TOWARDS AUTOMATED AERIAL REFUELING: REAL TIME POSITION ESTIMATION WITH STEREO VISION

THESIS

Bradley D. Denby

AFIT-ENG-MS-16-M-252

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-16-M-252

TOWARDS AUTOMATED AERIAL REFUELING: REAL TIME POSITION ESTIMATION WITH STEREO VISION

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Bradley D. Denby, B.S.Phy.

March 24, 2016

AFIT-ENG-MS-16-M-252

TOWARDS AUTOMATED AERIAL REFUELING:
REAL TIME POSITION ESTIMATION WITH STEREO VISION

THESIS

Bradley D. Denby, B.S.Phy.

Committee Membership:

Maj Brian Woolley, PhD
Chair

Maj John Pecarina, PhD
Member

Dr. John Raquet
Member

AFIT-ENG-MS-16-M-252

Abstract

Aerial refueling is essential to the United States Air Force (USAF) core mission of rapid global mobility. However, in-flight refueling is not available to remotely piloted aircraft (RPA) or unmanned aerial systems (UAS). As reliance on drones for intelligence, surveillance, and reconnaissance (ISR) and other USAF core missions grows, the ability to automate aerial refueling for such systems becomes increasingly critical. New refueling platforms include sensors that could be used to estimate the relative position of an approaching aircraft. Relative position estimation is a key component to solving the automated aerial refueling (AAR) problem. Analysis of data from a one-seventh scale, real world refueling scenario demonstrates that the relative position of an approaching aircraft can be estimated at rates between 10 Hz and 30 Hz using stereo vision. Linear regression models on position estimate accuracies predict results reported by other research in the simulation domain, suggesting that real world accuracies are comparable to simulation domain accuracies reported by others. Further, by seeding the position estimation algorithm with previous position estimates, subsequent errors in position estimation are reduced.

Table of Contents

Page

List of Figures

List of Tables

TOWARDS AUTOMATED AERIAL REFUELING:
REAL TIME POSITION ESTIMATION WITH STEREO VISION

I. Introduction

Aerial refueling is essential to the United States Air Force (USAF) core mission of rapid global mobility. However, in-flight refueling is not available to remotely piloted aircraft (RPA) or unmanned aerial systems (UAS). As reliance on drones for intelligence, surveillance, and reconnaissance (ISR) and other USAF core missions grows, the ability to automate aerial refueling for such systems becomes increasingly critical [15].

Solving the automated aerial refueling (AAR) problem aims to fix this issue. AAR consists of many challenges, including relative pose estimation between aircraft, navigation and control processes, and data transfer between aircraft. Researchers have been investigating the AAR problem for over a decade [34]. Some solutions to the relative navigation aspect of AAR have focused on the global positioning system (GPS), inertial sensor data, monocular machine vision, and light detection and ranging (LiDaR) [5, 10, 17].

Work towards solving the AAR problem involving monocular machine vision and LiDaR have approached the challenge from the perspective of the receiving aircraft. In these studies, the aircraft uses onboard sensors in order to detect the refueling tanker and estimate a relative position [10, 17]. This approach requires onboard sensors that may not be present in the original aircraft designs.

Recently, the KC-X Tanker Modernization Program has introduced the KC-46 refueling tanker for future deployment [14]. This tanker could be capable of employing

a stereo imaging system during refueling operations [3]. Utilizing existing sensors towards solving the AAR problem can reduce costs and speed system deployment. As a result, recent research focused on using stereo sensors on the refueling tanker for relative position estimation of an approaching aircraft [42].

While other work examined relative position estimation in the simulation domain, several questions regarding the utility of stereo vision in support of solving the AAR problem remain. For example, does position estimation using stereo vision on real world data result in accuracies comparable to accuracies observed in the simulation domain? Is position estimation from stereo vision feasible in real time, and does real time execution have an effect on position estimation accuracies? Do different stereo vision algorithms exhibit different position estimation accuracies? Finally, does seeding the registration algorithm with solutions from previous time steps affect position estimation accuracy?

1.1 Problem outline

Here, the central problem consists of position estimation from stereo image data. Specifically this investigation focuses on estimating an object's position relative to the stereo camera, given a pair of rectified stereo camera images and the associated region of interest masks. This problem has been examined from the perspective of AAR by Werner [42]. Werner's work examined the accuracy of relative position estimation from simulated stereo vision using semi-global block matching on the CPU and an unseeded iterative closest point registration algorithm. An emphasis was placed on high-fidelity three dimensional (3D) reconstruction by tuning correspondence function parameters at the expense of high execution times [42].

Because simulated data lacks real world noise factors, results in the simulation domain may exhibit smaller errors. This effect has been observed in other AAR

investigations. Specifically, real world flight tests have exhibited greater errors than errors exhibited in simulation [5]. A main thrust of this investigation focuses on evaluating position estimation accuracies from real world stereo imagery. The goal for this aspect of the investigation is to test whether position estimation accuracies produced from real world data differ from position estimation accuracies produced from simulated data.

Another question central to this investigation relates to real time execution. Previous work demonstrated that stereo vision can produce relative position estimates. However, each pair of stereo images required over one second of processing to produce a position estimate [42]. Real time execution is an essential requirement for the feasibility of stereo vision as a solution to the relative navigation aspect of AAR. Thus, this investigation examines whether execution at a rate greater than 1 Hz is feasible, noting that an online system would prefer an execution rate exceeding the system frame rate. Additionally, changes to support real time execution could impact relative position estimation accuracies.

Several stereo correspondence algorithms exist. Due to different approaches to stereo analysis, the execution times and the resulting disparity maps of different stereo algorithms vary. Previous work emphasized tuning the function parameters of a semi-global block matching stereo algorithm in order to obtain high fidelity 3D reconstructions at the expense of high execution times [42]. In order to examine the effect of different stereo algorithms on position estimates, three different stereo algorithms with identical input parameters are tested.

Finally, this investigation considers the effect of seeding the point cloud registration function. In other work, a source or truth point cloud was registered to a target point cloud generated from stereo correspondence data in order to estimate position. The iterative closest point algorithm was used for registration. At each time step,

the source point cloud was placed at the origin of the camera frame of reference before performing iterations [42]. Because the position of the imaged aircraft is highly related between time frames, seeding the position of the source point cloud with the solution from the previous time frame could improve position estimation accuracy.

1.2 General approach

Evaluating position estimation accuracies from real world stereo imagery requires a real world data collection. Specifically, stereo imagery of an aircraft from the perspective of a refueling tanker at various distances is required. A full, one-to-one scale data collection is not feasible. Instead, real world stereo imagery was collected at a one-seventh scale. A 1:7 scale F-15E was imaged with a pair of cameras at distance vectors varying from 2 m to 8 m in magnitude. These distances correspond to distance vectors ranging from 14 m to 56 m in a full scale scenario.

In order to support reproducibility, a database schema was developed for collecting, organizing, storing, and retrieving this real world stereo data. By wrapping the database in an application programming interface (API), the collected data may be accessed and analyzed programatically. The database structure is essential in making large sets of data feasible.

Unlike the simulation domain, truth data is not intrinsically available in the real world domain. Truth data is collected in parallel with the collection of stereo imagery. All data collection takes place within a Vicon motion capture area. The poses of the model F-15E and each camera are tracked by the motion capture system.

In order to construct a correspondence between stereo imagery and motion capture data, precise timing is required. Specifically, clock times must be synchronized between the motion capture computer, the stereo data collection computer, and each camera. In order to achieve clock synchronization, these devices were connected to a

local area network (LAN). A stratum 1 network time protocol (NTP) server provided time synchronization services to these devices.

Evaluating the feasibility of real time execution requires robust, high-quality computer vision software. Under this requirement and the desire to support reproducibility, OpenCV 3.0.0 and Point Cloud Library (PCL) 1.8.0 were used for data analysis. OpenCV 3.0.0 provides stereo correspondence algorithms that utilize both the CPU and the GPU. Faster execution due to parallelization on the GPU is an important consideration for real time execution.

In particular, real world data was analyzed using semi-global block matching on the CPU, block matching on the CPU, and block matching on the GPU. Assuming an input of rectified stereo image pairs and corresponding region of interest masks, these algorithms produce a disparity map. Disparity maps may be projected into a three dimensional point cloud using calibration information.

After producing a point cloud from stereo imagery, the iterative closest point (ICP) algorithm is applied in order to estimate position. Specifically, a point cloud generated from a computer model of an F-15E is iteratively perturbed so as to reduce the distance to the nearest neighbor points in the target point cloud. Two initial conditions are tested for this algorithm. In one case, the source point cloud initially is placed at the origin of the camera frame of reference. In the other case, the source point cloud initially is placed at its final position in the previous time step.

1.3 Content summary

Results demonstrate that stereo analysis of real world data provides relative position estimates comparable to estimates produced in the simulation domain. In particular, real world data linear regression models for error vector magnitudes predict simulation domain accuracies reported at close range in other work [42].

Additionally, results indicate that real time execution at rates between 10 Hz and 30 Hz are feasible. Block matching on the GPU exhibits the fastest execution times. Block matching on the CPU also executes at a rate appropriate for real time position estimation. Semi-global block matching on the CPU is the least suitable algorithm for real time execution of the three stereo correspondence functions tested.

Despite constricted execution times, no major impact on position estimation accuracies are observed. Linear regression models suggest that semi-global block matching on the CPU produces relative position estimates with the most noise, and block matching on the GPU produces relative position estimates with the least noise. However, these characteristics could be specific to the stereo correspondence parameters, which were held constant across all three algorithms.

Finally, seeding the iterative closest point algorithm with the solution from the previous time step results in a nominal reduction in error. The relative position estimate error vector magnitudes produced by the unseeded iterative closest point algorithm are larger on average than the magnitudes produced by the seeded iterative closest point algorithm.

II. Background

2.1 Previous work

Researchers have been investigating the automated aerial refueling (AAR) problem for over a decade [34]. The global positioning system (GPS) is a key resource in these investigations. In practice, differential GPS (DGPS) produces relative position estimation between aircraft with accuracies within tens of centimeters compared to truth data [5]. In simulation, sensor fusion techniques combining GPS data with monocular electro-optical (EO) sensor data estimate the pose of refueling aircraft with errors on the order of centimeters [29]. Because GPS service can be interrupted, researchers continue to search for additional solutions to the AAR problem. For example, sensor fusion between inertial measurement unit (IMU) data and EO sensor data reduces IMU navigation error in GPS-deprived environments [41]. Researchers have investigated medium wave infrared (MWIR), light detection and ranging (LiDaR), and stereo EO as solutions to the AAR problem as well [10, 39, 42].

Differential GPS.

Differential GPS falls into three major categories [32]. Local area differential GPS (LADGPS) depends on a single reference station with a well-defined location. This station broadcasts a scalar correction to nearby receivers. Subscribers within 1000 km that receive the correction within 10 s can achieve position accuracies within 1 m to 10 m [32]. Note that these accuracies assume service degradation due to selective availability (SA). Some LADGPS implementations leverage "pseudosatellites," shortened to "pseudolites" (PL), as ground-based GPS sources in order to improve the accuracy of subscribers' positions [32].

Wide area differential GPS (WADGPS) utilizes a network of reference stations with well-defined locations. Together, these stations calculate a correction vector for incoming GPS signals. WADGPS has a greater range than LADGPS. Additionally, WADGPS can produce greater location accuracy for receivers compared to location accuracy under LADGPS. As with LADGPS, the greater the time between a correction's broadcast and its reception, the less the resulting location accuracy improves [32].

Carrier-phase differential GPS (CDGPS) offers the best location accuracy compared to LADGPS and WADGPS location accuracies. CDGPS compares the phase of a GPS signal to the phase of a signal broadcast by a reference site. Under this system, receiver locations can be determined at the scale of centimeters [32].

Relative navigation studies have used GPS and CDGPS for position estimation. For example, NASA reported the real-world accuracy of three relative position estimation techniques using GPS, CDGPS, and the fusion of CDGPS data with inertial navigation sensor (INS) data [5].

The first technique, an independent separation measurement system (ISMS), uses locations independently reported by GPS receivers on aircraft flying in formation. These aircraft communicate their respective GPS locations at 2 Hz intervals over a radio modem. The reported locations are translated from latitude, longitude, altitude (LLA) coordinates to Earth-centered, Earth-fixed (ECEF) coordinates. Ultimately, the ISMS returns the magnitude of the separation vector between the reported locations [5].

Errors in this method may arise if the two GPS receivers do not receive signals from the same set of satellites. Latency increases error in real-time scenarios, particularly when the relative acceleration between aircraft is high. Tests suggest that this technique estimates the magnitude of the separation vector between aircraft with less

than 61 cm of error 95% of the time, ideally. However, real-time tests exhibit higher levels of error [5].

The second technique, labelled "formation needles" (FN), makes use of a reference ground location. Using the GPS data reported by the aircraft's receivers, the technique first calculates their north, east, down (NED) positions with respect to the reference location. These positions are rotated into the flight's formation frame using heading information. Finally, the aircraft's pose and velocity information is extrapolated into the future in order to reduce the impact of latency [5].

The FN technique calculates the difference between an aircraft's intended formation pose and its estimated pose and returns a difference vector. Unlike the ISMS technique, the FN technique only reports results when both GPS receivers are working from the same set of satellites. Real-time tests of the FN technique produce performance comparable to the ideal performance of the ISMS technique [5].

The third technique, a formation flight instrumentation system (FFIS), does not depend on reference stations. The FFIS uses CDGPS to estimate relative position. It leverages both GPS and INS data in order to estimate the relative orientation and velocity between aircraft. As with the ISMS and FN techniques, the FFIS relies on wireless communication between aircraft. Tests indicate that the FFIS estimates relative position within 7 ± 13 cm. However, the FFIS is not effective at estimating relative orientation in these tests [43].

Simulated monocular machine vision.

As exhibited by the FFIS, combining GPS and INS data can improve relative position estimation compared to GPS-only techniques. Other investigations have explored combining additional sensor data, such as monocular EO data, with GPS and inertial data [29]. Monocular EO machine vision uses visible spectrum data

captured by a single camera in order to determine information about the environment. Researchers have used feature matching and template matching with monocular EO sensor data as approaches when investigating solutions to the AAR problem [8, 17].

A collaboration between the University of Perugia and West Virginia University focused specifically on the automated aerial refueling of unmanned aerial vehicles (UAVs) with the aid of monocular machine vision. The group constructed a simulated environment consisting of a refueling tanker aircraft with the United States Air Force (USAF) refueling boom and an approaching drone aircraft. The environment simulates various sensor data and physical effects, including monocular EO camera data from the perspective of the approaching UAV viewing the refueling tanker [17].

