_y(:, n) + updateY * Ky;
            updateX = (Z_x - C * state_x(:,n));
            updateY = (Z_y - C * state_y(:,n));
        end
    end

    %save the estimated positions and velocities
    est_posX(image,1:num_meas) = state_x(1,1:num_meas);
    est_posY(image,1:num_meas) = state_y(1,1:num_meas);
    est_velX(image,1:num_meas) = state_x(2,1:num_meas);
    est_velY(image,1:num_meas) = state_y(2,1:num_meas);

    % kalman step 5: update error covariance estimation for x ...
        and y
    I = [1 0; 0 1]; %identity matrix
    E_predX = (I - Kx * C) * E_predX; %updated predicted error ...
        covariance for x direction
    E_predY = (I - Ky * C) * E_predY; %updated predicted error ...
        covariance for y direction

    % update new particle paths
    label_particles = 1:size(measured_position, 1); %label ...
        existing particles
    non_members = ¬ismember(label_particles,munkres); %check for ...
        nomembers
    non_members_meas = measured_position(non_members,:)'; %find ...
        non-existing members

    if ¬isempty(non_members_meas) % if non-existing members ...
        exist, add them
        L = length(non_members_meas);
        state_x(1, num_meas + 1: num_meas + L) = ...
            non_members_meas(1,:);
        state_y(1, num_meas + 1: num_meas + L) = ...
            non_members_meas(2,:);
        num_meas = num_meas + L;
    end
end
toc
disp('tracking ended...')
%% save to mat file
%store kalman filtered estimates into a .mat file
```

```matlab
disp('saving data...')
tic
filePath = strcat(currentFolder,...
    '\saved files');
cd(filePath);
save('kalman_test11_v1.mat',  'images', 'est_posX', 'est_posY', ...
    'x_centroid', 'y_centroid', 'num_meas', 'range1','est_velX', ...
    'est_velY')
toc
disp('data saved!')

%% preview particle paths
if enablePreview == 1
    disp('plotting preview of tracks...')
    tic
    plot_particle_paths(images, est_posX, est_posY, x_centroid, ...
        y_centroid, num_meas, range1 , currentFolder, startFrame);
    toc
end
%% functions
%plot path of particles: a preview
function plot_particle_paths(files, pos1, pos2, meas1, meas2, ...
    num, time_range, currentFolder, startFrame)
    pathname = strcat(currentFolder,...
    '\input files\stereo sample files\test 11 25736\actual');
    cd(pathname);
    files = dir('*tiff'); % list of all tif images

    %file_path = ['C:\Users\Sean\Desktop\RM master\RM Data\RM ...
        Open Material W-AL ...
        THOR\Test_1_cyl_W87.5_Al12.5_Process\OG Images\'];
    %pic_name_prefix = 'Test_1_cyl_W87.5_Al12.5_RP2';
    poi = 1;
    %save video of frames
    video = VideoWriter('test11_25736.avi');
    video.FrameRate = 1;
    open(video);

    for image = 1: time_range

        input = imread(files(image + startFrame - 1).name);
        imshow(input, []);
        hold on;
        for trail = 1:num
            trail_pos_x = pos1(1:image,trail);
            trail_pos_y = pos2(1:image,trail);
            if trail == poi
                %plot(trail_pos_x, ...
                    trail_pos_y,'-','color','b','linewidth',1) ...
                    %show estimated path
```

```
            else
                plot(trail_pos_x, ...
                    trail_pos_y,'-','color','r','linewidth',1) ...
                    %show estimated path
            end
        end
        drawnow;
        hold off
    end

end
```

A.3 Display Tracking

```
%% Sean Palmer, New Mexico Tech, SGDL 2022
% DisplayFilteredTracks.m
% A code for isolating and displaying non-erroneous filtered tracks
% determined from the 2D Kalman filter-based tracking code.
%% clear everything
clc
close all
clear all
%% user defined variables
startFrame = 45; % user defined input of when to start process
finalFrame = 100; % user defined input of when to end process
updateInitialize = 1; % offset from startFrame if needed
Nfiles = finalFrame - startFrame + 1; % number of files
final = Nfiles;
enableColor = 2; % Y = 1, N = 2
%% load in saved tracks from user defined directories
currentFolder = pwd;
filePath = strcat(currentFolder,...
    '\saved files');
cd(filePath);
load('kalman_test11_v1.mat')
%% read in images from user defined directories
pathname = strcat(currentFolder,...
    '\input files\stereo sample files\test 11 25736\actual');
cd(pathname);
files = dir('*tiff'); % list of all tif images
%files =  dir('*tif');
input = (imread(files(1).name));


%% assumed size of circles to encapsulate tracked fragments
fragment_area = 5;
```

```
equiv_diam = sqrt(4*fragment_area/pi); %m
[row, col] = size(input);
[xgrid, ygrid] = meshgrid(1:size(input ,2), 1:size(input ,1));
th = 0:pi/50:2*pi;
r = 2 * equiv_diam;


%% rewrite variables
files = images;
pos1 = est_posX;
pos2 = est_posY;
meas1 = x_centroid;
meas2 = y_centroid;
num = num_meas;
time_range = range1;
velx = est_velX;
vely = est_velY;


%% setup font, label, color map properties
labelShiftX =  0;
textFontSize = 7;
c = parula;
colormap(c);
A = colormap(c);


%% filter erroneous vectors
for image = 1 + updateInitialize:final
    % filter the trails and get the desired velocities
    filtered_trails = [];
    desired_velocities = [];
    n = 1;
    for trail = 1:num
        trail_pos_x = pos1(1:image,trail);
        trail_pos_y = pos2(1:image,trail);

        if image > 1 && ¬isnan(vely(image, trail)) && ¬...
            isnan(vely(image-1, trail))
             ∆_vely = vely(image, trail) - vely(image-1, trail);
        end
        vel2D = sqrt(velx(image, trail) ^ 2 + vely(image, trail) ...
            ^ 2);
        % this line below filters bad tracks
        if  velx(image, trail) > 0 && abs(vely(image, trail)) < ...
            4 && abs(velx(image, trail)) ≠ 5 && velx(image, ...
            trail) < 3%
             increment = (pos1(1,trail)) - (pos1(image,trail));

```

```
            if (image == final) %&  (abs(increment) < 2 || ...
                isnan(increment)) % && (pos1(image,trail) > 160) ...
                && pos2(image, trail) > 100%
                filtered_trails(n) = trail;
                n = n + 1;
            end
        end
    end

    for k = 1:n-1
        desired_velocities(k) = sqrt(abs(velx(range1, ...
            filtered_trails(k)))^2 + abs(vely(range1, ...
            filtered_trails(k))) ^ 2);
    end
end
%% filter some tracks by
% filtered_trails(17:19) = [];
% d1 = [36,38,10,41,11];
% filtered_trails(d1) = [];

%% check velocities
% find and sort non-zero velocities
des_vel_index = find(desired_velocities);
des_vel = desired_velocities(des_vel_index);
sort_vel = sort(des_vel);

%% define color range
minVel = sort_vel(1);
maxVel = sort_vel(end);
% endOffset = 1;
% while isnan(maxVel)
%     maxVel = sort_vel(end - endOffset);
%     endOffset = endOffset + 1;
% end
%% apply calibration and determine min and max values for color var
% the following is an example of a THOR test calibration
diameterSphere = 58.32/1000; % mm to m
calibration = diameterSphere/178; % mm/px

maxValue = maxVel * (calibration)/ 2.5e-5; % m/px); %px;
minValue = minVel * (calibration)/ 2.5e-5;

a = minValue:((maxValue - minValue)/255):maxValue;
a_px = minVel:((maxVel - minVel)/255):maxVel;


%% generate colors for tracks and color bars
RGB = [];
for trail = 1:length(des_vel)
    for j = 1:length(a_px)
```

```
            absolute(j,trail) = abs(a_px(j) - abs(des_vel(trail)));
    end
    [val, idx] = min(absolute(:, trail));
    if idx > 256
        idx = 256;
    end
    RGB(trail, :, :, :) = A(idx, :);
end

fig = figure;
ax1 = axes(fig);
%% display tracking of particles with colors
for image = 1:final

    [input, map] = imread(files(image + startFrame-1).name);
    input = mat2gray(input);
    imshow(input, []);
    n = 1;
    hold on;
    for trail = 1:length(des_vel)
        trail_pos_x = pos1(1:image,filtered_trails(trail));
        trail_pos_y = pos2(1:image,filtered_trails(trail));
        tail_x = 1.0 * pos1(image,filtered_trails(trail));
        tail_y = 1.0 * pos2(image,filtered_trails(trail));
        xunit = r * cos(th) + tail_x;
        yunit = r * sin(th) + tail_y;
        if enableColor == 1
            plot(trail_pos_x, ...
               trail_pos_y,'-','color',RGB(trail,:,:,:),'linewidth',2) ...
               %show estimated path
%              plot(xunit, yunit, RGB(trail,:,:,:));
        else
            plot(trail_pos_x, ...
               trail_pos_y,'-','color','g','linewidth',2) %show ...
               estimated path
%              plot(xunit, yunit, 'g');
        end

%          text(trail_pos_x, trail_pos_y, num2str(trail), ...
    'FontSize', textFontSize, 'FontWeight', 'Bold', 'color', ...
    'white');
        value = (pos1(1,trail)) - (pos1(image,trail));

        if (image == final) % && (pos1(image,trail) < 370) && ...
            (abs(value) > 0 || isnan(value))  && (trail < 500)
            %text(pos1(image:image,trail) + labelShiftX, ...
               pos2(image:image,trail), num2str(trail), ...
               'FontSize', textFontSize, 'FontWeight', 'Bold', ...
               'color', 'r');; %show measured centroid
            filtered_trails(n) = trail;
```

```
            trail_pos_xref = pos1(1:image,filtered_trails(trail));
            trail_pos_yref = pos2(1:image,filtered_trails(trail));
            n = n + 1;
        end

    end
    drawnow;
    pause(0.5)

    %write frames into directory
    str_i = num2str(image,'%03.f');
    frame = strcat('frame_burning_centerCam2',str_i, '.tif');
    iptsetpref('ImshowBorder','tight');
    imwrite(getframe(gcf).cdata, frame,  'Compression', 'none')

end

%% setup color bar
if enableColor == 1
    ax2=axes(fig,...
    'Position',[ax1.Position(1)+ax1.Position(3) + ...
       .5,ax1.Position(2),0.0,0.4]);
    axis off
    set(ax2,'color','none');

    % hcb = colorbar(ax2,'Position',...
    % [ax1.Position(1)+ax1.Position(3)+0.025,0.15,0.05,0.7],...
    %     'AxisLocation','in');
    ax2=axes(fig,...
    'Position',[ax1.Position(1)+ax1.Position(3) + ...
       .5,ax1.Position(2),0.0,0.4]);
    axis off
    set(ax2,'color','none');

    cmap = parula;
    cmap1 = colormap(cmap);
    hcb = colorbar(ax2, 'Position', [.9 0.1 0.03 .8]);

    ax2.CLim = [minValue,maxValue];
    colorTitleHandle = get(hcb,'Title');
    titleString = '\color{white} velocity (m/s)';
    set(colorTitleHandle ,'String',titleString)
    colormap(ax1, 'gray');

    % hcb.Layout.Tile = 'east';
    hcb.Visible = 'on';
    hcb.TickLabelInterpreter = 'tex';
    hcb.FontSize = 12;
    hcb.Ticks = linspace(minValue, maxValue, 5);
    tickRange = linspace(minValue, maxValue, 5);
```

```
    white = [1,1,1];
    tickLabelsCheck = hcb.TickLabels;
    % ticksRound = [];
    for ii = 1:numel(hcb.TickLabels)
        ticksRound(ii) = tickRange(ii);
        ticksRound(ii) = round(ticksRound(ii),3,'significant');
        hcb.TickLabels{ii} = [sprintf('\\color[rgb]{%f,%f,%f} ', ...
            white), num2str(ticksRound(ii))];
    end

%% save/write out color bar
str_i = num2str(image,'%03.f');
    frame = strcat('frame_burning_centerCam2',str_i, '.tif');
iptsetpref('ImshowBorder','tight');
imwrite(getframe(gcf).cdata, frame,  'Compression', 'none')
end

%% save files
save('filtered_IHshot11_25736V2.mat','des_vel', ...
    'filtered_trails', 'images', 'est_posX', 'est_posY', ...
    'x_centroid', 'y_centroid', 'num_meas', 'range1','est_velX', ...
    'est_velY','RGB')
```

A.4 Fragment Matching