The simulation environment aims to study the three-dimensional position and orientation estimation problem, i.e. the pose estimation problem. Once the pose estimation problem is solved, an approaching UAV could make corrections to its estimated position so as to place itself in a target area relative to the tanker. It is assumed that the boom operator can proceed with refueling once the UAV is inside this target area [17].

In order to solve the pose estimation problem, red markers are placed on the simulated refueling tanker. The monocular EO data, which is produced from the perspective of the UAV looking towards the refueling tanker, is simulated as a 1280×1280 pixel bitmap refreshed at 20 Hz. After an image frame is collected, a three stage process begins [17].

First, the process extracts features from the image. In this case, the features of interest are the red markers on the simulated refueling tanker. Since these markers are the only sources of red in the simulation, a red pass-through filter is applied to the image. The feature is accepted so long as its area exceeds a predetermined amount. The location of the feature is taken to be the centroid of the pixel area [17].

Second, the process matches the resulting features to expected features. Initially, the actual positions of these expected features is assumed to be available. The matching function between observed features and known features minimizes the offset distance sum between the sets. Finally, the pose is estimated according to the reported matching [17].

The initial results from this simulation study are less accurate than the previous, real-world experiments focused around GPS. While the magnitude of the separation vector between the approaching UAV and the refueling tanker is between 10 and 30 m, the resulting error in relative position estimates range from 0 m to 10 m. However, further work on this process improves the pose estimation dramatically [17].

For example, the researchers removed the red markers and replaced the feature extraction process with a corner detection algorithm. Again assuming that the actual pose of the corners are known, the monocular EO data is passed through a Harris corner detector in order to identify points on the refueling tanker [29]. A smallest univalue segment assimilating nucleus (SUSAN) corner detector was also investigated. However, the Harris corner detector produces better results [16].

In order to further improve accuracy, the position of the UAV is approximated as the previously determined position. The orientation is approximated as the rotation between the orientations reported by the aircrafts' IMUs. By seeding the pose estimation algorithm with GPS and IMU data, the error reduces to a range from -0.015 m to 0.010 m [29]. Changing the sensor fusion technique from linear interpolation to an extended Kalman filter (EKF) improves results by another order of magnitude, reducing the average error to millimeters in simulation [27].

Two other point matching algorithms considered are the mutual nearest point (MNP) and maximum clique detection (MCD) algorithms. The MNP algorithm minimizes the "distance" across four dimensions: the two pixel coordinate positions,

the feature area, and hue. Before minimization, each dimension is weighted. The researchers found that weighting the two pixel coordinates equally and weighting the remaining dimensions at zero (i.e. omitting them) produced the best results [28].

The MCD algorithm cannot be executed in real-time without applying a heuristic. However, the heuristic allows non-optimal solutions. As a result, while the MCD algorithm improves over the MNP algorithm in some cases, the MNP algorithm performs better on average. Because the MNP algorithm requires less computation than the MCD algorithm and performs better on average, it is preferable over the MCD algorithm [28].

Real world machine vision.

Another investigation into automated aerial refueling for both manned and unmanned aircraft uses real world monocular EO data. In this experiment, an approaching aircraft contains a forward facing EO camera, an embedded GPS inertial (EGI) sensor, and other sources of navigation data. The approaching aircraft images the refueling tanker. Using a three dimensional scan of the tanker and the monocular image source, the relative position vector between the two aircraft is calculated [8].

In particular, both the real world image and a simulated image, which is created as a two dimensional projection of the refueling tanker model, are passed through a Sobel edge filter. Filtering the images reduces noise due to environmental factors. The refueling tanker model is perturbed in space under a sum squared difference (SSD) metric so that its two dimensional projection converges to the camera image. At this point, the relative position of the refueling tanker model is taken to be the position of the actual tanker [8].

This approach has some challenges. Decreasing the resolution of the tanker model's iterative perturbations decreases execution time but increases the likelihood

of erroneous solutions. Ultimately, the pose of the approaching aircraft relative to the refueling tanker can be determined within 35 cm 95% of the time while the aircraft are within 20 m of each other [8].

Machine vision, in combination with other sensor sources, has even been shown to provide navigation capabilities in the absence of GPS. In one approach, machine vision data complements inertial sensor data through fusion via an extended Kalman filter. As sensor data becomes available, image information predicts incoming inertial information and corrects detected errors. Likewise, inertial information predicts incoming image information and corrects detected errors. The resulting navigation information exhibits improvement over inertial navigation alone [41].

Using medium wave infrared to aid navigation.

Because machine vision has been shown to complement existing navigation methods, researchers have explored other sensors as sources of navigation data. Tharp proposed that medium wave infrared (MWIR) sensor data could aid in navigation tasks. This investigation also focused on navigation in GPS-deprived environments, which is of interest to the AAR problem [39].

As previously described, one navigation solution uses GPS data and inertial data to produce pose estimations. However, inertial sensor systems tend to exhibit large and rapid increases in error in the absence of GPS. By combining inertial data and EO data in order to produce position estimates using structure from motion (SfM) [36], these IMU data errors were limited successfully. As a result, EO data can be used to correct errors in IMU data in GPS-denied environments [39].

From these results, Tharp suggests that MWIR imagery could fulfill a similar function. First, EO data corrects IMU errors successfully. Second, SfM using MWIR imagery returns qualitatively similar results to SfM results from EO imagery. Thus,

it is possible that MWIR imagery could correct IMU errors. No quantitative testing of this proposal could be performed due to a lack of timing information in the MWIR dataset [39].

Using medium wave infrared to aid navigation provides multiple benefits over EO imagery. For example, MWIR can provide meaningful data in environments where EO cannot provide useful sensor information. These environments include night, fog, clouds, smoke, and so on. However, MWIR does have limitations. The equipment tends to have a higher price while simultaneously offering lower resolution than EO solutions [39].

LiDaR.

Light Detection and Ranging (LiDaR) has been investigated as a supplement to existing GPS and monocular vision AAR approaches. In this system, a LiDaR sensor is placed on an aircraft approaching a refueling tanker. The resulting LiDaR sensor data is used to estimate the position of the refueling tanker relative to the approaching aircraft. This method was also tested using real world data [10].

First, the relative orientation of the aircraft is determined using INS data. Next, two separate algorithms were investigated to provide the relative position estimate. One algorithm perturbs a previously determined location of the refueling tanker into the future, producing several candidate positions. The algorithm then produces simulated LiDaR data for each of these possible positions. When actual LiDaR data becomes available, each of the predicted tanker positions are matched with the measured LiDaR data. The simulated position most closely matching the actual data is selected as the position estimate [10].

The other algorithm provides relative position estimates through iterative closest points (ICP). In this scheme, a three dimensional model of the refueling tanker is

converted into a point cloud. ICP then iteratively translates the point cloud model to fit the real-world points measured by the LiDaR. Upon convergence or a timeout, the position of the point cloud model is selected as the position estimate [10].

Testing revealed that ICP produces more accurate position estimates than the other approach. After post-processing the real world data, estimated positions were within 40 cm of the truth data on average [10].

Simulated stereo machine vision for AAR.

Stereo machine vision offers the possibility of solving the pose estimation problem for AAR without the aid of GPS or other sensor data. Stereo imagery from two cameras provides depth information that monocular imagery, which is composed of image data from a single camera, lacks. With proper calibration, a stereo pair of cameras may be used to reconstruct an imaged scene in three dimensions [42].

Werner applied stereo machine vision techniques toward solving the AAR problem [42]. The KC-46 refueling tanker will employ a stereo imaging system during refueling operations. As a result, the approach proposed by Werner shifted the pose estimation problem to the perspective of the refueling tanker observing approaching aircraft [42].

As the tanker captures stereo imagery of the approaching aircraft, it produces disparity maps from image pairs. These maps are projected into three dimensions in order to create a point cloud of the approaching aircraft. Using ICP, a three dimensional model of the aircraft is matched to the point cloud. Upon convergence or when a time limit is reached, the pose of the model is selected as the solution to the pose estimation problem [42].

Werner concludes that solving the pose estimation problem for AAR with stereo imagery is feasible. Additionally, the accuracy of the resulting pose estimation is com-

parable to other AAR approaches. Position estimates from simulations are accurate to ± 10 cm when in the contact position.

However, investigation of the stereo machine vision approach is not complete. As shown in previous studies [5], real world deployment can result in significantly degraded accuracies. Further, the execution time of stereo machine vision solutions must be small enough to allow real time applications. Currently, this stereo analysis process requires approximately 1.367 s in execution time per image pair [42].

Due to these limitations, there are several open questions that remain regarding stereo machine vision as a solution to the pose estimation problem. For example, cameras can capture 30 or more frames per second. As a result, real time approaches require execution times on the order of fractions of a second. Other real world factors may have an impact as well. In order to fully evaluate stereo machine vision as a solution to the pose estimation problem, real world data and real time execution are required. These factors compose the main thrust of this investigation.

2.2 Stereo machine vision

A key utility of stereo machine vision is the reconstruction of a three-dimensional (3D) scene from a pair of camera images. This process is based upon idealized mathematical camera models.

An idealized camera may be described mathematically as a centre of projection C' and an image plane π'. A centre of projection C' is a point in 3D space, taken to be at the center of the camera lens. An image plane π' is a two-dimensional (2D) plane in 3D space containing images of points from the 3D space. For a point P in 3D space, its image P' is defined as the intersection of the ray $\overrightarrow{PC'}$ and the image plane π' [19].

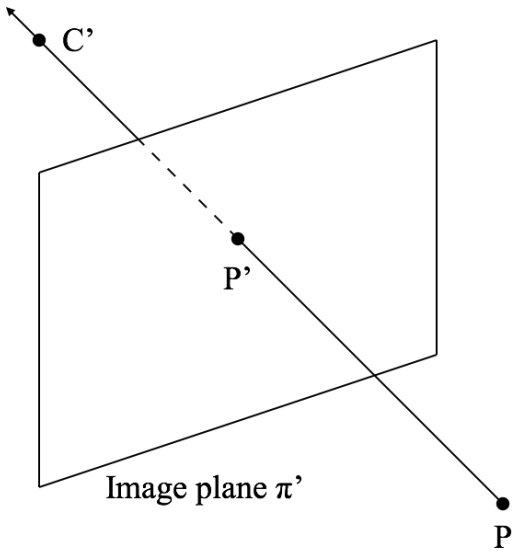


Figure 1. A simple mathematical model of a camera

Triangulation.

By introducing a second idealized camera, the position of a point P in 3D space can be determined from its images P' and P''. Specifically, point P is located at the intersection of the ray $\overrightarrow{C'P'}$ and the ray from $\overrightarrow{C''P''}$. This process is referred to as "triangulation" [11].

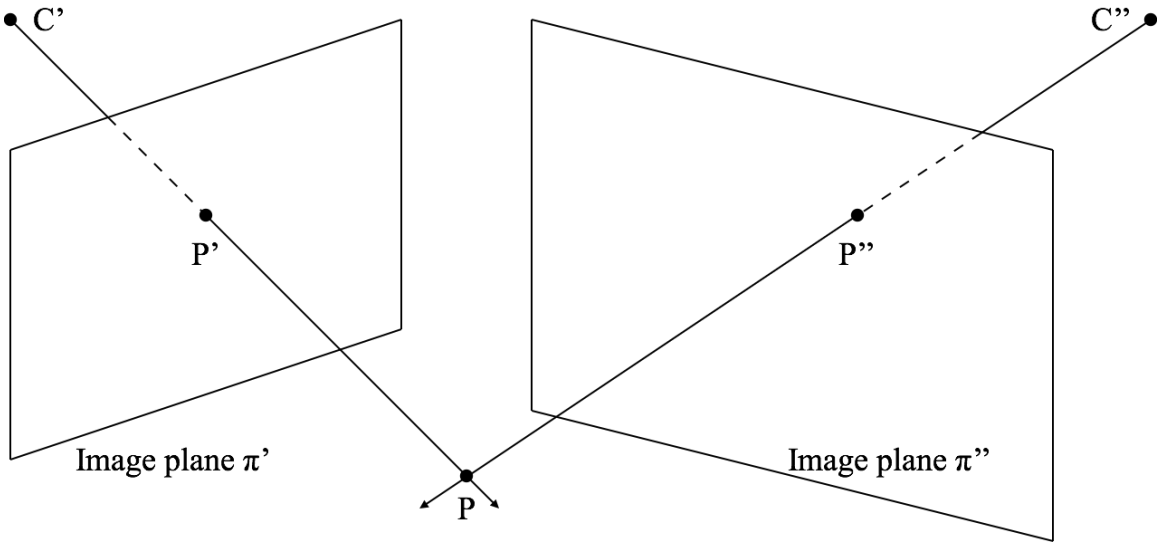


Figure 2. An illustration of ideal triangulation

However, these rays may not intersect when using data from real world cameras. For example, the image plane of a real world camera may not be continuous. When a digital camera records the image of a point P, the resulting P' may be perturbed from its geometric position due to the presence of discrete pixels. As a result, triangulation could fail to relocate the 3D point P [11].

One solution to this problem approximates P as the midpoint of the line segment perpendicular to both ray $\overrightarrow{C'P'}$ and ray $\overrightarrow{C''P''}$ [11]. This solution does not work in some situations, and in other situations this method can make poor approximations [21]. A triangulation method presented by Hartley and Sturm provides better results than the midpoint method [21].

This method leverages the epipolar geometry of the two cameras [21]. Given two image planes, the associated epipolar geometry depends on their relative pose. This geometry is summarized in a fundamental matrix $\mathbf{F}$ such that $\vec{p}^{\top}\mathbf{F}\vec{p}' = 0$, where $\vec{p}$ and $\vec{p}'$ are vectors representing the positions of image points P and P' [19]. In a pair of image planes with image points perturbed by noise, two corresponding image points P and P' generally do not satisfy the constraint $\vec{p}^{\top}\mathbf{F}\vec{p}' = 0$. However, perturbing both image points in order to eliminate the noise should result in points $\hat{\vec{p}}$ and $\hat{\vec{p}}'$ that do satisfy the constraint [21].

Hartley and Sturm assume a Gaussian noise function for their study. Their triangulation algorithm, proposed as a replacement of the midpoint method, minimizes the function $d(\vec{p}, \hat{\vec{p}})^2 + d(\vec{p}', \hat{\vec{p}}')^2$ with the constraint $\hat{\vec{p}}^{\top}\mathbf{F}\hat{\vec{p}}' = 0$. The function $d(\vec{x}, \vec{y})$ returns the Euclidean distance between $\vec{x}$ and $\vec{y}$. The solution is analytical, and the researchers report moderate execution times and consistently better results compared to other triangulation approaches [21].

Correspondence.

Triangulation depends on the key assumption that corresponding image points P' and P'' are already known. Generally, the bijection between image points in two image planes generated from cameras is not known. Before performing triangulation, this bijection must be constructed.

Stereo algorithms produce two main categories of bijections. Sparse or feature-based bijections describe correspondences between a subset of images points [35]. These algorithms, which identify image features and attempt to construct a bijection between this subset of image points, are suitable for resource-constrained environments [12]. Dense bijections attempt to describe correspondences between all image points [35]. Image point pairs with correspondence certainties below some threshold are rejected. A study investigating real time stereo correlation focuses on this second approach [12].

In developing a bijection, the researchers utilize the epipolar constraint between image planes [12]. The epipolar constraint observes that a point P, its images P' and P'', and the centre of projection points C' and C'' must be coplanar. Thus, by defining a plane $P'C'C''$, the image point P'' must lie on the epipolar line at the intersection of image plane π'' and plane $P'C'C''$ [19].

In reality, P'' is likely to have been perturbed off of this epipolar line due to noise or other factors. Thus, given an image point P', the researchers perform a search for the corresponding point P'' within a window around the epipolar line. Specifically, a window of pixels around P' is compared to a window of pixels centered on the epipolar line in the other image. A correlation score is generated according to a set of criteria, and then the window is shifted along the epipolar line. The process is repeated until a correlation curve has been generated for the entire epipolar line, and a correspondence may be selected from this curve [12].

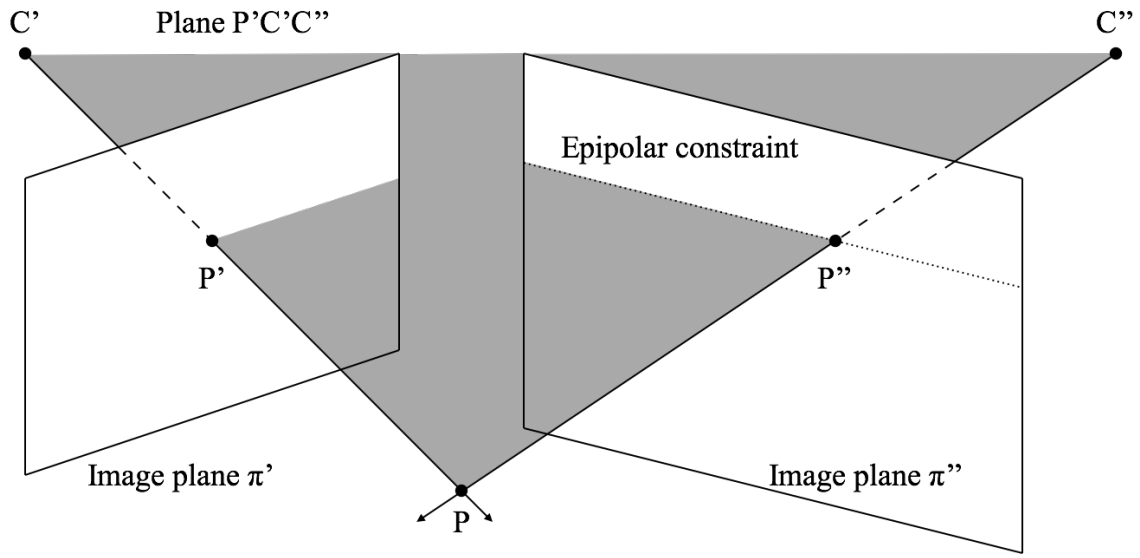


Figure 3. An illustration of the epipolar constraint

Characteristics of dense stereo algorithms.

Many other dense stereo correspondence algorithms exist. In order to better organize and compare these algorithms, researchers have outlined four characteristics common to these algorithms. Dense stereo correspondence algorithms tend to compute matching costs, aggregate costs, compute a disparity map, and refine the disparity map. Some algorithms only perform a subset of these tasks [35].

Two matching cost functions leveraged by some stereo algorithms include the sum of squared differences in pixel intensity and the sum of absolute differences in pixel intensity [35]. However, these approaches have been shown to produce incorrect matches when performed on individual pixels. Errors due to image sampling can result in large matching costs for corresponding image points. The problem becomes particularly significant when an algorithm accepts or rejects a match based on threshold values. Another pixel-based cost function eliminates sensitivity to image sampling by linear interpolation of surrounding pixel intensities. Applying this

correction results in a less than 10% increase in execution time while improving the performance of stereo algorithms on real world data [6].

After computing a matching cost function, local or window-based stereo algorithms generally conduct cost aggregation. Cost aggregation serves as an intermediary step between matching cost computation and disparity map generation for local stereo algorithms, which determine disparities based on the information within limited pixel areas. In contrast, global stereo algorithms generally determine disparities based on a global energy minimization function. As a result, global stereo functions are less likely to conduct cost aggregation [6].

In a local stereo algorithm, the costs from the matching functions may be summed or averaged over some pixel area. This aggregation of cost effectively creates an inherent smoothness constraint on disparities, as opposed to explicit smoothness constraints made by some global stereo functions. Several other cost aggregation strategies exist for local stereo algorithms as well [6].

The key output of a stereo algorithm is a disparity map. A disparity map may be defined as a function $d(x, y)$ that outputs a scalar value for each pixel coordinate of a reference image in a stereo pair. Disparity values may be conceptualized as inverse depth. Alternatively, a disparity map may be described as a 3D projective transform of a 3D scene [6].

For local stereo algorithms, the map is generated by selecting the disparity corresponding to the lowest cost at each pixel. Global stereo algorithms perform comparatively more work in order to generate a disparity map, because these algorithms have not conducted cost aggregation. Instead, some of these algorithms minimize energy as a function of the disparity map. These energy functions may depend on how well the disparity map conforms to the original images and how well neighboring pixels

adhere to a smoothness constraint. A local minimum of the energy function may be found through simulated annealing, max-flow, graph-cut, or other techniques [6].

Depending on the application, the resulting disparity map may be refined. For example, disparity values tend to be partitioned into discrete values. Applications aimed at 3D reconstruction may produce subprime output if these discrete values are not refined. Instead of a smooth reconstruction of a 3D scene, the output may be segmented into discrete layers. Sub-pixel disparity interpolation can alleviate this issue. However, such refinement may not be appropriate under certain conditions [6].

In addition to having execution steps in common, dense stereo algorithms also share some problems. For example, smoothness constraints can reduce the accuracy of disparity maps at object boundaries and for thin objects. Foreground objects have been observed to include parts of the background at their boundaries. Occlusion, in which part of a scene is not visible to one or more images, also presents challenges for stereo algorithms. Problems have also been observed in textureless areas of images, with global algorithms tending to perform better than local algorithms in these cases [6].

Failure to determine matches in low-texture images is referred to as "dropout." To mitigate dropout, one researcher demonstrated overlaying a random texture onto stereo images. However, these textures must be random in particular ways in order to achieve the best results. Some random overlays are too uniform and, as a result, different parts of the image are still too similar to prevent dropout [24].

In order to best reduce dropout, these texture overlays must be as different as possible in different parts of the image. Two different methods are presented in order to achieve this effect. The first method is inspired by coding theory techniques that encode characters of the alphabet as differently as possible for transmission. This method aims to maximize the Hamming distance between blocks. However, because

the general form of this problem is not easily solvable, a greedy algorithm is used to produce lexicographic codes [24].

The second method begins with a randomly generated overlay and modifies it using simulated annealing. The algorithm repeatedly selects pixels according to a cost function and swaps them. After performing simulated annealing multiple times, the best overlay is selected. This second method exhibits better results than the first, especially as image size increases [24].

Calibration and rectification.

Just as triangulation assumes a bijection, some stereo correspondence algorithms assume that image pairs are subject to the epipolar constraint. However, images captured with real world cameras generally do not satisfy the epipolar constraint. For example, an imperfect camera lens results in distorted images. Before some stereo correspondence algorithms produce a bijection, the cameras must be calibrated and the resulting images must be rectified.

About twenty years ago, camera calibration techniques generally fell into one of two categories. In photogrammetric calibration, a camera observes an object with precisely known 3D geometry. The object does not need to be complex; two perpendicular planes are sufficient. However, proper calibration using this technique requires very precise conditions [46].

In self-calibration, a camera observes a static scene from different locations. This technique does not require conditions as precise as the conditions required for photogrammetric calibration. However, self-calibration tends to produce less reliable results. Neither photogrammetric calibration nor self-calibration could be described as particularly robust, flexible, or inexpensive [46].

A newer calibration technique occupies a middle ground between photogrammetric calibration and self-calibration. In this technique, a patterned board is imaged by a camera from multiple poses. Because a known, 2D pattern is being used for calibration, the approach is similar to photogrammetric calibration. Because either the board or the camera is moved to different poses, the approach is similar to self-calibration [46].

Ultimately, any calibration method must produce a matrix describing the intrinsic parameters of a camera. These parameters are organized into an intrinsic parameters matrix $\mathbf{A}$, where $\mathbf{A}$ has the following form [46].

$$\mathbf{A} = \begin{pmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

In this matrix, (u_0, v_0) is the principal point of the image (ideally in the center), α and β are scale factors associated with the u and v axis respectively, and γ describes skew in the image axes [46].

Once the intrinsic parameters are determined, the extrinsic parameters (rotation and translation) of the camera can also be determined. These parameters take the form of a 3×3 rotation matrix $\mathbf{R}$ and a three-dimensional translation column vector $\vec{t}$, often combined into a 3×4 matrix written as $\begin{pmatrix} \mathbf{R}\vec{t} \end{pmatrix}$ [46].

The relationship between a 3D point in the world and the corresponding point in an image plane may be summarized as follows [46].

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{A} \begin{pmatrix} \vec{r_1} & \vec{r_2} & \vec{r_3} & \vec{t} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{2}$$

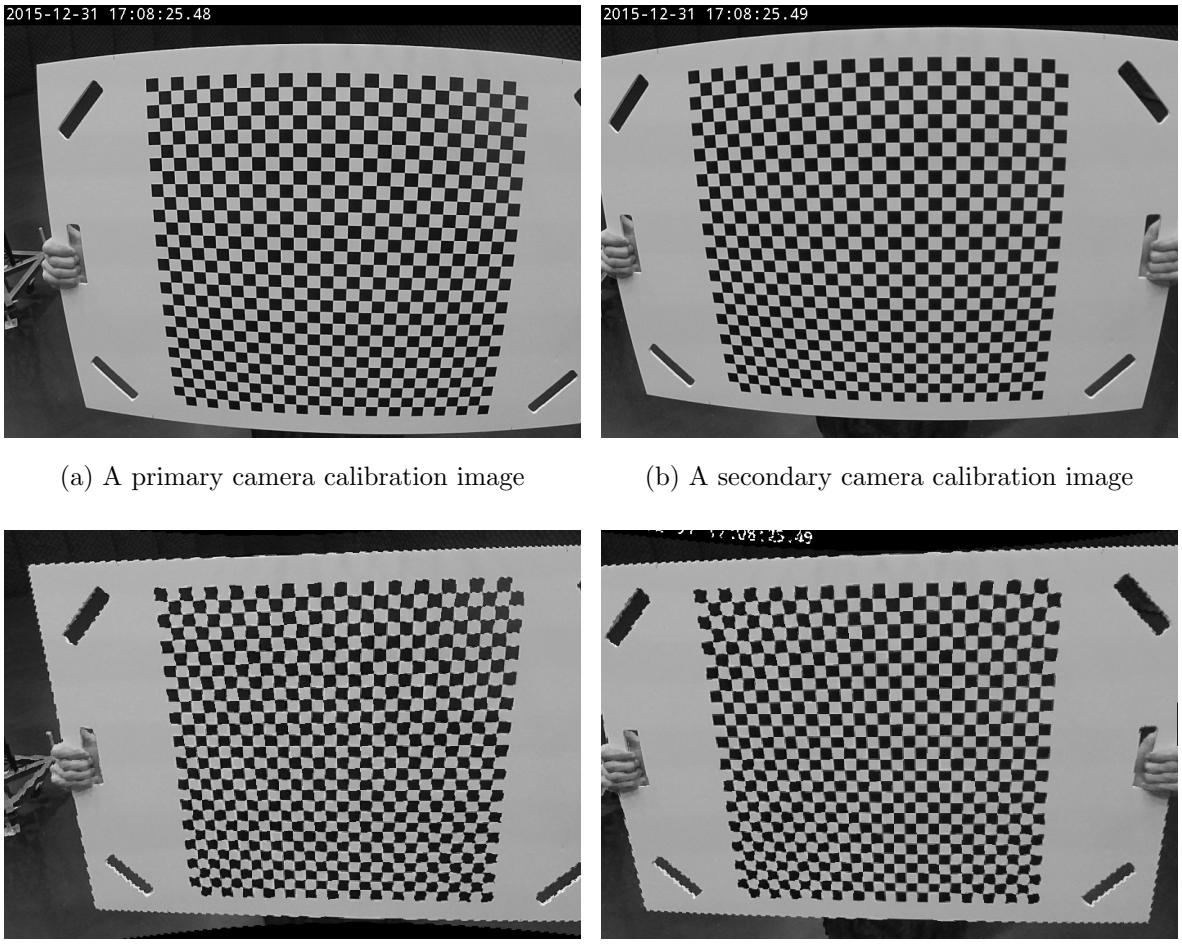
This calibration method involves a camera's view of a known pattern that lies within a 2D plane. Thus, without loss of generality, the observed pattern plane can be assumed to be coplanar with the z-plane in the world coordinate system. This assumption simplifies the relationship between a 3D point observed on the pattern plane and the corresponding 2D point on the image plane [46].

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{A} \begin{pmatrix} \vec{r_1} & \vec{r_2} & \vec{t} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3}$$

As a shorthand, let $\mathbf{H} = \mathbf{A} \begin{pmatrix} \vec{r_1} & \vec{r_2} & \vec{t} \end{pmatrix}$. Given an image of the pattern plane, $\mathbf{H}$ may be estimated using a maximum likelihood criterion and assuming that the image coordinates (u, v) have been perturbed by Gaussian noise with a mean of zero. Using the fact that $\vec{r_1}$ and $\vec{r_2}$ should be orthonormal, additional constraints may be placed on the intrinsic parameters [46].