```
% FINALIZED CODE


% Fragment Matching with Playback
% Check by plotting the epipolar lines in one camera view by ...
    multiplying
% the fundamental matrix by a position in another camera view. ...
    Playback the
% tracks from Kalman Filter. Uses the principle that only one ...
    fragment
% 'centroid' in theory should interesect with both epipolar ...
    lines (one
% early and one late)

clc
close all
clear all

%% Constants
fragment_area = 10;
equiv_diam = sqrt(4*fragment_area/pi); %m
```

```
th = 0:pi/50:2*pi;
r = 2 * equiv_diam;
%% Desired time frame
% start_frame = 143;
% final_frame = 164;
start_frame = 161;
final_frame = 181; %169


start_frame = 159;
final_frame = 179; %169

Nfiles = final_frame - start_frame + 2; % number of files
final = Nfiles;

start_frame2 = start_frame + 1;
final_frame2 = start_frame + 1;

initial = 159;

%% read in images and variables
pathname1 = 'D:\Sean\Documents\IndianHeadAnalysis\Test ...
    10\25736_test10\Test1\test10_25736\Cam1';
pathname1 = 'C:\Users\Sean\Desktop\Work In ...
    Progress\IndianHead\25736_RGB_test8_RGB_v2';
pathname1 = 'C:\Users\Sean\Desktop\Work In ...
    Progress\IndianHead\test14\2\Test1\test14_25736_RGB';
cd(pathname1);
Cam1 = dir('*tif'); % list of all tif images

for j = 1:length(Cam1)
    fileNames1{j} = strcat(Cam1(j).folder,'\', Cam1(j).name);
end
% testImageFileName = imageFileNames1{1};
% imshow(testImageFileName);
% gca

pathname2 = 'D:\Sean\Documents\IndianHeadAnalysis\Test ...
    10\22905_test10\Test1\test10_22905\Cam2';
pathname2 = 'C:\Users\Sean\Desktop\Work In ...
    Progress\IndianHead\22905_RGB_test8_RGB_v2';
pathname2 = 'C:\Users\Sean\Desktop\Work In ...
    Progress\IndianHead\test14\Test1\test14_22905_RGB';
cd(pathname2);
Cam2 = dir('*tif'); % list of all tif images

for j = 1:length(Cam2)
    fileNames2{j} = strcat(Cam2(j).folder,'\', Cam2(j).name);
end
%% Get tracks (Kalman Filtered Positions)
```

```
% filePath = ...
    'C:\Users\Sean\Desktop\IndianHead\22905_RGB_test8_RGB_v2';
% cd(filePath)


% first camera
filePath = 'D:\Sean\Documents\IndianHeadAnalysis\Test ...
    10\25736_test10\Test1\test10_25736\Cam1';
filePath = 'C:\Users\Sean\Desktop\Work In ...
    Progress\IndianHead\25736_RGB_test8_RGB_v2';
filePath = 'C:\Users\Sean\Desktop\Work In ...
    Progress\IndianHead\test14\Test1\test14_25736_RGB\trackedCam2Frames';
cd(filePath)
%Structure = load('filtered_test10_25736.mat', 'est_posX', ...
    'est_posY', 'filtered_trailsV2', 'des_vel','RGB2');
%Structure = ...
    load('valid_pathsV3_test8_cam1_burning_center_cam2_backup.mat', ...
    'est_posX', 'est_posY', 'filtered_trailsV2', 'des_vel','RGB2');
Structure = load('valid_pathsV3_test14_cam2.mat', 'est_posX', ...
    'est_posY', 'filtered_trailsV2');

est_posX_frag = Structure.est_posX;
est_posY_frag = Structure.est_posY;
trails = Structure.filtered_trailsV2;
% Svel = Structure.des_vel;
% rgb = Structure.RGB2;
pos1 = est_posX_frag;
pos2 = est_posY_frag;

%second camera
filePath ='D:\Sean\Documents\IndianHeadAnalysis\Test ...
    10\22905_test10\Test1\test10_22905\Cam2';
filePath = 'C:\Users\Sean\Desktop\Work In ...
    Progress\IndianHead\22905_RGB_test8_RGB_v2';
filePath = 'C:\Users\Sean\Desktop\Work In ...
    Progress\IndianHead\test14\Test1\test14_22905_RGB\trackedCam2Frames';
filePath = 'C:\Users\Sean\Desktop\Work In ...
    Progress\IndianHead\test14\Test1\test14_22905_RGB'
cd(filePath)
%Structure2 = load('filtered_test10_22905.mat', 'est_posX', ...
    'est_posY', 'filtered_trailsV2');
%Structure2 = ...
    load('valid_pathsV3_test8_cam1_burning_center_cam1_v12.mat', ...
    'est_posX', 'est_posY', 'filtered_trailsV2', 'des_vel');
Structure2 = load('valid_pathsV3_test14_cam1.mat', 'est_posX', ...
    'est_posY', 'filtered_trailsV2', 'des_vel');

est_posX_frag2 = Structure2.est_posX;
est_posY_frag2 = Structure2.est_posY;
```

```
trails2 = Structure2.filtered_trailsV2;
% vel2 = Structure2.des_vel;
% rgb2 = Structure2.RGB2;
%% load stereo parameters

% save_path = 'D:\Sean\Documents\IndianHeadAnalysis\Test 10\';
% cd(save_path)
% load('params_cal2.mat')
pathname = 'C:\Users\Sean\Desktop\Work In ...
    Progress\IndianHead\25736 Cal1';
cd(pathname)

saveFolder = 'D:\Sean\Documents\IndianHeadAnalysis';
cd(saveFolder)

load('params_cal4.mat')

% Get fundamental matrix
F = stereoParams.FundamentalMatrix;

%%


%% User Control
user_cont = input('Start tracking fragment? 1=Y, 2=N: ');
if user_cont == 1
    while user_cont == 1

        % TODO: plot images
        %       plot tracks of interest (1 in 1 camera, all in ...
            another?)
        %       Add number associated with track (check video code)
        %       Enter specific values of image number?
        % Get early and late positions for each track

        %% Plot all fragment numbers
        textFontSize = 8;
        color_label = 1;
        cd(pathname1);
        figure
        movegui('center');
        for image = 1:final
            mainImage = imread(Cam1(image + initial - 1).name);
            mainImage = mat2gray(mainImage);

            imshow(mainImage);
            hold on
            n = 1;
            for trail = 1:length(trails)
                trail_pos_x = pos1(1:image,trails(trail));
```

```
            trail_pos_y = pos2(1:image,trails(trail));
            plot(trail_pos_x, ...
                trail_pos_y,'-','color','g','linewidth',1); ...
                %show estimated path
            tail_x = 1.0 * pos1(image,trails(trail));
            tail_y = 1.0 * pos2(image,trails(trail));
            xunit = r * cos(th) + tail_x;
            yunit = r * sin(th) + tail_y;
            plot(xunit, yunit, 'g');
            tailString = num2str(trail);
%               plot(trail_pos_x, ...
    trail_pos_y,'-','color','r','linewidth',1) %show ...
    estimated path
            %plot(pos1(image,filtered_trailsV2(trail)), ...
                pos2(image,filtered_trailsV2(trail)), 'or', ...
                'MarkerSize', eq_diameter{image + 46}(trail))

            if (image == final) % && (pos1(image,trail) < ...
                370) && (abs(value) > 0 || isnan(value))  && ...
                (trail < 500)
                text(tail_x, tail_y, tailString, 'FontSize', ...
                    textFontSize, 'FontWeight', 'Bold', ...
                    'Color', 'white'); %show measured centroid
            end

        end
        drawnow;
    end

    %% Preview
    disp('Camera A Preview')
    fprintf(1,'Track #        First Image Index        Last ...
        Image Index\n');
    for trail = 1:length(trails)
        savedTracks = pos1(:,trails(trail));
        A = savedTracks;
        B = ¬isnan(savedTracks);

        % indices
        IndicesLast = arrayfun(@(x) find(B(:, x), 1, ...
            'last'), 1:size(A, 2));
        IndicesFirst = arrayfun(@(x) find(B(:, x), 1, ...
            'first'), 1:size(A, 2));
        fprintf(1,'#%2d %17.1f %17.1f \n',trail, ...
            IndicesFirst, IndicesLast);
    end

    disp('Camera B Preview')
    fprintf(1,'Track #        First Image Index        Last ...
        Image Index\n');
```

```
        for trail = 1:length(trails)
            savedTracks = pos1(:,trails(trail));
            A = savedTracks;
            B = ¬isnan(savedTracks);

            % indices
            IndicesLast = arrayfun(@(x) find(B(:, x), 1, ...
               'last'), 1:size(A, 2));
            IndicesFirst = arrayfun(@(x) find(B(:, x), 1, ...
               'first'), 1:size(A, 2));
            fprintf(1,'#%2d %20.1f %25.1f \n',trail, ...
               IndicesFirst, IndicesLast);
        end


        %% Select and show desired fragment at early frame (SHOW ...
           PREVIEW)
        knownFragment = input('Known fragment number? 1=Y, 2=N: ');
        if knownFragment == 1
            desiredFragment = input('Select fragment number: ');
            trail = desiredFragment;
        end

        start_frame = input('Select start frame: ');



        %% Plot an image of camera view with query point
        figure
        movegui('northwest');
        firstImageCameraA = fileNames1{initial + start_frame};
        imshow(firstImageCameraA);
        hold on
        if knownFragment == 1
            plot(pos1(start_frame,trails(trail)), ...
               pos2(start_frame,trails(trail)),'go')
            text(pos1(start_frame,trails(trail)), ...
               pos2(start_frame,trails(trail)), num2str(trail), ...
               'FontSize', textFontSize, 'Color', 'white');
            queryPoints = [pos1(start_frame,trails(trail)); ...
               pos2(start_frame,trails(trail))];
        else
            [getQueryX,getQueryY] = ginput;
            plot(getQueryX,getQueryY,'go');
            queryPoints = [getQueryX;getQueryY];
        end

        XL = xlim(gca); % define x axis limits
        X = linspace(0,XL(2)); % define x axis values
        hold off
```

```

        %% plot epipolar lines on other camera view
        cd(pathname2);
        figure
        movegui('northeast');
        start_frame2 = start_frame + 1;

        firstImageCameraB = fileNames2{initial + start_frame2};
        imshow(firstImageCameraB)
        drawnow;
        hold on


        %% Get epipolar lines in other camera

%         for i = 1:length(queryPoints)
            lines = F' * [queryPoints;1];
            a = lines(1);
            b = lines(2);
            c = lines(3);
            pp = [-a,-c]/b;
            pv = polyval(pp,X);
            plot(X,pv,'Color', 'g')
%         end
        drawnow;

        %% Select and show desired fragment at early frame (SHOW ...
            PREVIEW)
%         knownFragment = input('Known fragment number? 1=Y, ...
    2=N: ');
%         if knownFragment == 1
%             desiredFragment = input('Select fragment number: ');
%             trail = desiredFragment;
%         end

        update_frame = input('Select next frame to check: ');


        %% Plot an image of camera view with query point
        figure
        movegui('southwest');
        firstImageCameraA = fileNames1{initial + start_frame + ...
            update_frame};
        imshow(firstImageCameraA);
        hold on
        if knownFragment == 1
            plot(pos1(start_frame + update_frame,trails(trail)), ...
                pos2(start_frame + update_frame,trails(trail)),'go')
            text(pos1(start_frame + update_frame,trails(trail)), ...
                pos2(start_frame + update_frame,trails(trail)), ...
```

```
                   num2str(trail), 'FontSize', textFontSize, ...
                   'Color', 'white');
            queryPoints = [pos1(start_frame + ...
                   update_frame,trails(trail)); pos2(start_frame + ...
                   update_frame,trails(trail))];
        else
            [getQueryX,getQueryY] = ginput;
            plot(getQueryX,getQueryY,'go');
            queryPoints = [getQueryX;getQueryY];
        end

        XL = xlim(gca); % define x axis limits
        X = linspace(0,XL(2)); % define x axis values
        hold off

        %% plot epipolar lines on other camera view
        cd(pathname2);
        figure
        movegui('southeast');
%         start_frame2 = start_frame + 1;

        firstImageCameraB = fileNames2{initial + start_frame2 + ...
           update_frame};
        imshow(firstImageCameraB)
        drawnow;
        hold on


        %% Get epipolar lines in other camera

%         for i = 1:length(queryPoints)
            lines = F' * [queryPoints;1];
            a = lines(1);
            b = lines(2);
            c = lines(3);
            pp = [-a,-c]/b;
            pv = polyval(pp,X);
            plot(X,pv,'Color', 'g')
%         end
        drawnow;

%
%         %         %% check with back projection
%         %         line = F' * p;
%         % match track number?
%
        user_cont = input('Retry or continue? 1=Y, 2=N: ');
    end
end
```

```
disp('Done!')
```

A.5 3D reconstruction

```
%% Sean Palmer, New Mexico Tech, SGDL 2022
% ReconstructionManual.m
% A code for manually assigning the tracked positions of ...
    fragments from two
% stereo camera views and reconstructing the 3D trajectory.
%% clear everything
clc
close all
clear all
currentFolder = pwd;
currentFolder = 'E:\Fragment Training SGDL';
%% user preset

% add path for mat files (Automatically Tracked)
% addpath 'D:\Sean\Documents\IH Analysis October\Test 11\Detections'
% currentFolder = pwd;
%%
% pathname1 = 'V:\IndianHeadMarch2022\Test 11\saveTracksAreduced';
    pathname1 = strcat(currentFolder,...
    '\input files\sample manual tracking\saveTracksAreduced');

% pathname2 = 'V:\IndianHeadMarch2022\Test 11\saveTracksBreduced';
    pathname2 = strcat(currentFolder,...
    '\input files\sample manual tracking\saveTracksBreduced');

isAutomated = 2; % Y = 1, N = 2
%% load stereo parameters
stereoPath = strcat(currentFolder,...
'\saved files');
cd(stereoPath)
load('stereo_cal3_debayer.mat')
params = stereoParams;
F = stereoParams.FundamentalMatrix;
%% read in auto tracked data
if isAutomated == 1
    fragmentStructure1 = load('filtered_test11_25736.mat', ...
        'est_posX', 'est_posY', 'filtered_trailsV2', ...
        'des_vel','RGB2');
    est_posX_frag1 = fragmentStructure1.est_posX;
    est_posY_frag1 = fragmentStructure1.est_posY;
    trails1 = fragmentStructure1.filtered_trailsV2;
    vel1 = fragmentStructure1.des_vel;
```

```

    fragmentStructure2 = load('filtered_test11_22905.mat', ...
        'est_posX', 'est_posY', 'filtered_trailsV2', ...
        'des_vel','RGB2');
    est_posX_frag2 = fragmentStructure2.est_posX;
    est_posY_frag2 = fragmentStructure2.est_posY;
    trails2 = fragmentStructure2.filtered_trailsV2;
    vel2 = fragmentStructure2.des_vel;
else
    %% read in manual data
    cd(pathname1);
    cd('E:\Fragment Training SGDL\input files\sample manual ...
        tracking\saveTracksAreduced')
    filenames1 = dir('*mat');
    hold on
    for kk = 1:numel(filenames1)
        S1 = load(filenames1(kk).name); % Best to load into an ...
            output variable.
        trail_pos_x = S1.x_centroid;
        trail_pos_y = S1.y_centroid;
        est_posX_frag1{kk} = trail_pos_x;
        est_posY_frag1{kk} = trail_pos_y;
    end
    cd(pathname2);
    cd('E:\Fragment Training SGDL\input files\sample manual ...
        tracking\saveTracksBreduced')
    filenames2 = dir('*mat');
    for kk = 1:numel(filenames2)
        S2 = load(filenames2(kk).name);
        trail_pos_x = S2.x_centroid;
        trail_pos_y = S2.y_centroid;
        est_posX_frag2{kk} = trail_pos_x;
        est_posY_frag2{kk} = trail_pos_y;
    end
end
%% combine into coordinates for each camera (with matched ...
    fragments, manual process)

posXA = est_posX_frag1; % cells of x positions of each fragment
posYA = est_posY_frag1; % cells of y positions of each fragment

posXB = est_posX_frag2; % cells of x positions of each fragment
posYB = est_posY_frag2; % cells of y positions of each fragment

posXA = trimTracks1D(posXA);
posYA = trimTracks1D(posYA);
[IndicesLastA, IndicesFirstA] = indicesExtraction(posXA, ...
    filenames1);

posXB = trimTracks1D(posXB);
```

```
posYB = trimTracks1D(posYB);
[IndicesLastB, IndicesFirstB] = indicesExtraction(posXB, ...
    filenames2);

%% Fragment Matching
[first, last] = matchFrames(1, 1,  IndicesLastA, IndicesFirstA, ...
    IndicesLastB, IndicesFirstB);
camA{1} = [est_posX_frag1{1}(first:last); ...
    est_posY_frag1{1}(first:last)];
camB{1} = [est_posX_frag2{1}(first:last); ...
    est_posY_frag2{1}(first:last)];

for i = 1:length(camA{1}) % first:last
     p1(i,:) = cell2mat(camA{1}(:,i))';
     p2(i,:) = cell2mat(camB{1}(:,i))';
end

for i = 1:length(p1)
     p3D_1{i} = triangulate(p1(i,:) , p2(i,:), stereoParams)/1000;
end

%%
% 2                             A   B
[first, last] = matchFrames(12, 53,  IndicesLastA, ...
    IndicesFirstA, IndicesLastB, IndicesFirstB);
camA{2} = [est_posX_frag1{12}(first:last); ...
    est_posY_frag1{12}(first:last)];
camB{2} = [est_posX_frag2{53}(first:last); ...
    est_posY_frag2{53}(first:last)];

for i = 1:length(camA{2}) % first:last
     p1_2(i,:) = cell2mat(camA{2}(:,i))';
     p2_2(i,:) = cell2mat(camB{2}(:,i))';
end
for i = 1:length(p1_2)
     p3D_2{i} = triangulate(p1_2(i,:) , p2_2(i,:), ...
         stereoParams)/1000;
end
%%
% 3                             A   B
[first, last] = matchFrames(17, 20,  IndicesLastA, ...
    IndicesFirstA, IndicesLastB, IndicesFirstB);
camA{3} = [est_posX_frag1{17}(first:last); ...
    est_posY_frag1{17}(first:last)];
camB{3} = [est_posX_frag2{20}(first:last); ...
    est_posY_frag2{20}(first:last)];

for i = 1:length(camA{3}) % first:last
     p1_3(i,:) = cell2mat(camA{3}(:,i))';
     p2_3(i,:) = cell2mat(camB{3}(:,i))';
```

```
end
for i = 1:length(p1_3)
    p3D_3{i} = triangulate(p1_3(i,:) , p2_3(i,:), ...
        stereoParams)/1000;
end
%%
% 4                            A   B
[first, last] = matchFrames(15, 28,  IndicesLastA, ...
    IndicesFirstA, IndicesLastB, IndicesFirstB);
camA{4} = [est_posX_frag1{15}(first:last); ...
    est_posY_frag1{15}(first:last)];
camB{4} = [est_posX_frag2{28}(first:last); ...
    est_posY_frag2{28}(first:last)];

for i = 1:length(camA{4}) % first:last
    p1_4(i,:) = cell2mat(camA{4}(:,i))';
    p2_4(i,:) = cell2mat(camB{4}(:,i))';
end
for i = 1:length(p1_4)
    p3D_4{i} = triangulate(p1_4(i,:) , p2_4(i,:), ...
        stereoParams)/1000;
end
%%
% 5                            A   B
[first, last] = matchFrames(16, 27,  IndicesLastA, ...
    IndicesFirstA, IndicesLastB, IndicesFirstB);
camA{5} = [est_posX_frag1{16}(first:last); ...
    est_posY_frag1{16}(first:last)];
camB{5} = [est_posX_frag2{27}(first:last); ...
    est_posY_frag2{27}(first:last)];

for i = 1:length(camA{5}) % first:last
    p1_5(i,:) = cell2mat(camA{5}(:,i))';
    p2_5(i,:) = cell2mat(camB{5}(:,i))';
end
for i = 1:length(p1_5)
    p3D_5{i} = triangulate(p1_5(i,:) , p2_5(i,:), ...
        stereoParams)/1000;
end

%%
% 9
fragNumber = 9;
[first, last] = matchFrames(5, 40,  IndicesLastA, IndicesFirstA, ...
    IndicesLastB, IndicesFirstB);
camA{fragNumber} = [est_posX_frag1{5}(first:last); ...
    est_posY_frag1{5}(first:last)];
camB{fragNumber} = [est_posX_frag2{40}(first:last); ...
    est_posY_frag2{40}(first:last)];

```

```
for i = 1:length(camA{fragNumber}) % first:last
    p1_9(i,:) = cell2mat(camA{fragNumber}(:,i))';
    p2_9(i,:) = cell2mat(camB{fragNumber}(:,i))';
end

for i = 1:length(p1_9)
    p3D_9{i} = triangulate(p1_9(i,:) , p2_9(i,:), ...
        stereoParams)/1000;
end
%%
% 10
fragNumber = 10;
[first, last] = matchFrames(27, 27,  IndicesLastA, ...
    IndicesFirstA, IndicesLastB, IndicesFirstB);
camA{fragNumber} = [est_posX_frag1{27}(first:last); ...
    est_posY_frag1{27}(first:last)];
camB{fragNumber} = [est_posX_frag2{27}(first:last); ...
    est_posY_frag2{27}(first:last)];

for i = 1:length(camA{fragNumber}) % first:last
    p1_10(i,:) = cell2mat(camA{fragNumber}(:,i))';
    p2_10(i,:) = cell2mat(camB{fragNumber}(:,i))';
end

for i = 1:length(p1_10)
    p3D_10{i} = triangulate(p1_10(i,:) , p2_10(i,:), ...
        stereoParams)/1000;
end
%%

%% save 3D points
cd('E:\Fragment Training SGDL\saved files')
save('test11_points3D.mat', 'p3D_1', 'p3D_2', 'p3D_3', ...
    'p3D_4','p3D_5', 'p3D_9','p3D_10')
%% rewrite 3D points into new variables specific to spatial ...
    directions
for i = 1:length(camA{1}) % first:last
    point3d_frag1{i} = triangulate(cell2mat(camA{1}(:,i))', ...
        cell2mat(camB{1}(:,i))', params);
    point3d_frag1{i} = point3d_frag1{i} / 1000; % convert from ...
        mm to m
    X1(i) = point3d_frag1{i}(1,1);
    Y1(i) = point3d_frag1{i}(1,2);
    Z1(i) = point3d_frag1{i}(1,3);
end

for i = 1:length(camA{2}) % first:last
    point3d_frag2{i} = triangulate(cell2mat(camA{2}(:,i))', ...
        cell2mat(camB{2}(:,i))', params);
```

```
    point3d_frag2{i} = point3d_frag2{i} / 1000; % convert from ...
        mm to m
    X2(i) = point3d_frag2{i}(1,1);
    Y2(i) = point3d_frag2{i}(1,2);
    Z2(i) = point3d_frag2{i}(1,3);
end

for i = 1:length(camA{3}) % first:last
    point3d_frag3{i} = triangulate(cell2mat(camA{3}(:,i))', ...
        cell2mat(camB{3}(:,i))', params);
    point3d_frag3{i} = point3d_frag3{i} / 1000; % convert from ...
        mm to m
    X3(i) = point3d_frag3{i}(1,1);
    Y3(i) = point3d_frag3{i}(1,2);
    Z3(i) = point3d_frag3{i}(1,3);
end

for i = 1:length(camA{4}) % first:last
    point3d_frag4{i} = triangulate(cell2mat(camA{4}(:,i))', ...
        cell2mat(camB{4}(:,i))', params);
    point3d_frag4{i} = point3d_frag4{i} / 1000; % convert from ...
        mm to m
    X4(i) = point3d_frag4{i}(1,1);
    Y4(i) = point3d_frag4{i}(1,2);
    Z4(i) = point3d_frag4{i}(1,3);
end

for i = 1:length(camA{5}) % first:last
    point3d_frag5{i} = triangulate(cell2mat(camA{5}(:,i))', ...
        cell2mat(camB{5}(:,i))', params);
    point3d_frag5{i} = point3d_frag5{i} / 1000; % convert from ...
        mm to m
    X5(i) = point3d_frag5{i}(1,1);
    Y5(i) = point3d_frag5{i}(1,2);
    Z5(i) = point3d_frag5{i}(1,3);
end

for i = 1:length(camA{6}) % first:last
    point3d_frag6{i} = triangulate(cell2mat(camA{6}(:,i))', ...
        cell2mat(camB{6}(:,i))', params);
    point3d_frag6{i} = point3d_frag6{i} / 1000; % convert from ...
        mm to m
    X6(i) = point3d_frag6{i}(1,1);
    Y6(i) = point3d_frag6{i}(1,2);
    Z6(i) = point3d_frag6{i}(1,3);
end

for i = 1:length(camA{7}) % first:last
    point3d_frag7{i} = triangulate(cell2mat(camA{7}(:,i))', ...
        cell2mat(camB{7}(:,i))', params);
```