Using this calibration technique, it can be shown mathematically that an additional image parallel to an existing image in the world coordinate system provides no new information. In simulation, using three images for calibration produces a relatively large increase in performance compared to using two images. Using more than three images produces a relatively smaller increase in performance. Rotations of the pattern plane of 45° from image to image produces the best performance [46].

After calibration, stereo image pairs must be rectified. In this step, calibration information is used to remap the images so that they adhere to the epipolar constraint. Note that it is possible to perform rectification without calibration [20].



(a) A primary camera calibration image

(b) A secondary camera calibration image

(c) A rectified primary camera image

(d) A rectified secondary camera image

Figure 4. Example stereo calibration images and resulting rectified images

Stereo algorithms in OpenCV.

OpenCV 3.0.0 offers five stereo algorithms. Two of these algorithms, a block matching algorithm and a semi-global matching algorithm, utilize only the CPU for analysis. The remaining three algorithms utilize CUDA and Nvidia GPU processing for analysis. These algorithms include a block matching algorithm, a belief propagation algorithm, and a constant space belief propagation algorithm [31].

CPU block matching.

The OpenCV CPU block matching algorithm, `StereoBM`, is based on a block matching algorithm from Konolige [7, 25]. At a high level, the algorithm expects to compute a disparity map given two rectified, stereo images. Because the images are rectified, the search space for a given pixel's match in the other image is reduced to only the pixels near the epipolar constraint. Additionally, the algorithm is able to make the assumption that, for a given block in the primary image, the matching block in the secondary image has x-coordinates less than or equal to the x-coordinates of the primary block [7].

The `StereoBM` algorithm accepts up to two parameters. The first parameter, `numDisparities`, sets the number of pixels along the epipolar constraint that are searched for matches. The algorithm begins by searching the same pixels in the secondary image as the corresponding pixels in the primary image, and continues searching to the left for `numDisparities` pixels. The default argument value is 0 [31].

The second parameter, `blockSize`, sets the side length of the square search blocks. Arguments must be odd so that search blocks are centered around the pixel of interest. As block size increases, the resulting disparity map may become more smooth but less accurate, while smaller block sizes increase the possibility of incorrect matches. The default argument value is 21 [31].

In order to produce matches, the original algorithm first applies an image transform. The image transform is designed to minimize the effects of real world artifacts and factors that may skew pixel correspondence. The laplacian of gaussian transform first blurs the image with a standard deviation of a couple pixels. Then, image edge intensities and directions are identified. After transforming the images, the algorithm matches areas of the images. An L1 norm with the sum of absolute differences serves as the correspondence function [25].

Once the algorithm has produced a disparity map by minimizing the results of correspondence, the map is filtered. One filter that reduces errors at object boundaries, which are produced due to overlapping blocks in areas with discontinuous disparities, is a left/right check. By producing a disparity map twice, alternating the primary and secondary images, disagreements that occur at object boundaries may be rejected. Another filter, an "interest operator" that measures image texture along epipolar lines, may be used to reject correspondences in areas of the image with little texture. The threshold of this operator may be set above the level of noise characteristic to the imaging device [25].

This algorithm assumes that the provided images are rectified. As a result, the epipolar constraints are straight, horizontal lines. Additionally, points at a given y-coordinate in the primary image are expected to be located at the same y-coordinate in the secondary image (with the x-coordinate being different). Experiments show that a vertical offset of even a couple of pixels between images in a pair may result in poor disparity maps [25].

CPU semi-global block matching.

The second CPU stereo algorithm, `StereoSGBM`, takes at least three or up to eleven parameters. The first parameter, `minDisparity`, identifies the starting offset value for the algorithm to search for a match. This value may even be set negative if, for some reason, the user expects matching pixels to be located in the opposite direction as expected along the epipolar constraint [31].

The second parameter, `numDisparities`, is used to indicate the difference between the specified minimum disparity and the desired maximum disparity. The `blockSize` parameter must be odd, and it specifies the side length of the search window. The next two parameters, `P1` and `P2`, are integers controlling disparity smoothness. The

`P1` argument penalizes small disparity changes between neighboring pixels, and the `P2` argument penalizes large disparity changes between neighboring pixels [31].

The next parameter, `disp12MaxDiff`, controls the left/right check. By setting this integer to a negative value, the check is disabled. Positive values govern the largest allowed difference in pixels between disparities before they are rejected. The `preFilterCap` parameter creates an interval that limits values passed to the pixel cost function. The `uniquenessRatio` parameter describes how well a selected disparity must exceed the next-closest option in order to be selected [31].

The last three parameters include `speckleWindowSize`, `speckleRange`, and `mode`. Post-processing on the disparity map checks for "speckles," or areas of disparity to label as noise. The `speckleWindowSize` parameter governs how large these noise regions are allowed to be. The `speckleRange` value governs how similar neighboring disparities are allowed to be in speckles. Finally, the `mode` parameter governs whether or not a memory-intensive version of the algorithm is executed [31].

The original algorithm generates a correspondence cost curve along epipolar constraints. The cost function can use mutual information (MI) [22]. In the case of the OpenCV implementation, a cost function insensitive to image sampling is used [6,31]. The algorithm is semi-global in that an energy function exists to be minimized, but the minimization process is approximated by conducting cost aggregation [22].

Cost aggregation solutions along epipolar lines are vulnerable to errors or artifacts in the horizontal direction. In order to mitigate this issue, the semi-global solution aggregates correspondence costs in multiple one-dimensional directions emanating from each pixel [22].

In order to further reduce errors, the global energy function tracks costs for two separate types of disparity discontinuities. A smaller cost is levied for neighboring pixels with small disparity differences, while a larger cost is levied for large disparity

differences in neighboring pixels. This scheme better allows for continuous surfaces with gradients with respect to the image plane [22].

The resulting disparity maps still exhibit some consistent issues. For example, small regions of the resulting map may have drastically different disparity values than other nearby pixels. Additionally, this semi-global matching technique may have higher error rates in low texture portions of images [22].

CUDA block matching.

The CUDA implementation of stereo block matching, `cuda::StereoBM`, follows from the CPU block matching algorithm. The algorithm accepts two parameters, `numDisparities` and `blockSize`. The `numDisparities` parameter has a default value of 64. This parameter determines the number of pixels along the epipolar constraint that are searched for matches. The `blockSize` parameter has a default value of 19. This parameter sets the side length of the square search blocks [31].

CUDA belief propagation.

Another stereo matching algorithm that utilizes CUDA for parallel processing, `cuda::StereoBeliefPropagation`, accepts four parameters. These parameters include `ndisp`, `iters`, `levels`, and `msg_type`. The computation function for producing a disparity map also accepts an array for specifying data cost [31].

The belief propagation algorithm aims to approximate optimization on Markov random field (MRF) models. This optimization is an NP hard problem, but belief propagation acts as a suitable approximation by identifying local minima over large enough neighborhoods when applied to stereo correspondence. Using MRF models for stereo correspondence compares positively to local stereo algorithms. However,

without approximation, using MRF models requires more execution time than local stereo algorithms [13].

As with other stereo algorithms, the belief propagation algorithm defines an energy function with costs for data and discontinuities. To minimize this energy function, images are represented as graphs. Messages are passed between neighboring pixels in parallel, and the process is repeated iteratively. As a consequence of this approach, belief approximation can require relatively large amounts of memory due to the presence of multiple messages per pixel [13].

To further reduce execution time, some optimizations are made. The message update process is reduced to linear time. The graph representing the image is encoded as a bipartite graph in order to reduce both execution time and memory requirements. Finally, the graph is organized into multiple levels in order to expedite moving messages between disparate portions of images [13].

CUDA constant space belief propagation.

The final stereo matching algorithm, which also utilizes CUDA for parallel processing, is the `cuda::StereoConstantSpaceBP` algorithm. This algorithm accepts five parameters, `ndisp`, `iters`, `levels`, `nr_plane`, and `msg_type` [31].

This algorithm, like the previous belief propagation algorithm, represents images as graphs with multiple levels. Also like the previous algorithm, belief propagation is performed iteratively and messages are updated in linear time. However, messages are also updated in constant space [44].

This new method performs comparably in terms of execution time and accuracy to the standard belief propagation approach while requiring less memory. However, this method does exhibit a shortcoming. At discontinuities in disparity values, the algorithm tends to perform more poorly than standard belief propagation. In order

to alleviate this issue, a bilateral filter may be applied to the function output. This filter does not require significant additional execution time or space [44].

2.3 Iterative closest point algorithm

Stereo imagery may be used to create a three dimensional reconstruction of an imaged scene. In order to use this reconstruction for relative position estimation, the resulting scene must be registered to truth models. The iterative closest point algorithm provides one method for performing this procedure.

The iterative closest point (ICP) algorithm aims to determine a transformation, which is composed of a rotation and a translation, that aligns a source (also "model" or "truth") point cloud with a target (or "data") point cloud according to an error metric. This process of alignment is referred to as registration. The mean square distance between paired points in the source and target point clouds is one example of an ICP error metric [4].

Previous work investigated using a synthetic virtual model as a source point cloud and sensor data to compose a target point cloud. Target point clouds generated from sensor data are preprocessed in order to remove erroneous points due to sensor noise. Additionally, the search range for minimizing the error metric is narrowed by using properties specific to the shapes being registered [4].

Each iteration of the algorithm consists of two main steps. First, a map is generated between points in the source point cloud and points in the target point cloud. The map may be generated according to many different criteria. For example, points in the source point cloud may be mapped to the nearest point in the target point cloud according to the Euclidean distance function. In general, a map between a source point cloud and a target point cloud is not a one-to-one correspondence [4].

After generating a map between point clouds, a transformation is applied in order to minimize the error criteria. For example, singular value decomposition (SVD) or a quaternion operation may be applied under the least squares metric. This transformation uses the previously generated map as input [4].

By repeating this process iteratively under the mean squared error metric, the iterative closest point algorithm is proven to converge to the nearest local minimum monotonically. Experimental results indicate that the largest decrease in error occurs during the first sequence of iterations. After the first sequence of iterations, errors decrease by considerably smaller amounts between iterations [4].

Iterative closest point algorithms have been applied to stereo vision data and navigation problems. In navigation scenarios, computational challenges have been reduced by utilizing motion continuity constraints. For example, this method was applied in order to determine the motion of an observer in an environment. While this example aimed to determine the position of the observer through scene reconstruction for navigation, ICP can be used for object recognition and tracking as well [45].

Iterative closest point algorithm in Point Cloud Library.

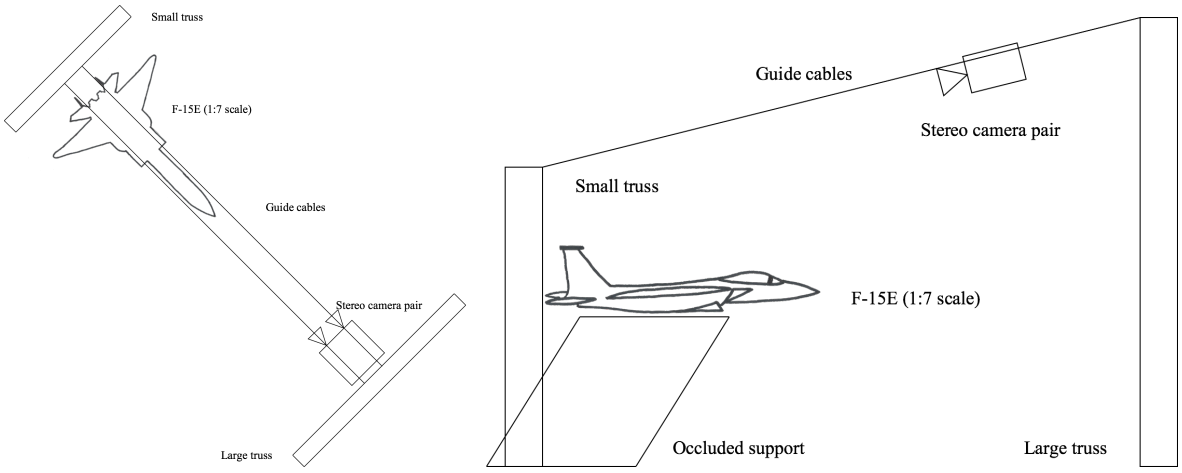
An implementation of the iterative closest point algorithm is included in Point Cloud Library 1.8.0. Users set a source point cloud and a target point cloud. Additionally, convergence criteria based on the number of iterations, the differences between subsequent iterations, or the error criteria may be set. After executing the alignment function, the ICP object may be queried to check convergence. The object may also be queried for a fitness score, which reports the average distance between a point and its nearest neighbor. A 4×4 transformation matrix is returned after alignment [33].

III. Methodology

A main thrust of this investigation focuses on evaluating position estimation accuracies from real world stereo imagery. Evaluating position estimation accuracies from real world stereo imagery requires a real world data collection. Because a full, one-to-one scale data collection is not feasible, real world stereo imagery was collected at a one-seventh scale.

3.1 Real world data collection environment

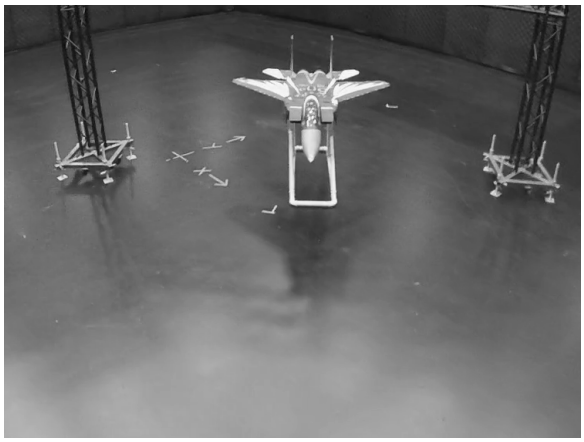
A 1:7 scale F-15E was imaged with a pair of cameras at distance vectors varying from 2 m to 8 m in magnitude. These distances correspond to distance vectors ranging from 14 m to 56 m in a full scale refueling scenario. Unlike the simulation domain, truth data is not intrinsically available in the real world domain. Truth data is collected in parallel with the collection of stereo imagery. All data collection takes place within a Vicon motion capture area. The poses of the model F15-E and each camera are tracked by the motion capture system.