```
    point3d_frag7{i} = point3d_frag7{i} / 1000; % convert from ...
        mm to m
    X7(i) = point3d_frag7{i}(1,1);
    Y7(i) = point3d_frag7{i}(1,2);
    Z7(i) = point3d_frag7{i}(1,3);
end

for i = 1:length(camA{8}) % first:last
    point3d_frag8{i} = triangulate(cell2mat(camA{8}(:,i))', ...
        cell2mat(camB{8}(:,i))', params);
    point3d_frag8{i} = point3d_frag8{i} / 1000; % convert from ...
        mm to m
    X8(i) = point3d_frag8{i}(1,1);
    Y8(i) = point3d_frag8{i}(1,2);
    Z8(i) = point3d_frag8{i}(1,3);
end

for i = 1:length(camA{9}) % first:last
    point3d_frag9{i} = triangulate(cell2mat(camA{9}(:,i))', ...
        cell2mat(camB{9}(:,i))', params);
    point3d_frag9{i} = point3d_frag9{i} / 1000; % convert from ...
        mm to m
    X9(i) = point3d_frag9{i}(1,1);
    Y9(i) = point3d_frag9{i}(1,2);
    Z9(i) = point3d_frag9{i}(1,3);
end

for i = 1:length(camA{10}) % first:last
    point3d_frag10{i} = triangulate(cell2mat(camA{10}(:,i))', ...
        cell2mat(camB{10}(:,i))', params);
    point3d_frag10{i} = point3d_frag10{i} / 1000; % convert from ...
        mm to m
    X10(i) = point3d_frag10{i}(1,1);
    Y10(i) = point3d_frag10{i}(1,2);
    Z10(i) = point3d_frag10{i}(1,3);
end


%% offset to reorient impact point to origin
plateX = -0.075;
plateY = -0.03757 - 0.0035;
plateZ = 1.053 - 0.005;

%% load and prepare velocity variable
velocities = ...
    [629.389816635431,121.635484788761,30.4024211995876,56.6372606021560,52.02479
save('saved_velocities.m', 'velocities')
velocities = velocities(2:end);
% velocities = velocities([1:4, 9:end]);

```

```
%% setup colors and color bar according to velocity range
minValue = 0; % min(velocities);
maxValue = max(velocities);
a = minValue:((maxValue - minValue)/255):maxValue;
fig = figure;
ax1 = axes(fig);
c = parula;
colormap(c);
A = colormap(c);
% generate colors for tracks
RGB = [];
vel3D = velocities;
for trail = 1:length(vel3D)
    for j = 1:length(a)
            absolute(j,trail) = abs(a(j) - abs(vel3D(trail)));
    end
    [val, idx] = min(absolute(:, trail));

    if idx > 256
        idx = 256;
    end
    RGB(trail, :, :, :) = A(idx, :);
end
%% plot 3D reconstructed paths
size_mark = 200; % size of markers in 3D scatter plots
nonzeroX = find(X1);
nonzeroY = find(Y1);
nonzeroZ = find(Z1);
for i = 1:length(nonzeroX)
    scatter3(X1(nonzeroX(i))- plateX,Y1(nonzeroY(i))- plateY,...
        Z1(nonzeroZ(i))- plateZ, size_mark, 'filled', ...
            'MarkerEdgeColor',...
        'black', 'MarkerFaceColor', 'black')
    hold on
    xlabel('x (m)')
    ylabel('y (m)')
    zlabel('z (m)')

end

size_mark = 100;
hold on
nonzeroX = find(X2);
nonzeroY = find(Y2);
nonzeroZ = find(Z2);
for i = 1:length(nonzeroX)
    scatter3(X2(nonzeroX(i))- plateX,Y2(nonzeroY(i))- ...
        plateY,Z2(nonzeroZ(i))- plateZ,size_mark, 'filled', ...
        'MarkerEdgeColor', RGB(1, :, :, :), 'MarkerFaceColor', ...
```

```
        RGB(1, :, :, :))
    hold on
    xlabel('x (m)')
    ylabel('y (m)')
    zlabel('z (m)')
end

hold on
nonzeroX = find(X3);
nonzeroY = find(Y3);
nonzeroZ = find(Z3);
for i = 1:length(nonzeroX)
    scatter3(X3(nonzeroX(i))- plateX,Y3(nonzeroY(i))- ...
        plateY,Z3(nonzeroZ(i))- plateZ, size_mark, 'filled', ...
        'MarkerEdgeColor', RGB(2, :, :, :), 'MarkerFaceColor', ...
        RGB(2, :, :, :))
    hold on
    xlabel('x (m)')
    ylabel('y (m)')
    zlabel('z (m)')
end
hold on
nonzeroX = find(X4);
nonzeroY = find(Y4);
nonzeroZ = find(Z4);
for i = 1:length(nonzeroX)
    scatter3(X4(nonzeroX(i))- plateX,Y4(nonzeroY(i))- ...
        plateY,Z4(nonzeroZ(i))- plateZ, size_mark, 'filled', ...
        'MarkerEdgeColor', RGB(3, :, :, :), 'MarkerFaceColor', ...
        RGB(3, :, :, :))
    hold on
    xlabel('x (m)')
    ylabel('y (m)')
    zlabel('z (m)')
end

hold on
nonzeroX = find(X9);
nonzeroY = find(Y9);
nonzeroZ = find(Z9);
for i = 1:length(nonzeroX)
    scatter3(X9(nonzeroX(i)) - plateX,Y9(nonzeroY(i)) - ...
        plateY,Z9(nonzeroZ(i)) - plateZ, size_mark, 'filled', ...
        'MarkerEdgeColor', RGB(8-4, :, :, :), ...
        'MarkerFaceColor',RGB(8 -4, :, :, :))
    hold on
    xlabel('x (m)')
    ylabel('y (m)')
    zlabel('z (m)')
end
```

```
hold on
nonzeroX = find(X10);
nonzeroY = find(Y10);
nonzeroZ = find(Z10);
for i = 1:length(nonzeroX)
    scatter3(X10(nonzeroX(i)) - plateX,Y10(nonzeroY(i)) - ...
        plateY,Z10(nonzeroZ(i)) - plateZ, size_mark, 'filled', ...
        'MarkerEdgeColor', RGB(9-4, :, :, :), 'MarkerFaceColor', ...
        RGB(9-4, :, :, :))
    hold on
    xlabel('x (m)')
    ylabel('y (m)')
    zlabel('z (m)')
end

%% define axes stemming from origin
hold on
hold all
q = quiver3(0,0,-max(zlim),0,0,2*max(zlim),'k','LineWidth',3);
q.ShowArrowHead = 'off';
q = quiver3(0,-max(ylim),0,0,2*max(ylim),0,'k','LineWidth',3);
q.ShowArrowHead = 'off';
q = quiver3(0,0,0,max(xlim),0,0,'k','LineWidth',3);
q.ShowArrowHead = 'off';
text(0,0,max(zlim),'Z','Color','k','FontSize',25)
text(0,max(ylim),0,'Y','Color','k','FontSize',25)
text(max(xlim),0,0,'X','Color','k','FontSize',25)

hold on
xlabel('x (m)')
ylabel('y (m)')
zlabel('z (m)')

ax = gca;
ax.FontSize = 60;
legend('incident projectile')
view(45,15)

%% color bar
fig2 = figure
ax2=axes(fig2,...
'Position',[ax1.Position(1)+ax1.Position(3) + ...
    .5,ax1.Position(2),0.0,0.4]);
axis off
set(ax2,'color','none');

cmap = parula;
cmap1 = colormap(cmap);
hcb = colorbar(ax2, 'Position', [.9 0.1 0.03 .8]);
```

```
ax2.CLim = [minValue,maxValue];
colorTitleHandle = get(hcb,'Title');
titleString = '\color{black} velocity (m/s)';
set(colorTitleHandle ,'String',titleString)
colormap(ax1, 'gray');

% hcb.Layout.Tile = 'east';
hcb.Visible = 'on';
hcb.TickLabelInterpreter = 'tex';
hcb.FontSize = 12;
hcb.Ticks = linspace(minValue, maxValue, 5);
tickRange = linspace(minValue, maxValue, 5);
black = [0,0,0];
tickLabelsCheck = hcb.TickLabels;
% ticksRound = [];
for ii = 1:numel(hcb.TickLabels)
    ticksRound(ii) = tickRange(ii);
    ticksRound(ii) = round(ticksRound(ii),3,'significant');
    hcb.TickLabels{ii} = [sprintf('\\color[rgb]{%f,%f,%f} ', ...
        black), num2str(ticksRound(ii))];
end


%% check plot with error bounds
figure
dif = 1;
for i = 1:length(p1)
    [Xdist, Ydist1, Zdist ] = sensitivity2(p1(i,:) , p2(i,:), ...
        stereoParams, dif);
    hold on
end
ax = gca;
ax.FontSize = 16;

%% FUNCTIONS
% distance 3D
function dist3d = dist(x1,y1,z1,x2,y2,z2)
    term1 = x1 - x2;
    term2 = y1 - y2;
    term3 = z1 - z2;
    dist3d = sqrt(term1^2 + term2^2 + term3^2);
end

% triangulation sensitivity
function [p1_xdist, p1_ydist, p2_xdist, p2_ydist ] = ...
    sensitivity(p1, p2, stereoParams, dif)


    p1_xplus = p1 + [dif 0];
```

```
    p1_xminus = p1 + [-dif 0];
    p1_yplus = p1 + [0 dif];
    p1_yminus = p1 + [0 -dif];

    p2_xplus = p2 + [dif 0];
    p2_xminus = p2 + [-dif 0];
    p2_yplus = p2 + [0 dif];
    p2_yminus = p2 + [0 -dif];


    p3D = triangulate(p1, p2, stereoParams)/1000;
    p3D_1 = triangulate(p1_xplus, p2, stereoParams)/1000;
    p3D_2 = triangulate(p1_xminus, p2, stereoParams)/1000;
    p3D_3 = triangulate(p1_yplus, p2, stereoParams)/1000;
    p3D_4 = triangulate(p1_yminus, p2, stereoParams)/1000;

    p3D_1b = triangulate(p1, p2_xplus, stereoParams)/1000;
    p3D_2b = triangulate(p1, p2_xminus, stereoParams)/1000;
    p3D_3b = triangulate(p1, p2_yplus, stereoParams)/1000;
    p3D_4b = triangulate(p1, p2_yminus, stereoParams)/1000;

    disp('p1_xdist')
    p1_xdist = pdist([p3D_1 ; p3D_2],'euclidean')
    disp('p1_ydist')
    p1_ydist = pdist([p3D_3 ; p3D_2],'euclidean')
    disp('p2_xdist')
    p2_xdist = pdist([p3D_1b ; p3D_2b],'euclidean')
    disp('p2_xdist')
    p2_ydist = pdist([p3D_3b ; p3D_4b],'euclidean')
%
%     p1_xdist = pdist([p3D_1 ; p3D_2],'euclidean')
%     p1_ydist = pdist([p3D_1 ; p3D_2],'euclidean')
%
%     p1_xdist = pdist([p3D_1 ; p3D_2],'euclidean')
%     p1_ydist = pdist([p3D_1 ; p3D_2],'euclidean')

%     figure
    scatter3(p3D(1), p3D(2), p3D(3), 'MarkerFaceColor', 'k')
%     hold on
    scatter3(p3D_1(1),p3D_1(2),p3D_1(3), 'MarkerFaceColor', 'r')
    scatter3(p3D_2(1),p3D_2(2),p3D_2(3), 'MarkerFaceColor', 'r')

    scatter3(p3D_3(1),p3D_3(2),p3D_3(3), 'MarkerFaceColor', 'g')
    scatter3(p3D_4(1),p3D_4(2),p3D_4(3), 'MarkerFaceColor', 'g')

    scatter3(p3D_1b(1),p3D_1b(2),p3D_1b(3), 'MarkerFaceColor', 'b')
    scatter3(p3D_2b(1),p3D_2b(2),p3D_2b(3), 'MarkerFaceColor', 'b')

    scatter3(p3D_3b(1),p3D_3b(2),p3D_3b(3), 'MarkerFaceColor', 'y')
    scatter3(p3D_4b(1),p3D_4b(2),p3D_4b(3), 'MarkerFaceColor', 'y')
```

```
end

% triangulation sensitivity
 function [Xdist, Ydist1, Zdist ] = sensitivity2(p1, p2, ...
    stereoParams, dif)

    size(p1)
    % add pixel differences to determine error/sensitivity of ...
       triangulation
    p1_xplus = p1 + [dif 0];
    p1_xminus = p1 + [-dif 0];
    p1_yplus = p1 + [0 dif];
    p1_yminus = p1 + [0 -dif];

    p2_xplus = p2 + [dif 0];
    p2_xminus = p2 + [-dif 0];
    p2_yplus = p2 + [0 dif];
    p2_yminus = p2 + [0 -dif];

    % triangulation of original point
    p3D = triangulate(p1, p2, stereoParams)/1000;
    % triangulation of original point with pixel differenecs in ...
       each view
    p3D_1 = triangulate(p1_xplus, p2, stereoParams)/1000;
    p3D_2 = triangulate(p1_xminus, p2, stereoParams)/1000;
    p3D_3 = triangulate(p1_yplus, p2, stereoParams)/1000;
    p3D_4 = triangulate(p1_yminus, p2, stereoParams)/1000;

    p3D_1b = triangulate(p1, p2_xplus, stereoParams)/1000;
    p3D_2b = triangulate(p1, p2_xminus, stereoParams)/1000;
    p3D_3b = triangulate(p1, p2_yplus, stereoParams)/1000;
    p3D_4b = triangulate(p1, p2_yminus, stereoParams)/1000;

    scatter3(p3D(1), p3D(2), p3D(3), 10, 'MarkerFaceColor', 'b')
    hold on

    % Y distance
    X = [p3D_3(1) p3D_4(1)];
    Y = [p3D_3(2) p3D_4(2)];
    Z = [p3D_3(3) p3D_4(3)];
    Ydist1 = dist(X(1),Y(1),Z(1),X(2),Y(2),Z(2));
    plot3(X,Y,Z, 'Color', 'r', 'LineStyle', '-', ...
       'DisplayName','T0');

    % Z distance
    X = [p3D_1b(1) p3D_2b(1)];
    Y = [p3D_1b(2) p3D_2b(2)];
    Z = [p3D_1b(3) p3D_2b(3)];
    Zdist = dist(X(1),Y(1),Z(1),X(2),Y(2),Z(2));
```

```
    plot3(X,Y,Z, 'Color', 'g', 'LineStyle', '-', ...
        'DisplayName','T1');

    % Y distance
    X = [p3D_3b(1) p3D_4b(1)];
    Y = [p3D_3b(2) p3D_4b(2)];
    Z = [p3D_3b(3) p3D_4b(3)];
    Ydist2 = dist(X(1),Y(1),Z(1),X(2),Y(2),Z(2));
    plot3(X,Y,Z, 'Color', 'b', 'LineStyle', '-', ...
        'DisplayName','T2');
    if Ydist2 > Ydist1
        Ydist1 = Ydist2;
    end

    % X distance
    X = [p3D_1(1) p3D_2(1)];
    Y = [p3D_1(2) p3D_2(2)];
    Z = [(p3D_1(3)+ p3D_2(3))/2 (p3D_1(3) + p3D_2(3))/2];
    Xdist = dist(X(1),Y(1),Z(1),X(2),Y(2),Z(2));
    plot3(X,Y,Z, 'Color', 'k', 'LineStyle', '-', ...
        'DisplayName','T3');
    hold on
    title('Stereo Calibration')
    xlabel('x (m)')
    ylabel('y (m)')
    zlabel('z (m)')

%     legend(h,'T0','T1','T2','T3');


 end

 function [pos1] = trimTracks1D(pos1)
    % for x and y positions, for each fragment, remove empty ...
        cells (1 Camera)
    for n = 1:length(pos1)
       empties = cellfun('isempty',pos1{n});
       pos1{n}(empties) = {NaN};
    end
end

% indices extraction
function [IndicesLastA, IndicesFirstA] = indicesExtraction(pos1, ...
    filenames1)
    % obtain start and stop indices for each fragment
    trails = length(filenames1);
    for trail = 1:trails
        savedTracks = pos1{trail};
        savedTracks = cell2mat(savedTracks);
        B = ¬isnan(savedTracks);
```

```
        IndicesLastA{trail} = find(B, 1, 'last');
        IndicesFirstA{trail} = find(B, 1, 'first');
    end
end

function [first, last] = matchFrames(fragA, fragB,  ...
    IndicesLastA, IndicesFirstA, IndicesLastB, IndicesFirstB)
    FirstA = IndicesFirstA{fragA};
    FirstB =  IndicesFirstB{fragB};
    LastA = IndicesLastA{fragA};
    LastB = IndicesLastB{fragB};

    % valid points to assign values
    if FirstA < FirstB
        first = FirstB;
    else
        first = FirstA;
    end

    if LastA < LastB
        last = LastA;
    else
        last = LastB;
    end
end
```

A.6 3D Kalman Filter

```
%% Sean Palmer, New Mexico Tech, SGDL 2022
% Kalman3D.m
% A code for performing Kalman Filtering of 3D trajectories. ...
    Generates
% plots of position and velocities vs time.
%% clear everything
clc
close all
clear all
%% user input for important parameters
suppress_graphs = 2; % Y = 1, N = 2
dt = 20e-6; % change in time step (time associated with frame rate)
rp = 1e6;
P_xyz = 10e3;
sigma_a = 1e3;

rp = 1e2;
P_xyz = 1e2;
```

```
sigma_a = 1e1;
 %% Initialize Kalman Filter Matrices
% Measurement Matrix
H = [1 0 0 0 0 0; ...
     0 1 0 0 0 0; ...
     0 0 1 0 0 0];

% dynamic matrix
A = [1 0 0 dt 0 0 ; ...
     0 1 0 0 dt 0 ; ...
     0 0 1 0 0 dt; ...
     0 0 0 1 0 0 ; ...
     0 0 0 0 1 0; ...
     0 0 0 0 0 1];
% Measurement Noise Covariance Matrix

R = [rp 0 0; 0 rp 0; 0 0 rp];
% Identity Matrix
I = eye(6);

% initial error covariance
% P = 100.0*np.eye(9)
P = P_xyz * I;
% P = [100 0 0 0 0 0; ...
%      0 100 0 0 0 0; ...
%      0 0 100 0 0 0; ...
%      0 0 0 10 0 0; ...
%      0 0 0 0 10 0; ...
%      0 0 0 0 0 10]

% Process Noise Covariance Matrix
c1 = dt^4/4; % constant 1
c2 = dt^3/2; % constant 2
c3 = dt^2; % constant 3

Q = [c1 0 0 c2 0 0 ; ...
     0 c1 0 0 c2 0 ; ...
     0 0 c1 0 0 c2 ; ...
     c2 0 0 c3 0 0 ; ...
     0 c2 0 0 c3 0 ; ...
     0 0 c2 0 0 c3 ]* sigma_a^2;

%% initialize position measurements
cd('V:\IndianHeadMarch2022\Test 10\test 10 3D points')
load('saved_position_comp_test10.mat')
measurements = [];

%% Preallocation of structure/arrays
for j = 1:45
    struct(j).xt = [];
```

```
    struct(j).yt = [];
    struct(j).zt = [];
    struct(j).dxt = [];
    struct(j).dyt = [];
    struct(j).dzt = [];

    struct(j).Zx = [];
    struct(j).Zy = [];
    struct(j).Zz = [];
    struct(j).Px = [];
    struct(j).Py = [];
    struct(j).Pz = [];
    struct(j).Pdx = [];
    struct(j).Pdy = [];
    struct(j).Pdz = [];

    struct(j).Kx = [];
    struct(j).Ky = [];
    struct(j).Kz = [];
    struct(j).Kdx = [];
    struct(j).Kdy = [];
    struct(j).Kdz = [];

    struct(j).K = {};
end

%% Run Kalman Filter process
Ntracks = length(trackX);
startTrack = 1;
endTrack = Ntracks; % startTrack %
fragtrack = 20;
for j =  startTrack:endTrack %fragtrack: fragtrack
    % obtain estimates for assumed initial velocities
    gradientX = gradient(trackX{j});
    gradientY = gradient(trackY{j});
    gradientZ = gradient(trackZ{j});
    velX = gradientX / dt;
    velY = gradientY / dt;
    velZ = gradientZ / dt;
    Nframes = length(trackX{j});
    x = [trackX{j}(1) trackY{j}(1) trackZ{j}(1), ...
        mean(velX(:)),mean(velY(:)), mean(velZ(:))]';

    % rewrite measurements into a new variable
    for n = 1:Nframes
        measurements(n,:) = [trackX{j}(n) trackY{j}(n) ...
            trackZ{j}(n)];
    end

    % perform Kalman filtering approach
```

```
    for n = 1:length(trackX{j})
        x = A * x;
        P = A*P*A' + Q;
        S = H*P*H' + R;
        K = (P*H') *  inv(S);
        Nframes = length(trackX{j});

        z = measurements(n, :);
        shapeH = size(H,1);
        z = reshape(z, [shapeH, 1]);

        y = z - (H*x);
        x = x + (K*y);
        P = (I - (K*H)) *  P;

        struct(j).xt = [struct(j).xt x(1)];
        struct(j).yt = [struct(j).yt x(2)];
        struct(j).zt = [struct(j).zt x(3)];
        struct(j).dxt = [struct(j).dxt x(4)];
        struct(j).dyt = [struct(j).dyt x(5)];
        struct(j).dzt = [struct(j).dzt x(6)];

        struct(j).Zx = [struct(j).Zx z(1)];
        struct(j).Zy = [struct(j).Zy z(2)];
        struct(j).Zz = [struct(j).Zz z(3)];
        struct(j).Px = [struct(j).Px P(1,1)];
        struct(j).Py = [struct(j).Py P(2,2)];
        struct(j).Pz = [struct(j).Pz P(3,3)];
        struct(j).Pdx = [struct(j).Pdx P(4,4)];
        struct(j).Pdy = [struct(j).Pdy P(5,5)];
        struct(j).Pdz = [struct(j).Pdz P(6,6)];


        struct(j).Kx = [struct(j).Kx K(1,1)];
        struct(j).Ky = [struct(j).Ky K(2,2)];
        struct(j).Kz = [struct(j).Kz K(3,3)];
        struct(j).Kdx = [struct(j).Kdx K(4,1)];
        struct(j).Kdy = [struct(j).Kdy K(5,2)];
        struct(j).Kdz = [struct(j).Kdz K(6,3)];
        struct(j).K = {struct(j).K K};
    end
end

%% plots vs time
for j = startTrack:endTrack % fragtrack:  fragtrack
    Nframes = length(trackX{j});
    t = 0:dt:((Nframes-1)*dt);
    figure
    plot(t, struct(j).dxt)
    hold on
```