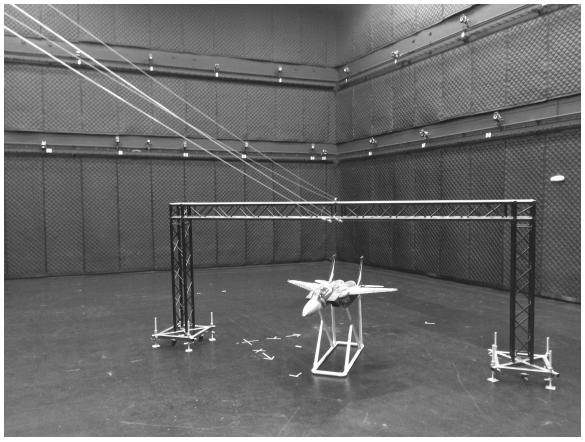
(a) Top view (b) Side view

Figure 5. Illustrations of the data collection environment (not to scale)

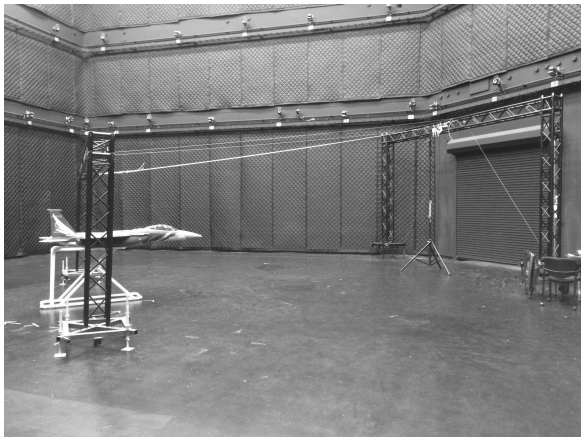
(a) The 1:7 scale F-15E



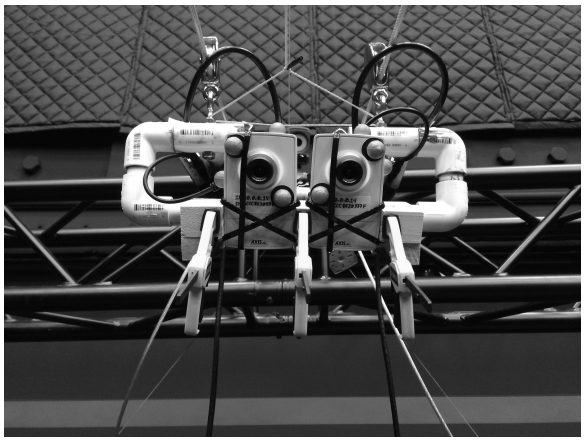
(b) A view from the primary camera



(c) A front view of the motion capture area



(d) A side view of the motion capture area



(e) The stereo camera pair

(f) A side view of the stereo camera pair

Figure 6. Images of the data collection environment

3.2 Time synchronization and network structure

In order to construct a correspondence between stereo imagery and motion capture data, precise timing is required. Specifically, clock times must be synchronized between the motion capture computer, the stereo data collection computer, and each camera. In order to achieve clock synchronization, these devices were connected to a local area network (LAN). A stratum 1 network time protocol (NTP) server provided time synchronization services to each device.

The stratum 1 NTP server provided time to the LAN using information from GPS signals. Four devices on the LAN record timestamp data. The venue motion capture data collection computer receives data from the venue motion capture data broadcast computer. Once this motion capture data is received, it is timestamped by the venue software. These timestamps have microsecond precision.

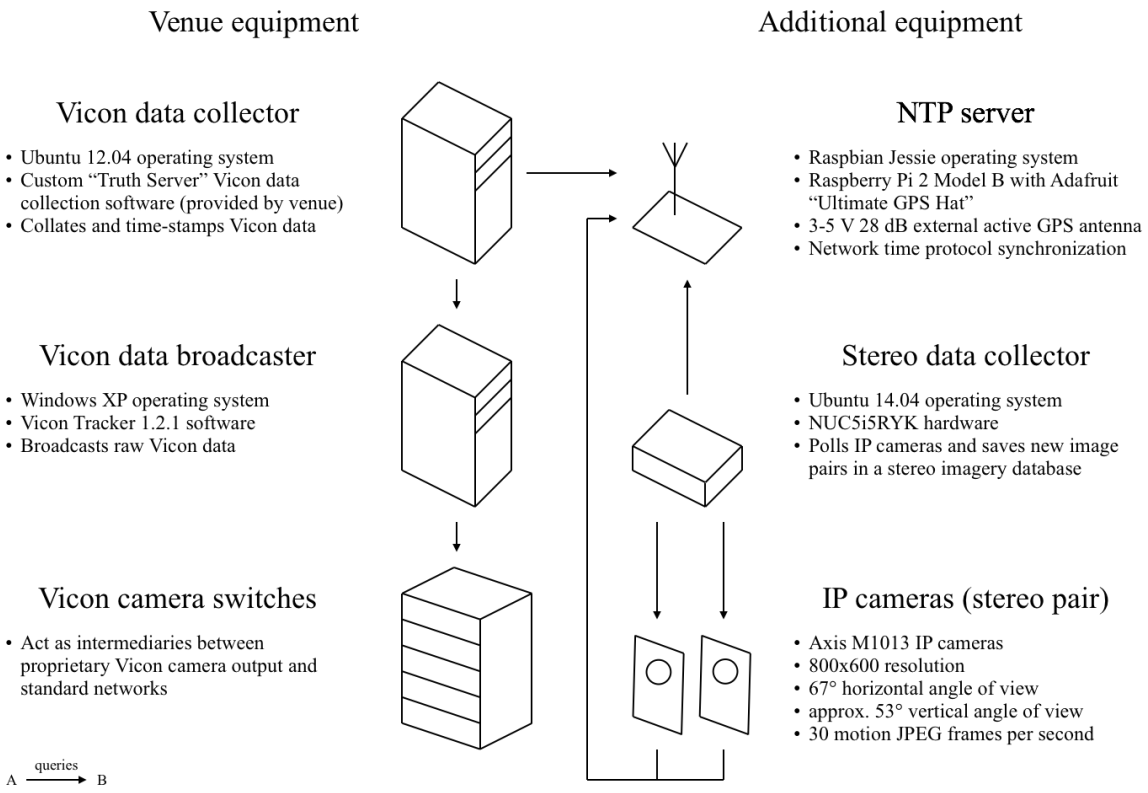


Figure 7. A chart summarizing the local area network

Each of the two cameras timestamp images according to their time sources. The camera options include a setting to use timing information from an NTP server. Timestamps are burned into the top of the images themselves, and they have precision up to one hundredth of a second.

Before data analysis can take place, the timestamps burned into the stereo image data must be extracted as text. Once the timestamps are available as text, the timestamped motion capture truth data can be programmatically paired with the timestamped stereo imagery data. The image time stamps were extracted as text using the open source Tesseract OCR (optical character recognition) package [38].

The stereo data collection computer also subscribed to the NTP server time. This computer recorded the arrival time of camera images in the database. Coupled with the image time stamps, these arrival times indicate the time between image capture and the arrival of the image at the database.

3.3 Stereo camera API and database

OpenCV provides robust, high-quality computer vision functions. However, due to its scope, OpenCV does not offer an interface for stereo cameras. Instead OpenCV offers interfaces to single video sources, and these interfaces may be combined for multi-camera systems.

In order to reduce code complexity and increase code flexibility and reusability, a stereo camera application programming interface (API) and an associated stereo imagery database were developed. The stereo imagery database schema was designed with general stereo imagery experiments in mind. Using a database to collect, organize, store, and retrieve stereo data is essential in making large sets of data feasible.

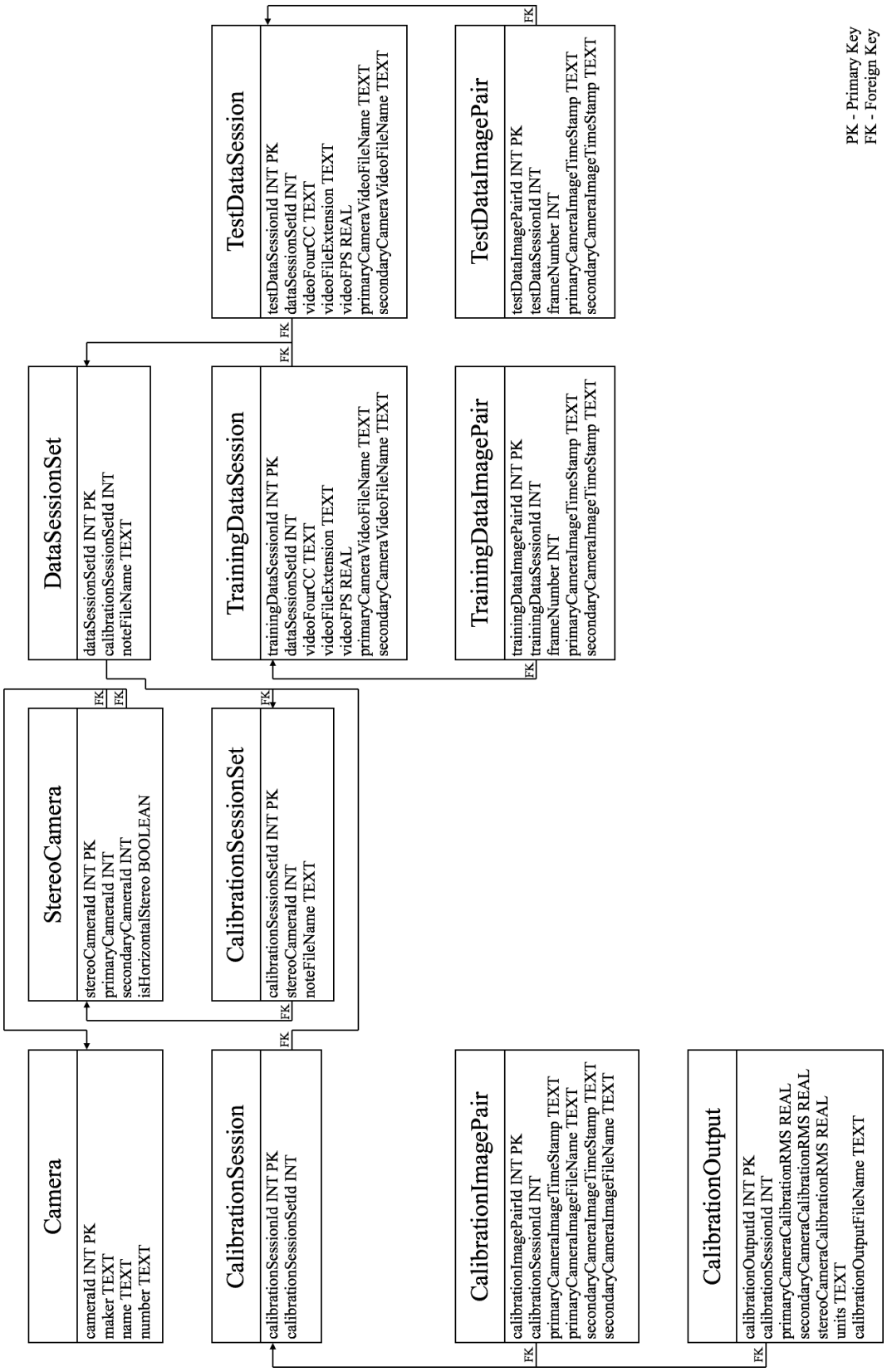


Figure 8. An illustration of the StereoCamera class database schema

Camera table and associated functions.

At least one row must be entered into the Camera table before other data may be placed in the database. When collecting real world data (as opposed to simulation domain data), at least two rows will be present in the Camera table before placing other data in the database.

Each row is composed of an automatically assigned camera ID integer primary key, a maker text field, a name text field, and a number text field. Entries in all four column cannot be null, and the (maker, name, number) tuple must be unique.

For a physical, real world camera, the maker field should contain the camera brand, the name field should contain the camera model, and the number field should contain the serial number. For a virtual camera generated by a commercial application, the maker field should contain the company, the name field should contain the application name, and the number field should contain the app version number. For a virtual camera generated by a custom program, the maker field should contain the program author, the name field should contain the git repository address, and the number field should contain the commit hash.

The only `StereoCamera` class method associated with this table is `getCameraId`. The method accepts make, name, and number arguments. If a corresponding camera exists in the table, then its ID is returned. Otherwise, the camera is entered into the table and its ID is returned.

Stereo camera table and associated functions.

The stereo camera table tracks pairs of cameras from the camera table. Each row is automatically assigned a stereo camera ID integer primary key. The primary camera ID and secondary camera ID entries are foreign keys referencing camera table IDs. The fourth entry in each row indicates whether the two cameras have a horizontal

or vertical baseline. A check on this entry restricts values to zero or one. Entries in all four columns cannot be null, and the (primary camera ID, secondary camera ID, is horizontal stereo) tuple must be unique.

The `StereoCamera` class method `getStereoCameraId` is associated with this table. The method accepts a primary camera ID, a secondary camera ID, and a `StereoType` enum value (`HORIZONTAL` or `VERTICAL`). If a corresponding stereo camera exists in the table, then its ID is returned. Otherwise, the stereo camera is entered into the table and its ID is returned.

Calibration session set table and associated functions.

Tests demonstrate that stereo camera calibration results from OpenCV can vary dramatically between two different calibration sessions. In particular, two practically identical calibration sessions performed back-to-back can result in significantly different stereo calibration output. As a result, it is important to perform stereo calibration sessions in sets.

Each calibration session results in stereo calibration output, and a calibration session set is a collection of calibration sessions. In other words, a calibration session set is a collection of stereo calibration trials (sessions). A calibration session set (i.e. one or more calibration sessions) must be performed before collecting data for analysis.

Each row is composed of an automatically assigned calibration session set ID integer primary key, a stereo camera ID foreign key referencing a stereo camera table ID, and a note file name. The calibration session set ID and the stereo camera ID entries cannot be null, and the note file name must be unique.

A note `string` object should be used to record details apparent before capturing images. Examples include experimenter names and contact information, the stereo

camera baseline measurement, and other settings. The contents of the note `string` object are written into a note file, which is tracked in the database by the note file name. The location of the note file is meant to be opaque to the user.

Four `StereoCamera` class methods interface with the calibration session set table. To insert a calibration session set row, pass a stereo camera ID and a note `string` object to the `insertCalibrationSessionSet` method. This method returns the resulting calibration session set ID. The `selectCalibrationSessionSetIds` method returns a `vector` object containing all calibration session set IDs associated with the given stereo camera ID.