```
    plot(t, struct(j).dyt)
    plot(t, struct(j).dzt)
    xlabel('time step')
    ylabel('Velocity')
    title('Velocity v. time step')
    legend('x velocity', 'y velocity','z velocity')
    %% x and y positions with time
    figure
    plot(t, struct(j).xt)
    hold on
    plot(t, struct(j).yt)
    plot(t, struct(j).zt)
    xlabel('time step')
    ylabel('Position')
    title('Position v. time step')
    legend('x position', 'y position', 'z position')
    %% uncertainty  associated with x and y positions
    figure
    plot(t, struct(j).Px)
    hold on
    plot(t, struct(j).Py)
    plot(t, struct(j).Pz)
    xlabel('time step')
    ylabel('Uncertainty')
    title('Position Uncertainty v. time step')
    legend('x position', 'y position', 'z position')
    %% uncertainty with x and y velocities
    figure
    plot(t, struct(j).Pdx)
    hold on
    plot(t, struct(j).Pdy)
    plot(t, struct(j).Pdz)
    xlabel('time step')
    ylabel('Uncertainty')
    title('Velocity Uncertainty  v. time step')
    legend('x vel', 'y vel', 'z vel')
    %% absolute, relative and percent error

    measured = trackX{j};
    estimated = struct(j).xt;
    [abs_dy, relerr, pererrX, mean_err, MSE, RSME] = ...
       determineError(measured, estimated);

    measured = trackY{j};
    estimated = struct(j).xt;
    [abs_dy, relerr, pererrY, mean_err, MSE, RSME] = ...
       determineError(measured, estimated);

    measured = trackZ{j};
    estimated = struct(j).zt;
```

```
    [abs_dy, relerr, pererrZ, mean_err, MSE, RSME] = ...
        determineError(measured, estimated);

    figure,
    plot(t(2:end),abs_dy)
    xlabel('time step')
    ylabel('Absolute Error [m]')
    title('Absolute Error  v. time step')

    figure,
    plot(t(2:end),relerr)
    xlabel('time step')
    ylabel('Relative Error')
    title('Relative Error v. time step')

    figure,
    plot(t(2:end),pererrX)
    xlabel('time step')
    ylabel('Percent Error')
    title('Percent Error  v. time step')

    x = struct(j).xt;
    y = struct(j).yt;
    z = struct(j).zt;

    if pererrX < struct(j).Px(end)
        dx = (struct(j).Px(end)/100) * x;
    else
        dx = (pererrX) * x;
    end

    if pererrY < struct(j).Py(end)

        dy = (struct(j).Py(end)/100) * y;
    else
        dy = max(pererrY) * y;
    end
    if pererrZ < struct(j).Pz(end)
        dz = (struct(j).Pz(end)/100) * z;
    else

        dz = max(pererrZ) * z;
    end
    dx = (struct(j).Px(end)/100) * x;
    dy = (struct(j).Py(end)/100) * y;
    dz = (struct(j).Pz(end)/100) * z;


    dt = 5e-9; % jitter time of SILUX laser
    t = 1/50000; % duration of one frame in sec
```

```

    vx = struct(j).xt;
    vy = struct(j).yt;
    vz = struct(j).zt;

    for n = 1:length(measured)
        frac_uncert_vx(n) = uncert_vcomp(x(n),dx(n),t,dt);
        frac_uncert_vy(n) = uncert_vcomp(y(n),dy(n),t,dt);
        frac_uncert_vz(n) = uncert_vcomp(z(n),dz(n),t,dt);
        dvx(n) = frac_uncert_vx(n) * vx(n);
        dvy(n) = frac_uncert_vy(n) * vy(n);
        dvz(n) = frac_uncert_vz(n) * vz(n);
        frac_uncert_v3D{j}(n) = uncert_v3D(vx(n),dvx(n), vy(n), ...
            dvy(n), vz(n), dvz(n));
    end


%      figure
    for n = 1:length(struct(j).dxt)
        velocity3D{j}(n) = mag3Dvel(struct(j).dxt(n), ...
            struct(j).dyt(n), struct(j).dzt(n));
    end
    time_n = 1:1:length(struct(j).dxt);
%      plot(time_n, velocity3D{j})

    figure
    uncert3D{j} = frac_uncert_v3D{j} .* velocity3D{j};
    shadedErrorBar(time_n, velocity3D{j}, [uncert3D{j} ...
        ;uncert3D{j} ], 'lineprops', '-g')
end

%% x and y velocities  with time
for n = 1:length(struct(j).dxt)
    velocity3D{j}(n) = mag3Dvel(struct(j).dxt(n), ...
        struct(j).dyt(n), struct(j).dzt(n));
end
time_n = 1:1:length(struct(j).dxt);
%      plot(time_n, velocity3D{j})
for kk = 1:length(velocity3D)
    final_vel(kk) = velocity3D{kk}(end);
end


%% x and y velocities uncertainties  with time
for j = fragtrack: fragtrack %startTrack:endTrack %fragtrack:  ...
    fragtrack % startTrack %
    Nframes = length(trackX{j});
    t = 0:dt:((Nframes-1)*dt);
    for k = 1:length(struct(j).Px)
        dx(k) = (struct(j).Px(k)/100) * x(k);
```

```
        dy(k) = (struct(j).Py(k)/100) * y(k);
        dz(k) = (struct(j).Pz(k)/100) * z(k);
        vx(k) = struct(j).xt(k);
        vy(k) = struct(j).yt(k);
        vz(k) = struct(j).zt(k);
    end

    dt = 5e-9; % jitter time of SILUX laser
    t = 1/50000; % duration of one frame in sec

    for n = 1:length(measured)
        frac_uncert_vx(n) = uncert_vcomp(x(n),dx(n),t,dt);
        frac_uncert_vy(n) = uncert_vcomp(y(n),dy(n),t,dt);
        frac_uncert_vz(n) = uncert_vcomp(z(n),dz(n),t,dt);
        dvx(n) = frac_uncert_vx(n) * vx(n);
        dvy(n) = frac_uncert_vy(n) * vy(n);
        dvz(n) = frac_uncert_vz(n) * vz(n);
        frac_uncert_v3D{j}(n) = uncert_v3D(vx(n),dvx(n), vy(n), ...
            dvy(n), vz(n), dvz(n));
    end

    for n = 1:length(struct(j).dxt)
        velocity3D{j}(n) = mag3Dvel(struct(j).dxt(n), ...
            struct(j).dyt(n), struct(j).dzt(n));
    end
    time_n = 1:1:length(struct(j).dxt);

    figure
    percent3D{j} = frac_uncert_v3D{j} * 100;
    uncert3D{j} = frac_uncert_v3D{j} .* (velocity3D{j});
    shadedErrorBar(time_n, velocity3D{j}, [uncert3D{j} ...
        ;uncert3D{j} ], 'lineprops', '-g')
end

%% Plot of each estimated coordinate and measured coordinate
xzypath = 'V:\IndianHeadMarch2022\Test 11\xyz';
figure
for j  = fragtrack:fragtrack
    for n = 1:length(trackX{j}) - 1
        subplot(1,3,1)
        % compare x estimates and measurements
        line([n,n+1],[struct(j).xt(n),struct(j).xt(n+1)],'Color', ...
            'b')
        line([n,n+1], [trackX{j}(n) trackX{j}(n+1)],'Color', 'g')
        xlabel('time')
        ylabel('X [m]')
        title('X position')
        hold on
        drawnow

```

```
        subplot(1,3,2)
        % compare y estimates and measurements
        line([n,n+1],[struct(j).yt(n),struct(j).yt(n+1)],'Color', ...
            'b')
        line([n,n+1], [trackY{j}(n) trackY{j}(n+1)],'Color', 'g')
        xlabel('time')
        ylabel('Y [m]')
        title('Y position')
        hold on

        drawnow
        subplot(1,3,3)
        % compare z estimates and measurements
        line([n,n+1],[struct(j).zt(n),struct(j).zt(n+1)],'Color', ...
            'b')
        line([n,n+1], [trackZ{j}(n) trackZ{j}(n+1)],'Color', 'g')
        xlabel('time')
        ylabel('Z [m]')
        title('Z position')
        hold on
        drawnow
    end
end



%% functions
function [RGB] = returnColorMap(vel3D, minValue, maxValue)
    a = minValue:((maxValue - minValue)/255):maxValue;
    % a_px = minVel:((maxVel - minVel)/255):maxVel;

    c = parula;
    colormap(c);
    A = colormap(c);
    % generate colors for tracks
    RGB = [];
    for trail = 1:length(vel3D)
        for j = 1:length(a)
                absolute(j,trail) = abs(a(j) - abs(vel3D(trail)));
        end
        [val, idx] = min(absolute(:, trail));

        if idx > 256
            idx = 256;
        end
        RGB(trail, :, :, :) = A(idx, :);
    end
end

function [trackX, trackY, trackZ] = desiredValues(X1, Y1, Z1)
```

```
    x1_first = find(¬isnan(X1), 1, 'first');
    x1_last = find(¬isnan(X1), 1, 'last');
    dif = x1_last - x1_first;
    for n = 1:dif
        trackX{1}(n) = X1(x1_first + n - 1);
        trackY{1}(n) = Y1(x1_first + n - 1);
        trackZ{1}(n) = Z1(x1_first + n - 1);
    end
end


function [frac_uncert_v3D, uncert3D, vel3D, time_n] = ...
   displayErrorVel(measured_x, measured_y, measured_z, x,y,z, ...
   vx, vy, vz)

    [abs_dx, relerr, pererr, mean_err, MSE, RSME] = ...
       determineError(measured_x, x);
    [abs_dy, relerr, pererr, mean_err, MSE, RSME] = ...
       determineError(measured_y, y);
    [abs_dz, relerr, pererr, mean_err, MSE, RSME] = ...
       determineError(measured_z, z);


    dx = (abs_dx) .* x(2:end);
    dy = (abs_dy) .* y(2:end);
    dz = (abs_dz) .* z(2:end);
    dt = 5e-9; % jitter time of SILUX laser
    t = 1/50000; % duration of one frame in sec


    for n = 1:length(measured_x)-1
        frac_uncert_vx(n) = uncert_vcomp(x(n),dx(n),t,dt);
        frac_uncert_vy(n) = uncert_vcomp(y(n),dy(n),t,dt);
        frac_uncert_vz(n) = uncert_vcomp(z(n),dz(n),t,dt);
        dvx(n) = frac_uncert_vx(n) * vx(n);
        dvy(n) = frac_uncert_vy(n) * vy(n);
        dvz(n) = frac_uncert_vz(n) * vz(n);
        frac_uncert_v3D(n) = uncert_v3D(vx(n),dvx(n), vy(n), ...
           dvy(n), vz(n), dvz(n));
    end


    figure
    for n = 1:length(vx)-1
        vel3D(n) = mag3Dvel(vx(n),vy(n), vz(n));
    end
    time_n = 1:1:length(x)-1;
    disp(size(frac_uncert_v3D))
    disp(size(vel3D))
    plot(time_n, vel3D)
```

```
    figure
    uncert3D = frac_uncert_v3D .* vel3D;
    shadedErrorBar(time_n, vel3D, [uncert3D ;uncert3D ], ...
        'lineprops', '-g')
end

%%
function [abs_dy, relerr, pererr, mean_err, MSE, RMSE]  = ...
    determineError(measured, estimated)
    for n = 1:length(measured) - 1
        y0 =  measured(n); % original value
        y1 = estimated(n); % obtained value
        dy = y0-y1 ; % error
        abs_dy(n) = abs(y0-y1) ;   % absolute error
        relerr(n) = abs(y0-y1)./y0 ;  % relative error
        pererr(n) = abs(y0-y1)./y0*100 ;   % percentage error
        mean_err(n) = mean(abs(y0-y1)) ;    % mean absolute error
        MSE(n) = mean((y0-y1).^2) ;         % Mean square error
        RMSE(n) = sqrt(mean((y0-y1).^2)) ; % Root mean square error
    end
end


%%
function [vel3D] =  mag3Dvel(velx, vely, velz)
    vel3D = sqrt((velx ^ 2) + (vely ^ 2) + (velz ^ 2));
end
 function [pos1] = trimTracks1D(pos1)
    % for x and y positions, for each fragment, remove empty ...
        cells (1 Camera)
    for n = 1:length(pos1)
       empties = cellfun('isempty',pos1{n});
       pos1{n}(empties) = {NaN};
    end
end

% indices extraction
function [IndicesLastA, IndicesFirstA] = indicesExtraction(pos1, ...
    filenames1)
    % obtain start and stop indices for each fragment
    trails = length(filenames1);
    for trail = 1:trails
        savedTracks = pos1{trail};
        savedTracks = cell2mat(savedTracks);
        B = ¬isnan(savedTracks);
        IndicesLastA{trail} = find(B, 1, 'last');
        IndicesFirstA{trail} = find(B, 1, 'first');
    end
end
```

```
function [first, last] = matchFrames(fragA, fragB,  ...
    IndicesLastA, IndicesFirstA, IndicesLastB, IndicesFirstB)
    FirstA = IndicesFirstA{fragA};
    FirstB =  IndicesFirstB{fragB};
    LastA = IndicesLastA{fragA};
    LastB = IndicesLastB{fragB};

    % valid points to assign values
    if FirstA < FirstB
        first = FirstB;
    else
        first = FirstA;
    end

    if LastA < LastB
        last = LastA;
    else
        last = LastB;
    end
end