The `StereoCamera` class method `selectCalibrationSessionSetNote` and the `StereoCamera` class method `writeCalibrationSessionSetNote` provide access to the calibration session set note. These functions are read-only by design. Both methods accept a calibration session set ID. The former method returns a copy of the associated note as a `string` object, and the latter method writes a copy of the note to the location indicated by the second argument. Success is indicated by a returned Boolean value.

Calibration session table and associated functions.

A calibration session is composed of stereo camera calibration images. The calibration session table acts as an organizational table between calibration session sets and calibration images. Each row is composed of an automatically assigned calibration session ID integer primary key and a calibration session set ID foreign key referencing a calibration session set table ID. All entries in this table cannot be null.

The `insertCalibrationSession` and `selectCalibrationSessionIds` methods interface with this table. The former method accepts a calibration session set ID and returns the ID of the resulting calibration session. The latter method returns a `vector`

object containing all calibration session IDs associated with the given calibration session set ID.

Calibration image pair table and associated functions.

The calibration image pair table tracks stereo camera calibration images. Each row is composed of an automatically assigned calibration image pair ID integer primary key, a calibration session ID foreign key referencing a calibration session table ID, a primary camera image time stamp, a primary camera image file name, a secondary camera image time stamp, and a secondary camera image file name. Each primary camera image file name and each secondary camera image file name must be unique. Except for the file name entries, all entries in this table cannot be null.

The `insertCalibrationImagePair` method stores an image pair previously captured by the `captureImagePair` method with the given calibration session ID. The resulting calibration image pair ID is returned. The `selectCalibrationImagePairIds` method returns a `vector` object containing all calibration image pair IDs with the given calibration session ID. The `selectCalibrationImageTimeStamp` method returns the time stamp `string` object associated with the given calibration image pair ID and `Camera` enum value (`PRIMARY` or `SECONDARY`).

The methods `selectCalibrationImage` and `writeCalibrationImage` provide access to captured calibration images. These functions are read-only by design. Both methods accept a calibration image pair ID. The former method returns a copy of the associated image as a `cv::Mat` object, and the latter method writes a copy of the image to the location indicated by the second argument. Success is indicated by a returned Boolean value.

Calibration output table and associated functions.

After completing a calibration session, the data may be analyzed for calibration output. A `string` object containing the appropriately formatted calibration output is generated by the `getCalibrationOutputYML` method. This method accepts as input a calibration session ID, a `cv::Size` object representing the number of interior corners in the imaged checkered board, the length of a checker square, and the units of the checker square length.

Each row of the calibration output table is composed of an automatically assigned calibration output ID integer primary key, a calibration session ID foreign key referencing a calibration session table ID, a primary camera calibration RMS value, a secondary camera calibration RMS value, a stereo camera calibration RMS value, the units of the calibration data, the the calibration output file name.

Other than the calibration output file name, all entries in this table cannot be null. The calibration output file name must be unique. The primary camera calibration RMS, the secondary camera calibration RMS, and the stereo camera calibration RMS must be greater than or equal to zero.

The `insertCalibrationOutput` method takes a calibration session ID and an appropriately formatted calibration output `string` object and returns a calibration output ID. The `selectCalibrationOutputIds` method returns a `vector` object containing all calibration output IDs with the given calibration session ID. The `StereoCamera` method `selectCalibrationOutputCameraRMS` returns the camera calibration RMS value associated with the given calibration output ID and `Camera` enum value (`PRIMARY` or `SECONDARY`), and the `selectCalibrationOutputStereoRMS` method returns the stereo camera calibration RMS value associated with the given calibration output ID. The `selectCalibrationOutputUnits` method returns the units of the calibration output associated with the given calibration output ID.

The methods `selectCalibrationOutput` and `writeCalibrationOutput` provide access to calibration output. These functions are read-only by design. Both methods accept a calibration output ID. The former method returns a copy of the associated calibration output as a YML-formatted `string` object, and the latter method writes a copy of the calibration output to the location indicated by the second argument. Success is indicated by a returned Boolean value.

Data session set table and associated functions.

A data session set consists of a collection of data sessions. These data sessions can be described as trials. Thus, a data session set should be composed of different trials or "runs" of the same experiment. A data session set can contain both training data sessions and test data sessions.

Each row is composed of an automatically assigned data session set ID integer primary key, a calibration session set ID foreign key referencing a calibration session set table ID, and a note file name. The data session set ID and the calibration session set ID entries cannot be null, and the note file name must be unique.

A note `string` object should be used to record details apparent before capturing images. Examples include experimenter names, contact information, and a quantitative description of the experiment associated with the data session set. The contents of the note `string` object are written into a note file, which is tracked in the database by the note file name. The location of the note file is meant to be opaque to the user.

Four `StereoCamera` class methods interface with the data session set table. To insert a data session set row, pass a calibration session set ID and a note `string` object to the `insertDataSessionSet` method. This method returns the resulting data session set ID. The `selectDataSessionSetIds` method returns a `vector` object containing data session set IDs associated with the given calibration session set ID.

The `StereoCamera` class method `selectDataSessionSetNote` and the method `writeDataSessionSetNote` provide access to the data session set note. These functions are read-only by design. Both methods accept a data session set ID. The former method returns a copy of the associated note as a `string` object, and the latter method writes a copy of the note to the location indicated by the second argument. Success is indicated by a returned Boolean value.

Data session tables and associated functions.

Two data session tables exist in the database. These data session tables include a training data session table and a test data session table. Other than the utility of the associated data sessions (training vs test), these two data session tables are identical.

A data session is composed of a stereo pair of videos. Data session tables act as organizational tables between data session sets and data image pairs. Each row is composed of an automatically assigned data session ID integer primary key, a data session set ID foreign key referencing a data session set table ID, a video four character code (fourCC), a video file extension, video frames per second (FPS), a primary camera video file name, and a secondary camera video file name.

Other than the primary camera video file name and the secondary camera video file name, entries in this table cannot be null. The primary camera video file name must be unique, as does the secondary video file name. The video FPS value must be greater than zero.

The "insert data session" and "select data session IDs" methods interface with this table. The former method accepts a data session set ID, a fourCC value, a video file extension, and a video FPS value and returns the ID of the resulting data session. The latter method returns a `vector` object containing all data session IDs associated

with the given data session set ID. Methods to select the video fourCC value, the video file extension, and the video FPS value given the data session ID exist as well.

The "select data session video" and the "write data session video" methods provide access to data session videos. These functions are read-only by design. Both methods accept a data session ID. The former method returns a copy of the associated video as a `cv::VideoCapture` object, and the latter method writes a copy of the video to the location indicated by the second argument. Success is indicated by a returned Boolean value.

Data image pair tables and associated functions.

As with the data session tables, two data image pair tables exist in the database. These data image pair tables include a training data image pair table and a test data image pair table. Other than the utility of the associated data image pairs (training vs test), these two data image pair tables are identical.

The data image pair tables may be used to track frame numbers and time stamps. Each row is composed of an automatically assigned data image pair ID integer primary key, a data session ID foreign key referencing a data session table ID, a frame number, a primary camera image time stamp, and a secondary camera image time stamp. All entries in these tables cannot be null, the (data session ID, frame number) tuple must be unique, and frame numbers must be greater than or equal to zero.

The "insert data image pair" methods store an image pair previously captured by the `captureImagePair` method to the ends of the appropriate data session videos. The resulting data image pair ID is returned. The "select data image pair IDs" methods return a `vector` containing all data image pair IDs with the given data session ID. The "select data image time stamp" methods return the time stamp `string` associated with the given calibration image pair ID and `Camera` enum value.

Figure 9. An illustration of the StereoCamera class API

The "select data image" and "write data image" methods provide access to captured data images. These functions are read-only by design. Both methods accept a data image pair ID. The former method returns a copy of the associated image as a `cv::Mat` object, and the latter method writes a copy of the image to the location indicated by the second argument. Success is indicated by a returned Boolean value.

3.4 Calibration

Each calibration session follows the same methodology. Nine pairs of calibration images are captured during each calibration session. An example set of calibration images from the perspective of the primary camera is pictured below.

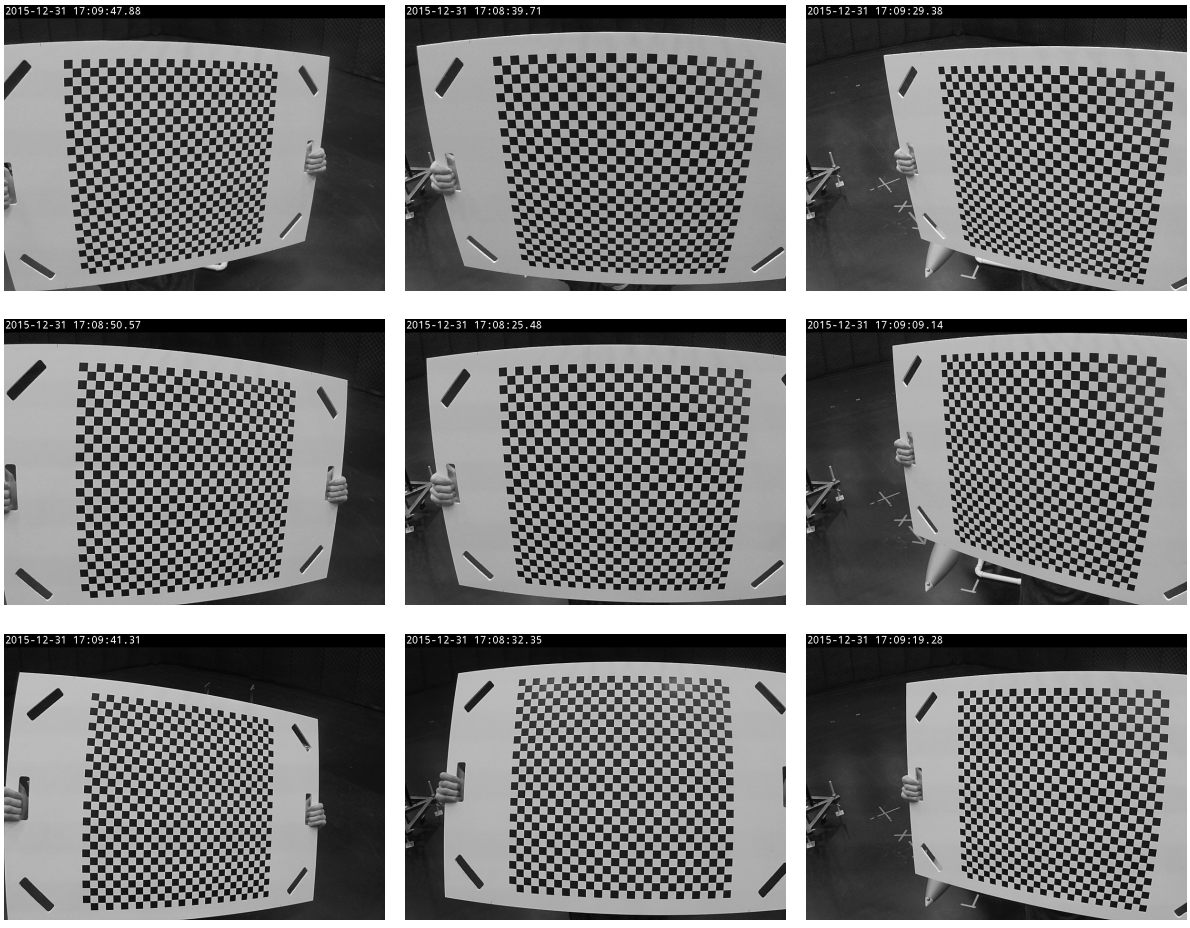


Figure 10. An example set of calibration images from the primary camera perspective

The checkered square pattern is 27 interior corners wide by 27 interior corners tall. Each square has a side length of 19 mm.

Calibration output exhibiting a stereo camera calibration RMS value of 0.306445 mm was used for the purposes of rectification. While individual camera calibration RMS values are consistently on the order of 0.30 mm, many of the observed stereo camera calibration RMS values are more than two orders of magnitude larger. Calibration outputs with these high stereo camera calibration RMS values were not selected for the purposes of rectification.

3.5 Relative position estimation process outline

In order to generate a relative position estimate, a stereo pair or rectified images with applied region of interest masks is passed through a stereo correspondence algorithm. Using these two images, the stereo correspondence algorithm generates a disparity map. This disparity map is passed through a speckle filter in order to reduce erroneous regions of disparity values. After filtering, the disparity map is projected into a three dimensional point cloud.

This data point cloud, along with a model point cloud, is passed through the iterative closest point (ICP) registration algorithm. The ICP algorithm iterates until a convergence criteria is met. After reaching a convergence criteria, the ICP algorithm outputs a transformed point cloud. The resulting transformed point cloud is used to generate a relative position estimate.

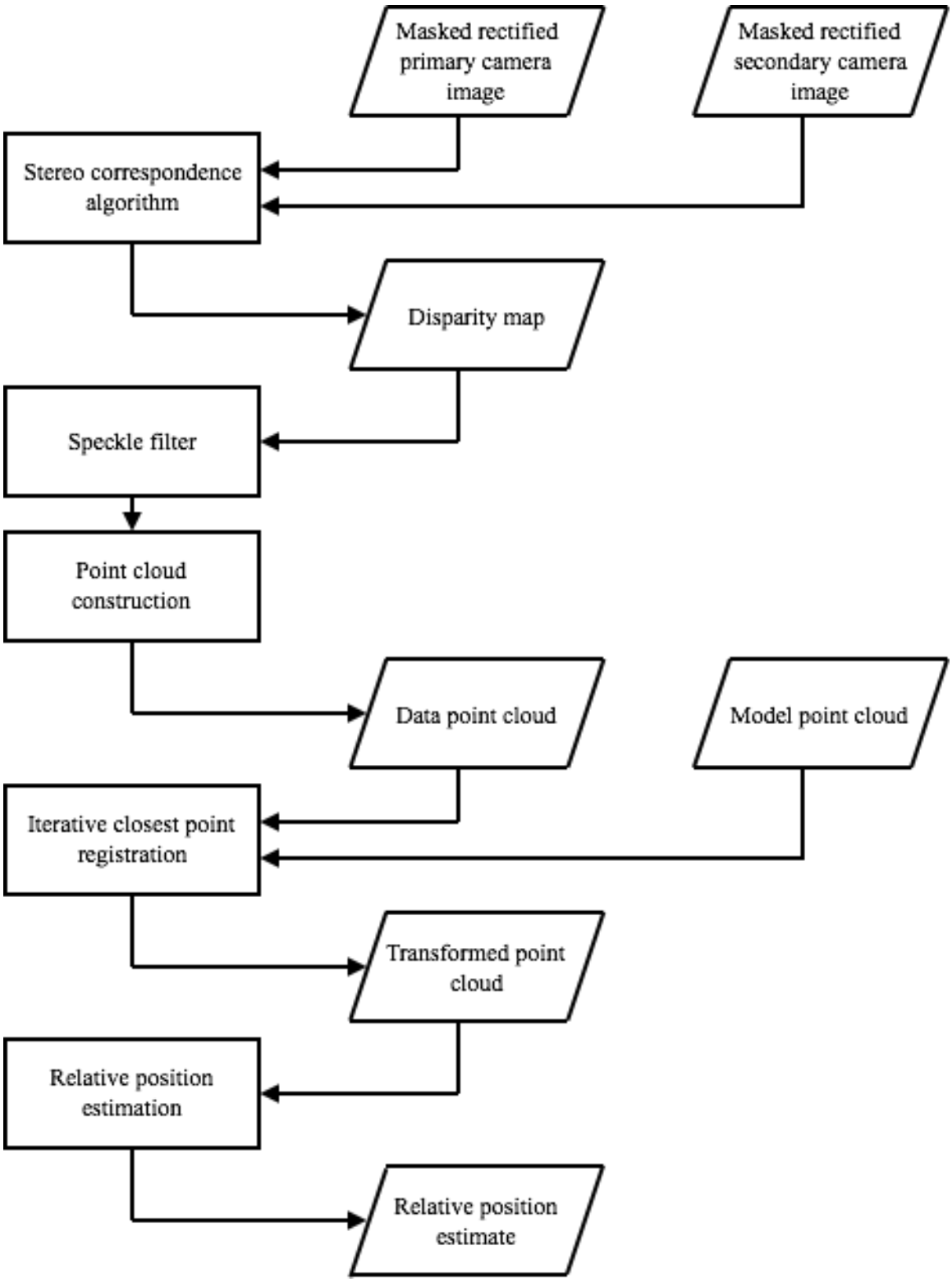


Figure 11. The relative position estimation process

IV. Results

Given rectified stereo images with associated regions of interest for the approaching aircraft, results suggest that the relative position of a refueling aircraft may be estimated in real time at a rate between 10 Hz and 30 Hz. Furthermore, real time execution is feasible not only on the GPU. Block matching on the CPU achieves execution times suitable for real time position estimation as well. Results with data from the real world domain exhibit error vector magnitudes comparable to previous research in the simulation domain after adjusting for scale [42]. Finally, seeding the iterative closest point registration algorithm with the final estimation from the previous time step results in nominally smaller position estimate error vector magnitudes.

4.1 Stereo algorithm execution times

Real world data was analyzed using three different stereo correspondence algorithms from OpenCV 3.0.0. Two of these algorithms, semi-global block matching and block matching, execute on the CPU. The third algorithm, also a block matching algorithm, executes in the GPU. These algorithms are abbreviated CPU SGBM (semi-global block matching on the CPU), CPU BM (block matching on the CPU), and GPU BM (block matching on the GPU). Each algorithm was set through its parameters to use a disparity search range of 48 pixels and a block size of 9 pixels.

The semi-global block matching algorithm accepts additional, optional parameters. These parameters enable extra processing that can improve resulting disparity maps at the expense of increased execution times, so long as the values are precisely tuned. Previous research focused on tuning these optional parameters for the simulation domain [42]. In order to maximize comparability between stereo algorithms, these optional parameters were not utilized for this analysis.

(a) Downsampled CPU SGBM point cloud

(b) Downsampled CPU BM point cloud

(c) Downsampled GPU BM point cloud

(d) Truth model point cloud

Figure 12. Example point clouds

The execution time for the CPU SGBM algorithm averages 57.97 ± 0.57 ms per image pair, or about 17.25 Hz. The execution time for the CPU BM algorithm is 4.20 ± 0.51 ms on average, or about 237.91 Hz. The GPU BM algorithm executes in 0.92 ± 0.01 ms on average. Including upload and download memory transfer times, the algorithm executes in 1.14 ± 0.01 ms on average, or about 879.76 Hz.

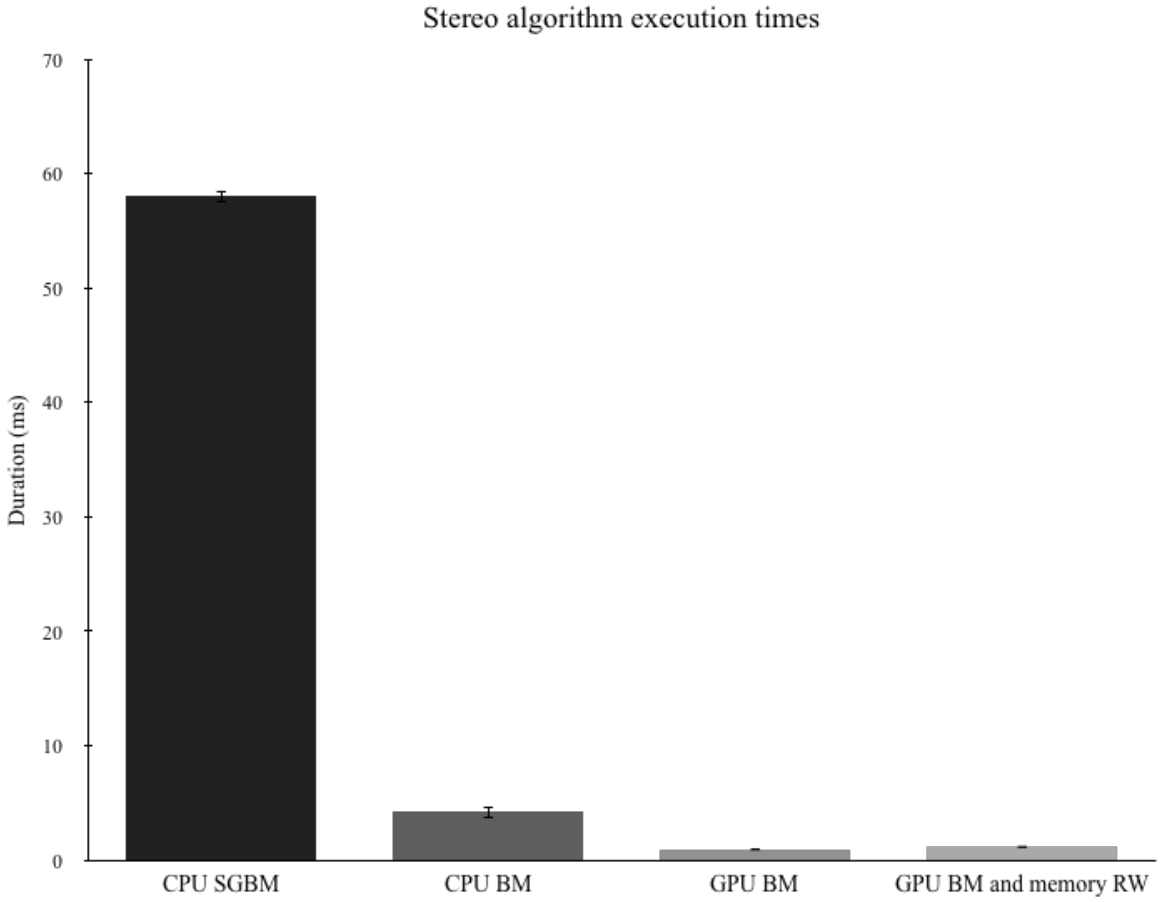


Figure 13. Average execution times for stereo algorithms

Each stereo algorithm generates disparity maps from stereo image pairs. These disparity maps must then be filtered and projected into three dimensions before using a registration algorithm in order to make a position estimate.

4.2 Disparity map speckle filter execution times

Some disparity maps output by stereo algorithms contain "speckles," or contiguous areas with erroneous disparity values. OpenCV provides a CPU-only function for filtering these small noise areas in disparity maps.

In practice, filtering a disparity map for speckles reduces the number of erroneous points after projecting into three dimensions. For example, projecting a disparity map into three dimensions without filtering for speckles could result in erroneous points placed at or near the origin of the camera frame of reference. Erroneous points could be filtered after projection into three dimensions. However, filtering in a higher dimensional space could be computationally costly.

In order to maximize comparability, the output of each algorithm was passed through the same speckle filter. Specifically, the speckle filter was initialized with a maximum speckle area of 144 pixels and a maximum disparity value difference of 4 between neighbor pixels in order to be considered part of the same speckle.

The speckle filter execution time on disparity maps from the CPU SGBM algorithm averages 0.96 ± 0.24 ms. The speckle filter execution time on disparity maps from the CPU BM algorithm is 2.16 ± 0.15 ms on average. The speckle filter on disparity maps from the GPU BM algorithm executes in 0.56 ± 0.11 ms on average.

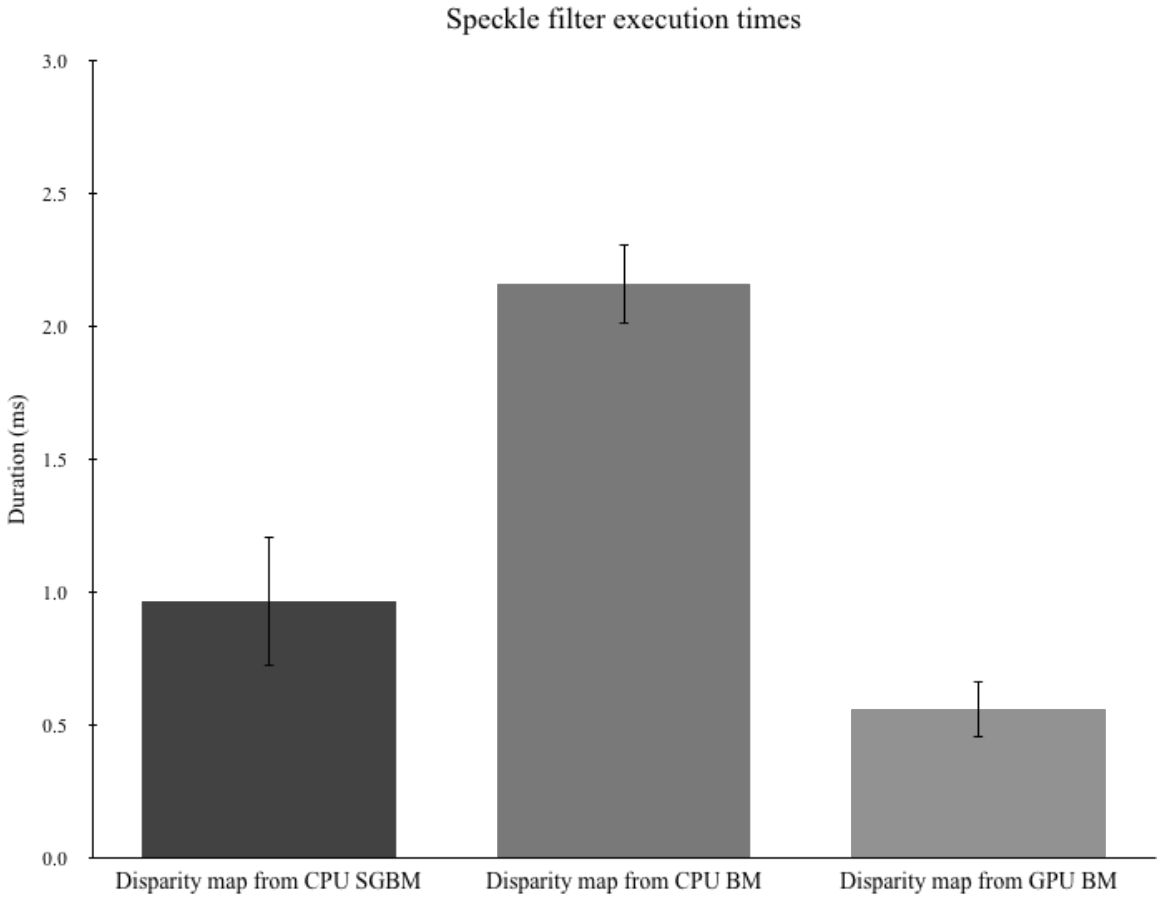


Figure 14. Average execution times for the speckle filter

Given a stereo image pair, a corresponding disparity map contains the same number of channels, the same width, and the same height regardless of the stereo algorithm used to generate the map. Thus, it may be expected that an identical speckle filter applied to disparity maps generated from identical stereo images passed through different stereo algorithms would execute in identical amounts of time on average.

Among disparity maps generated from different stereo imagery passed through the same stereo algorithm, this expectation is supported by data. However, the mean execution times of speckle filters on disparity maps generated from the same stereo imagery passed through different stereo algorithms are statistically different. This fact is illustrated by low t-test p-values.

Table 1. Speckle filter Welch two sample t-test results with an alternative hypothesis that the difference in means is not equal to zero. Values are in ms with ns precision.

CPU SGBM mean	CPU BM mean	GPU BM mean	p-value
0.9634078	2.1568558	-	$< 2.2 \times 10^{-16}$
0.9634078	-	0.5550956	$< 2.2 \times 10^{-16}$
-	2.1568558	0.5550956	$< 2.2 \times 10^{-16}$

4.3 Point cloud generation execution times

A disparity map is projected into three dimensions using information from a 4×4 matrix $\mathbf{Q}$ generated during stereo calibration. The $\mathbf{Q}$ matrix and disparity map values are used to calculate a three dimensional point cloud according to the following formula [7].

$$\mathbf{Q} \begin{pmatrix} x \\ y \\ d \\ 1 \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix} \tag{4}$$

In this formula, x and y are the pixel coordinates (column and row, respectively) of the disparity value d. The three dimensional point corresponding to a given disparity value has coordinates $(X/W, Y/W, Z/W)^\top$.

Depending on the distance between the cameras and the model jet, point clouds generated from the collected data contain between 1500 and 22000 valid points. In order to reduce iterative closest point execution times, a downsampling function is applied during point cloud generation.

Previous research applied stochastic universal sampling (SUS) to downsample point clouds with the aim of preserving point cloud features [42]. Here, a point is omitted under the downsampling criteria if a random value is below a threshold.

This approach exhibits no major drawbacks compared to the SUS downsampling method with the advantage of a straightforward implementation for real time processing. After downsampling and applying a bounds check, point clouds generated from the collected data contain between 200 and 1000 points.

Point cloud generation execution times range between 1 ms and 10 ms. Generation of the point clouds was performed on the CPU. Because each vector $(x, y, d, 1)^\top$ in a two dimensional disparity map undergoes the same calculation in order to produce a corresponding point in three dimensions, this process may be parallelized for increased speed.