% triangulation sensitivity
 function [Xdist, Ydist1, Zdist, max_norm_dist, min_norm_dist] = ...
     sensitivity2(p1, p2, stereoParams, dif)

    size(p1)
    % add pixel differences to determine error/sensitivity of ...
        triangulation
    p1_xplus = p1 + [dif 0];
    p1_xminus = p1 + [-dif 0];
    p1_yplus = p1 + [0 dif];
    p1_yminus = p1 + [0 -dif];

    p2_xplus = p2 + [dif 0];
    p2_xminus = p2 + [-dif 0];
    p2_yplus = p2 + [0 dif];
    p2_yminus = p2 + [0 -dif];

    % triangulation of original point
    p3D = triangulate(p1, p2, stereoParams)/1000;
    % triangulation of original point with pixel differenecs in ...
        each view
    p3D_1 = triangulate(p1_xplus, p2, stereoParams)/1000;
    p3D_2 = triangulate(p1_xminus, p2, stereoParams)/1000;
    p3D_3 = triangulate(p1_yplus, p2, stereoParams)/1000;
    p3D_4 = triangulate(p1_yminus, p2, stereoParams)/1000;

    p3D_1b = triangulate(p1, p2_xplus, stereoParams)/1000;
    p3D_2b = triangulate(p1, p2_xminus, stereoParams)/1000;
```

```
    p3D_3b = triangulate(p1, p2_yplus, stereoParams)/1000;
    p3D_4b = triangulate(p1, p2_yminus, stereoParams)/1000;

%     scatter3(p3D(1), p3D(2), p3D(3), 10, 'MarkerFaceColor', 'b')
%     hold on

    % Y distance
    X = [p3D_3(1) p3D_4(1)];
    Y = [p3D_3(2) p3D_4(2)];
    Z = [p3D_3(3) p3D_4(3)];
    Ydist1 = dist(X(1),Y(1),Z(1),X(2),Y(2),Z(2));
%     plot3(X,Y,Z, 'Color', 'r', 'LineStyle', '-', ... 'DisplayName','T0');

    % Z distance
    X = [p3D_1b(1) p3D_2b(1)];
    Y = [p3D_1b(2) p3D_2b(2)];
    Z = [p3D_1b(3) p3D_2b(3)];
    Zdist = dist(X(1),Y(1),Z(1),X(2),Y(2),Z(2));
%     plot3(X,Y,Z, 'Color', 'g', 'LineStyle', '-', ... 'DisplayName','T1');

    % Y distance
    X = [p3D_3b(1) p3D_4b(1)];
    Y = [p3D_3b(2) p3D_4b(2)];
    Z = [p3D_3b(3) p3D_4b(3)];
    Ydist2 = dist(X(1),Y(1),Z(1),X(2),Y(2),Z(2));
%     plot3(X,Y,Z, 'Color', 'b', 'LineStyle', '-', ... 'DisplayName','T2');
    if Ydist2 > Ydist1
        Ydist1 = Ydist2;
    end

    % X distance
    X = [p3D_1(1) p3D_2(1)];
    Y = [p3D_1(2) p3D_2(2)];
    Z = [(p3D_1(3)+ p3D_2(3))/2 (p3D_1(3) + p3D_2(3))/2];
    Xdist = dist(X(1),Y(1),Z(1),X(2),Y(2),Z(2));
%     plot3(X,Y,Z, 'Color', 'k', 'LineStyle', '-', ... 'DisplayName','T3');
%     hold on
%     title('Stereo Calibration')
%     xlabel('x (m)')
%     ylabel('y (m)')
%     zlabel('z (m)')
%
%     legend(h,'T0','T1','T2','T3');
    norm_dist(1) = norm(p3D_1); % m
    norm_dist(2) = norm(p3D_2); % m
    norm_dist(3) = norm(p3D_3); % m
```

```
    norm_dist(4) = norm(p3D_4); % m
    norm_dist(5) = norm(p3D_1b); % m
    norm_dist(6) = norm(p3D_2b); % m
    norm_dist(7) = norm(p3D_3b); % m
    norm_dist(8) = norm(p3D_4b); % m
    max_norm_dist = max( norm_dist);
    min_norm_dist = min( norm_dist);

 end

 % distance 3D
function dist3d = dist(x1,y1,z1,x2,y2,z2)
    term1 = x1 - x2;
    term2 = y1 - y2;
    term3 = z1 - z2;
    dist3d = sqrt(term1^2 + term2^2 + term3^2);
end


function frac_uncert_vcomp = uncert_vcomp(x,dx,t,dt)
    frac_uncert_vcomp = sqrt((dx/x)^2 + (dt/t)^2);
end

function frac_uncert_v3D = uncert_v3D(vx,vdx, vy, dvy, vz, dvz)
    frac_uncert_v3D = sqrt((vdx/ vx)^2 +  (dvy/vy)^2 + (dvz/vz)^2);
end
```

A.7 Dynamic Image Analysis

```
%% Sean Palmer, New Mexico Tech, SGDL 2022
% DynamicImageAnalysis.m
% A code for performing dynamic image analysis by extracting and
% observing the size of fragments as they are tracked with time.
%% clear everything
clc
close all
clear all


%% user input
pathname1 = 'V:\IndianHeadMarch2022\Test 10\test 10 25736\actual';
pathname2 = 'V:\IndianHeadMarch2022\Test 10\saveTracksBreduced';
cd(pathname1);
Cam1 = dir('*tiff'); % list of all tif images

startFrame = 13;
```

```
endFrame = 20;
minimumSize = 1;
maximumSize = 2000;

getSample = 9; % Y = 1, N = 2
desiredFrame = 196;
%% run through desired mat files
fragNumber = 70;
%%
for kk = fragNumber:fragNumber%numel(filenames)
    cd(pathname2);
    filenames = dir('*mat');
    S = load(filenames(kk).name); % Best to load into an output ...
        variable.
    startFrame = S.startFrame;
    endFrame = length(S.x_centroid);
    for i = startFrame:endFrame
        % show image
        cd(pathname1);
        Cam1 = dir('*tiff'); % list of all tif images

        mainImage = imread(Cam1(i).name);
            [rows,cols,dims] = size(mainImage);
        if dims == 3
            I = rgb2gray(mainImage);
        else
         I = mainImage;
        end
        level = graythresh(I);
        BW = ¬imbinarize(I, level);
        [BW,maskedImage] = segmentImage2(I);

%          figure, imshow(BW);
%          hold on
        movegui('north');
        [L, num_Obj] = bwlabel(BW, 8);
        %% load and plot centroid
        cd(pathname2);
        filenames = dir('*mat');
        S = load(filenames(kk).name); % Best to load into an ...
            output variable.
        trail_pos_x = S.x_centroid{i};
        trail_pos_y = S.y_centroid{i};
%            plot(trail_pos_x, ...
    trail_pos_y,'-','color',RGB(trail,:,:,:),'linewidth',1) ...
    %show estimated path
%          plot(trail_pos_x, trail_pos_y,'r+') %show estimated path
        xCenter = round(trail_pos_x);
        yCenter = round(trail_pos_y);

```

```
        labelNumber = L(yCenter, xCenter);
        extractedObject = ismember(L, labelNumber);
        testObject = ismember(L, labelNumber);

%         figure, imshow(extractedObject, []);
        %% determine properties
        properties = regionprops(extractedObject, 'all');
        fragmentAreas = [properties.Area];
        desiredSizes = (fragmentAreas < maximumSize) & ...
            (fragmentAreas > minimumSize); % desired sizes of ...
            fragments
        desired_indices = find(desiredSizes);
        if ¬isempty(desired_indices)
            desiredFragmentImage = ismember(extractedObject, ...
                desired_indices);
            labeledDesiredImage = bwlabel(desiredFragmentImage); ...
                    % Label each blob so we can make ...
                measurements of it
            properties2 = regionprops(labeledDesiredImage, 'all');

            % sort set number of particles by area, largest to ...
                smallest
            get_particles = [properties2.Area];
            centroids2 = [properties2.Centroid];
            equivDiam2 = [properties2.EquivDiameter];
            boundBox = [properties2.BoundingBox];

            numberOfBoxes = length(equivDiam2);
            boxes = reshape(boundBox,[],numberOfBoxes);
            x_cent = centroids2(1:2:end-1);
            y_cent = centroids2(2:2:end);

            for n = 1:length(x_cent)
                xCent(n) = x_cent(n);
                yCent(n) = y_cent(n);
            end
            %% show image boxes
%             figure, imshow(mainImage);
            for k = 1 : length(properties2)
              thisBB =  properties2(k).BoundingBox;
              rectangle('Position', ...
                  [thisBB(1),thisBB(2),thisBB(3),thisBB(4)],...
              'EdgeColor','g','LineWidth',1 )
            end

            cellPoint1 = {};
            cellPoint2 = {};
            cellDp = {};

            comp = imcomplement(I);
```

```
      for jj = 1:length(equivDiam2)
        temp{jj} = generateTemp(comp, ...
          properties2(jj).BoundingBox);
      %    [cellPoint1{i}, cellPoint2{i}, cellDp{i}] = ...
        corr(temp{i}, I1, I2);
      end
      %% otsu and area
%       figure, imshow(comp, [])
%       figure, imshow(temp{1}, [])
      level = graythresh(temp{1});
      bin_temp = imbinarize(temp{1}, level);
%       figure, imshow(temp{1})

      [threshold, maxval, idx, area] = otsu_return(temp{1});
      [save_black, lowerArea, upperArea] = ...
        area_grad(temp{1}, idx);

      if getSample == 1 && i == desiredFrame
%          [dummy1, dummy2, dummy3, dummy4, dummy5] = ...
  area_grad_display(temp{1}, idx);
%          otsu_full(temp{1})
        save_temp = temp{1};
      end
      frame(i) = i;
      save_area(i) = area;
      save_upper(i) = upperArea;
      save_lower(i) = lowerArea;
    end
  end
end
%% display area with time
x = frame(startFrame:end);
y = save_area(startFrame:end);
errA = abs(y - save_upper(startFrame:end));
errB = abs(y - save_lower(startFrame:end));

figure,
h(1) = plot(x, y, 'r')
hold on
shadedErrorBar(x, y, [errA;errB], 'lineprops', '-r')
xlabel('frame number')
ylabel('pixel area')

title('Pixel area vs frame number')
mean_y = mean(y) * ones(size(y));

hold on
h(2) = plot(x,mean_y, 'b')
lgd = legend(h, 'Pixel Area', 'Mean Pixel Area');
lgd.Location = 'northwest';
```

```
cd('V:\IndianHeadMarch2022\Test 10\saved sizes')
s1 = 'DIA_test10_Bfrag00';
s2 = num2str(fragNumber);
s3 = '.mat';
dir_save = strcat(s1,s2,s3)
save(dir_save, 'x', 'y', 'errA', 'errB', 'startFrame', ...
    'save_area', 'save_upper', 'save_lower', 'frame')
%% Get Sample Otsu/Area/Area Gradient Images
lvl = graythresh(save_temp);
bwtemp = imbinarize(save_temp, lvl);
figure, imshow(save_temp, [])
figure, imshow(bwtemp)

%% display area, area gradient, threshold range, etc.
otsu_full(save_temp)
[threshold, maxval, idx, area] = otsu_return(temp{1});
[dummy1, dummy2, dummy3, dummy4, dummy5]  = ...
    area_grad_display(temp{1}, idx);
[dummyA, dummyB, dummyC]  = area_grad(temp{1}, idx);

%% show saved bounding box of desired fragment
figure, imshow(save_temp, []);
figure, imhist(im2uint8(save_temp));

%% functions
function dist3d = dist(x1,y1,z1,x2,y2,z2)
term1 = x1 - x2;
term2 = y1 - y2;
term3 = z1 - z2;
dist3d = sqrt(term1^2 + term2^2 + term3^2);
end

function temp = generateTemp(image, boundingBox)
    coord = boundingBox;
    temp = imcrop(image, [coord(1) coord(2) coord(3) coord(4)]);
end

% save area for single region
function [save_black,less, more] =  area_grad(region, index)
    [rowTemp, colTemp] = size(region);
    for thr = 1:255
        thr2 = 256 - thr;
        Binar = (imbinarize(region,thr2/255));
        label = bwlabel(Binar,8);
        tempRegion = regionprops(label, 'all');
        tempArea = [tempRegion.Area];
    %     save_thr(thr) = thr;
        nBlack = sum(Binar(:));
        nWhite = numel(Binar ) - nBlack;
        save_black(thr) = nBlack;
```

```
        save_white(thr) = nWhite;
        save_thr2(thr) = thr2;
    end
    if length(index) > 1

        less = save_black(save_thr2(index(1)));
        more = save_black(save_thr2(index(2)));
    else
        asdf1 = index - 1;
        asdf2 = index + 1;

        less = save_black(save_thr2(asdf1)); % save_thr(idx)
        more = save_black(save_thr2(asdf2));
    end
end


function [less, more, save_black, save_white, save_thr2] = ...
    area_grad_display(region, index)
    [rowTemp, colTemp] = size(region);
    for thr = 1:256
        thr2 = 256 - thr;
        Binar = (imbinarize(region,thr2/256));
        label = bwlabel(Binar,8);
        tempRegion = regionprops(label, 'all');
        tempArea = [tempRegion.Area];
    %     save_thr(thr) = thr;
        nBlack = sum(Binar(:));
        nWhite = numel(Binar ) - nBlack;
        save_black(thr) = nBlack;
        save_white(thr) = nWhite;
        save_thr2(thr) = thr2;
    end
    figure,
    h(1) = plot(save_thr2,save_black)
    yyaxis left
    areaGradient = gradient(save_black);
    hold on
    yyaxis right
    ylim([0 max(areaGradient)]);
    h(2) = plot(save_thr2, areaGradient)
    yyaxis left

    % title('Plots with Different y-Scales')
    xlabel('threshold')
    ylabel('Area (px)')
    legend('Area', 'Gradient of Area')
    yyaxis right
    ylabel('Gradient of Area')
    title('Area and Gradient of Area vs Threshold')
```

```
    hold on,
    if length(index) > 1
        h(3) = line([index(1) index(2)],[0 ...
            max(save_black)],'Color',[1 0 0]);
        [fillhandle,msg]=jbfill([index(1) ...
            index(2)],save_black(index(1):index(2)),[0 0],'r', 'r');
        less = save_black(save_thr2(index(1)));
        more = save_black(save_thr2(index(2)));
    else
        asdf1 = index - 1;
        asdf2 = index + 1;
        h(3) = line([index index],[0 max(save_black)],'Color',[1 ...
            0 0]);
        hold on,
        h(4) = line([asdf1 asdf1],[0 max(save_black)],'Color',[1 ...
            0 0]);
        hold on,
        h(5) = line([asdf2 asdf2],[0 max(save_black)],'Color',[1 ...
            0 0]);
        [fillhandle,msg]=jbfill([asdf1 index ...
            asdf2],save_black(asdf1:asdf2),[0 0 0],'r', 'r');

        less = save_black(save_thr2(asdf1)); % save_thr(idx)
        more = save_black(save_thr2(asdf2));
    end
    legend(h,'area','area ...
        gradient','threshold','leftBound','rightBound');
    lgd = legend(h,'area','area gradient','threshold range');
    lgd.Location = 'southwest'
end

% return otsu parameters
function [threshold, maxval, idx, area] = otsu_return(I)
    sig_save = [];
    BetClassvariance = [];

%     I = temp{1};
    [rows,cols,dims] = size(I);
    mg = mean(I(:));

    BetClassvariance = zeros(1,256);

    n=imhist(I);
    N=sum(n);
    maxS=0;
    for i=1:256
        P(i)=n(i)/N;
    end
    for T=2:255
        w0=sum(P(1:T));
```

```
        w1=sum(P(T+1:256));
        u0=dot([0:T-1],P(1:T))/w0;
        u1=dot([T:255],P(T+1:256))/w1;
        sigma=w0*w1*((u1-u0)^2);
        sig_save(T) = sigma;
        BetClassvariance(T) = sigma^2;
        if sigma>maxS
            maxS=sigma;
            threshold=T-1;
        end
    end
    bw = imbinarize(I,threshold/255);
    maxval = max(BetClassvariance);
    idx = find(BetClassvariance == maxval);

    props = regionprops(bw, 'all');
    % sort set number of particles by area, largest to smallest
    area = max([props.Area]);

end

% show graphs of otsu parameters
function otsu_full(I)

    sig_save = [];
    BetClassvariance = [];

%     I = temp{1};
    [rows,cols,dims] = size(I);

     % Plot its histogram;
    [Frequency,bins] = imhist(I);
    figure,stem(bins,Frequency);title('Frequency ...
        Plot');xlabel('Intensities'),ylabel('Freq');

    % Compute Global mean
    mg = mean(I(:));
    hold on, line([mg mg],[0 max(Frequency)],'Color',[1 0 0]);

    % Let the threshold value varies from k = 0 to 255
    BetClassvariance = zeros(1,256);
    Goodness = BetClassvariance;
    NormalizedFreq = Frequency / (rows * cols);
    figure,stem(bins,NormalizedFreq);title('Normalized ...
        Frequency');xlabel('Intensities'),ylabel('Freq');
    hold on, line([mg mg],[0 max(NormalizedFreq)],'Color',[1 0 0]);
    SigmaGlobal = var(double(I(:)));


    figure(1),imshow(I);
```

```
    figure(2),imhist(I);
    n=imhist(I);
    N=sum(n);
    maxS=0;
    for i=1:256
        P(i)=n(i)/N;
    end
    for T=2:255
        w0=sum(P(1:T));
        w1=sum(P(T+1:256));
        u0=dot([0:T-1],P(1:T))/w0;
        u1=dot([T:255],P(T+1:256))/w1;
        sigma=w0*w1*((u1-u0)^2);
        sig_save(T) = sigma;
        BetClassvariance(T) = sigma^2;
        if sigma>maxS
            maxS=sigma;
            threshold=T-1;
        end
    end
    bw=im2bw(I,threshold/256);
    figure(3),imshow(bw);
    figure,plot(BetClassvariance);
    xlabel('Thresold Values'),ylabel('Between Class Variance');

    title('BetweenClass Variance')
    [¬,index]= max(BetClassvariance);
    hold on, line([index index],[0 ...
        max(BetClassvariance)],'Color',[1 0 0]);
    figure(2),hold on, line([index index],[0 ...
        max(Frequency)],'Color',[1 1 0]);

    figure,plot(sig_save);xlabel('Threshold ...
        Values'),ylabel('Between Class Variance');

    maxval = max(BetClassvariance);
    idx = find(BetClassvariance == maxval);
end

function[fillhandle,msg]=jbfill(xpoints,upper,lower,color,edge,add,transparency)
    %USAGE: ...
        [fillhandle,msg]=jbfill(xpoints,upper,lower,color,edge,add,transparency)
    %This function will fill a region with a color between the ...
        two vectors provided
    %using the Matlab fill command.
    %
    %fillhandle is the returned handle to the filled region in ...
        the plot.
    %xpoints= The horizontal data points (ie frequencies). Note ...
        length(Upper)
```

```
    %         must equal Length(lower)and must equal ...
        length(xpoints)!
    %upper = the upper curve values (data can be less than lower)
    %lower = the lower curve values (data can be more than upper)
    %color = the color of the filled area
    %edge  = the color around the edge of the filled area
    %add   = a flag to add to the current plot or make a new one.
    %transparency is a value ranging from 1 for opaque to 0 for ...
        invisible for
    %the filled color only.
    %
    %John A. Bockstege November 2006;
    %Example:
    %     a=rand(1,20);%Vector of random data
    %     b=a+2*rand(1,20);%2nd vector of data points;
    %     x=1:20;%horizontal vector
    %     [ph,msg]=jbfill(x,a,b,rand(1,3),rand(1,3),0,rand(1,1))
    %     grid on
    %     legend('Datr')
    if nargin<7;transparency=.5;end %default is to have a ...
        transparency of .5
    if nargin<6;add=1;end     %default is to add to current plot
    if nargin<5;edge='k';end  %dfault edge color is black
    if nargin<4;color='b';end %default color is blue
    if length(upper)==length(lower) && ...
        length(lower)==length(xpoints)
        msg='';
        filled=[upper,fliplr(lower)];
        xpoints=[xpoints,fliplr(xpoints)];
        if add
            hold on
        end
        fillhandle=fill(xpoints,filled,color);%plot the data
        set(fillhandle,'EdgeColor',edge,'FaceAlpha',transparency,'EdgeAlpha',tra
            edge color
        if add
            hold off
        end
    else
        msg='Error: Must use the same number of points in each ...
            vector';
    end
end

function [BW,maskedImage] = segmentImage(RGB)
%segmentImage Segment image using auto-generated code from ...
    imageSegmenter app
%  [BW,MASKEDIMAGE] = segmentImage(RGB) segments image RGB using
%  auto-generated code from the imageSegmenter app. The final ...
    segmentation
```

```
%  is returned in BW, and a masked image is returned in MASKEDIMAGE.

% Auto-generated by imageSegmenter app on 17-Mar-2022
%----------------------------------------------------


% Convert RGB image into L*a*b* color space.
X = rgb2lab(RGB);

% Create empty mask.
BW = false(size(X,1),size(X,2));

% Flood fill
row = 211;
column = 403;
tolerance = 1.500000e-01;
normX = sum((X - X(row,column,:)).^2,3);
normX = mat2gray(normX);
addedRegion = grayconnected(normX, row, column, tolerance);
BW = BW | addedRegion;

% Invert mask
BW = imcomplement(BW);

% Create masked image.
maskedImage = RGB;
maskedImage(repmat(¬BW,[1 1 3])) = 0;
end


function [BW,maskedImage] = segmentImage2(X)
%segmentImage Segment image using auto-generated code from ...
    imageSegmenter app
%  [BW,MASKEDIMAGE] = segmentImage(X) segments image X using ...
    auto-generated
%  code from the imageSegmenter app. The final segmentation is ...
    returned in
%  BW, and a masked image is returned in MASKEDIMAGE.

% Auto-generated by imageSegmenter app on 26-Mar-2022
%----------------------------------------------------


% Adjust data to span data range.
X = imadjust(X);

% Threshold image - adaptive threshold
BW = imbinarize(X, 'adaptive', 'Sensitivity', 1.000000, ...
    'ForegroundPolarity', 'bright');

```

```
% Invert mask
BW = imcomplement(BW);

% Create masked image.
maskedImage = X;
maskedImage(¬BW) = 0;
end
```

APPENDIX B

DERIVATION OF UNCERTAINTY IN EQUIVALENT DIAMETER

Define the equivalent diameter d_e in terms of pixel area A_{px} and the spatial calibration scale C_s:

(B.1)

Apply the generalized uncertainty equation to the equivalent diameter d_e:

$$\partial d_e = \sqrt{(\frac{\partial d_e}{\partial C_s}\delta C_s)^2 + (\frac{\partial d_e}{\partial A_{px}}\delta A_{px})^2} \tag{B.2}$$

Apply partial derivatives to obtain terms fractional uncertainties:

$$\frac{\partial d_e}{\partial C_s}\delta C_s = \sqrt{\frac{4A_{px}}{\pi}}\delta C_s = d_e\frac{\delta C_s}{C_s} \tag{B.3}$$

$$\frac{\partial d_e}{\partial A_{px}}\delta A_{px} = \frac{\partial}{\partial A}(\sqrt{\frac{4A}{\pi}}C_s)\delta A_{px} = \frac{C_s}{\sqrt{\pi A_{px}}}\delta A_{px}$$

Simplify to obtain fractional uncertainty of A_{px}:

$$\frac{d_e}{(\frac{C_s}{\sqrt{\pi A_{px}}})} = 2A_{px} \tag{B.4}$$

$$\frac{\partial d_e}{\partial A_{px}}\delta A_{px} = \frac{d_e}{2A_{px}}\delta A_{px} = \frac{d_e}{2}\frac{\delta A_{px}}{A_{px}} \tag{B.5}$$

Substitute terms into generalized uncertainty equation:

$$\delta d_e = \sqrt{(d_e\frac{\delta C_s}{C_s})^2 + (\frac{d_e}{2}\frac{\delta A_{px}}{A_{px}})^2} \tag{B.6}$$

Simplify the generalized uncertainty equation to obtain the fractional uncertainty of the diameter:

$$\frac{\delta d_e}{d_e} = \sqrt{(\frac{\delta C_s}{C_s})^2 + (\frac{\delta A_{px}}{2A_{px}})^2} \tag{B.7}$$

This agrees with the general rule of uncertainty in a power described by Taylor [40].

THREE-DIMENSIONAL FRAGMENT TRACKING AND SIZE ESTIMATION
USING STEREO FOCUSED SHADOWGRAPHY

by

Sean Palmer

NEW MEXICO TECH
SCIENCE • ENGINEERING • RESEARCH UNIVERSITY