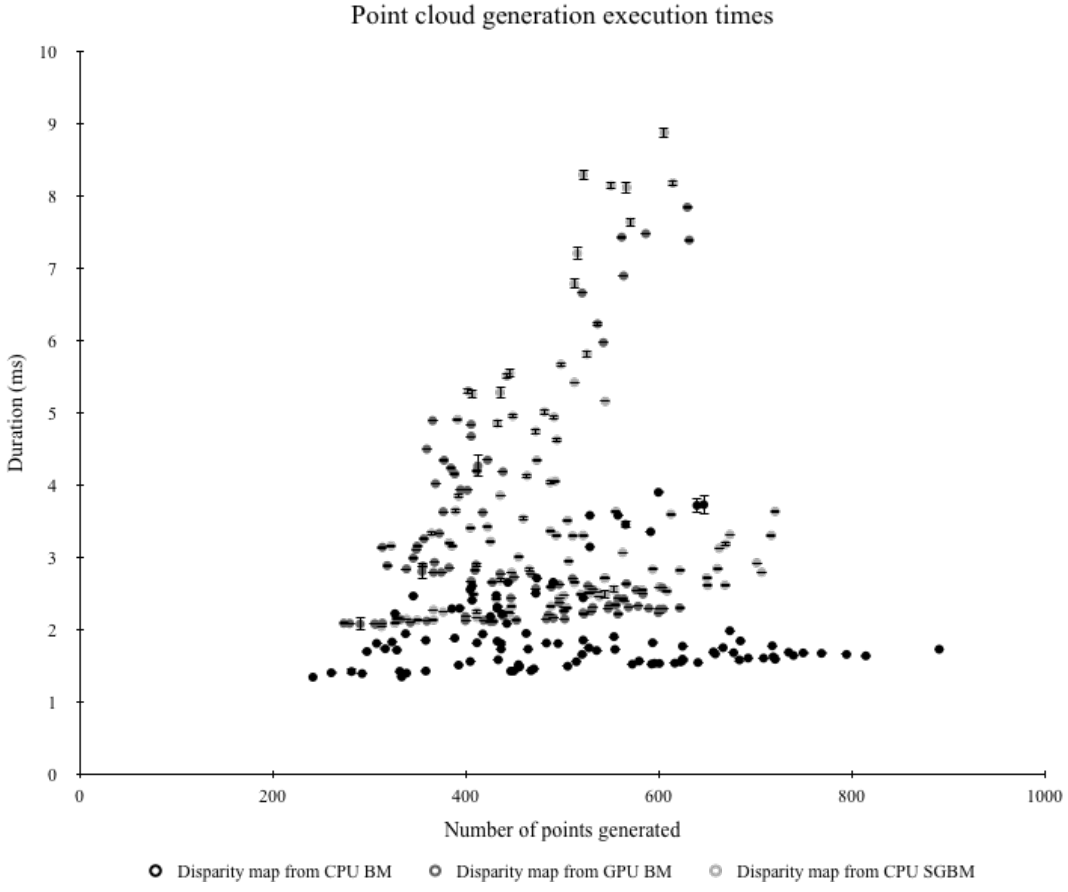


Figure 15. Average execution times for point cloud generation

4.4 Iterative closest point execution times

After projecting a disparity map into a three dimensional "target" point cloud, a "source" or "truth" point cloud is registered to the target point cloud using the iterative closest point algorithm. As noted previously, target point clouds generated from stereo data contain between 200 and 1000 points. The source point cloud is a three dimensional model of an F15-E composed of 1000 points.

Two different initial conditions for the iterative closest point algorithm were tested. For the "unseeded" initial condition, the source point cloud is placed at the origin of the camera frame of reference. For the "seeded" initial condition, the source point cloud is placed at its ending registration position in the previous time step. Because the position of the target point cloud in a given time step is likely to be similar to its position in adjacent time steps, seeding the position of the source point cloud with previous solutions could increase the quality of the registration.

Each execution of the iterative closest point algorithm ran for 10 iterations. The number of points in the target point clouds, ranging from 200 to 1000, exhibit little influence over the execution time. Additionally, the difference between the means of seeded ICP execution times and unseeded ICP execution times is not statistically significant. This fact is illustrated by a t-test p-value of 0.07822.

Table 2. ICP Welch two sample t-test results with an alternative hypothesis that the difference in means is not equal to zero. Values are in ms with ns precision.

Seeded ICP duration mean	Unseeded ICP duration mean	p-value
30.81123	30.77444	0.07822

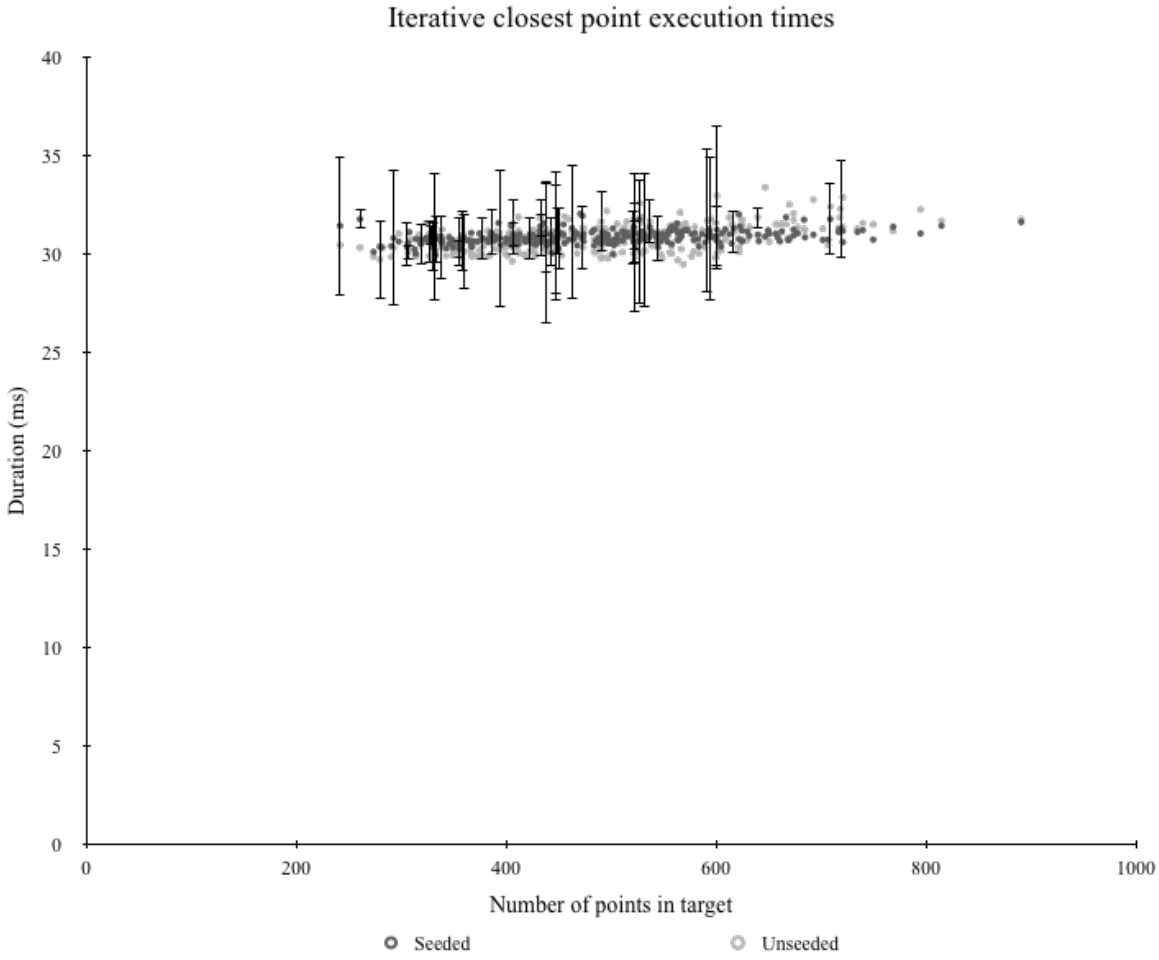


Figure 16. Execution times for iterative closest point algorithm. Points without visible error bars have error within ±0.25 ms.

Table 3. Linear regression analysis of iterative closest point execution times

	Estimate	**Std. error**	t **value**	$\mathbf{Pr}(>\|t\|)$	R^2
Seeded intercept	30.2575	0.0504	600.35	$< 2.0 \times 10^{-16}$	0.0412
Seeded slope	0.0011	0.0001	11.29	$< 2.0 \times 10^{-16}$	
Unseeded intercept	29.4023	0.0702	418.87	$< 2.0 \times 10^{-16}$	0.1197
Unseeded slope	0.0028	0.0001	20.09	$< 2.0 \times 10^{-16}$	

The iterative closest point algorithm occupies a significant portion of the total execution time for relative position estimation. Except in the case of semi-global block matching on the CPU, the iterative closest point algorithm occupies the majority of the total execution time for relative position estimation. When the source point cloud contains approximately 1000 points, the duration of the ICP algorithm with a convergence criteria of 10 iterations averages approximately 30.79 ms.

However, source point clouds with a greater number of points exhibit larger execution times. For example, ICP with a source point cloud containing approximately 5000 points executes in 150.00 ms on average. As a result, generating a relative position estimate before the next pair of stereo images is available would not be feasible. Therefore, larger source point cloud sizes can inhibit real time execution.

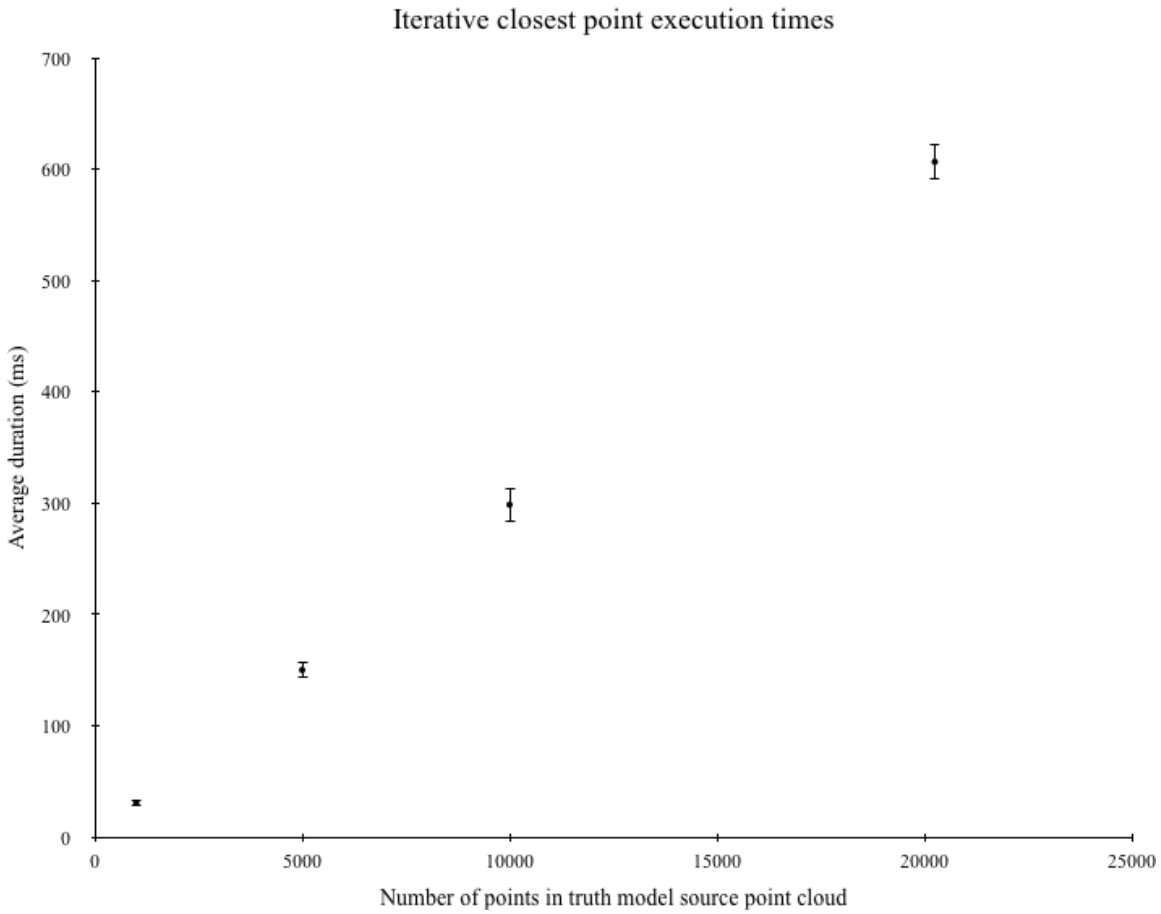


Figure 17. ICP execution times as a function of the source point cloud size

4.5 Summary of execution times

Given rectified stereo images with associated regions of interest for the approaching aircraft, the relative position of a refueling aircraft may be estimated in real time. Results suggest that semi-global block matching on the CPU could provide position estimates at a rate of up to approximately 10.7 Hz. Block matching on the CPU could provide position estimates at a rate of up to approximately 25.4 Hz. Finally, block matching on the GPU could provide position estimates at a rate of up to approximately 28.4 Hz. Thus, real-time execution is feasible not only by leveraging the GPU, but by using only the CPU as well. The majority of execution time for both the CPU and GPU block matching approaches is consumed by ICP.

Table 4. Average execution times. Timing values have ns precision.

	CPU SGBM	**CPU BM**	**GPU BM**
Stereo algorithm (ms)	57.973786	4.203305	1.136670
Speckle filter (ms)	0.963408	2.156856	0.555096
Point cloud generation (ms)	3.642200	1.931584	3.110842
Iterative closest points (ms)	30.838964	31.146360	30.393187
Sum (ms)	93.418358	39.438105	35.195795
Hz	10.704534	25.356188	28.412485

4.6 Position estimation accuracy

After using the iterative closest point algorithm to register the source F15-E model to the target point cloud, the centroid of the point cloud may be used in order to estimate the relative position of the observed aircraft. This method for estimating position was used in previous work [42].

In order to compare real world data position estimation accuracy to previous work's position estimation accuracy in the simulation domain, the same method for position estimation is employed. The real world data collection took place at an approximately one-seventh scale of the simulation data in previous work. Results from the simulation domain study have been scaled down by a factor of seven for the sake of comparison [42].

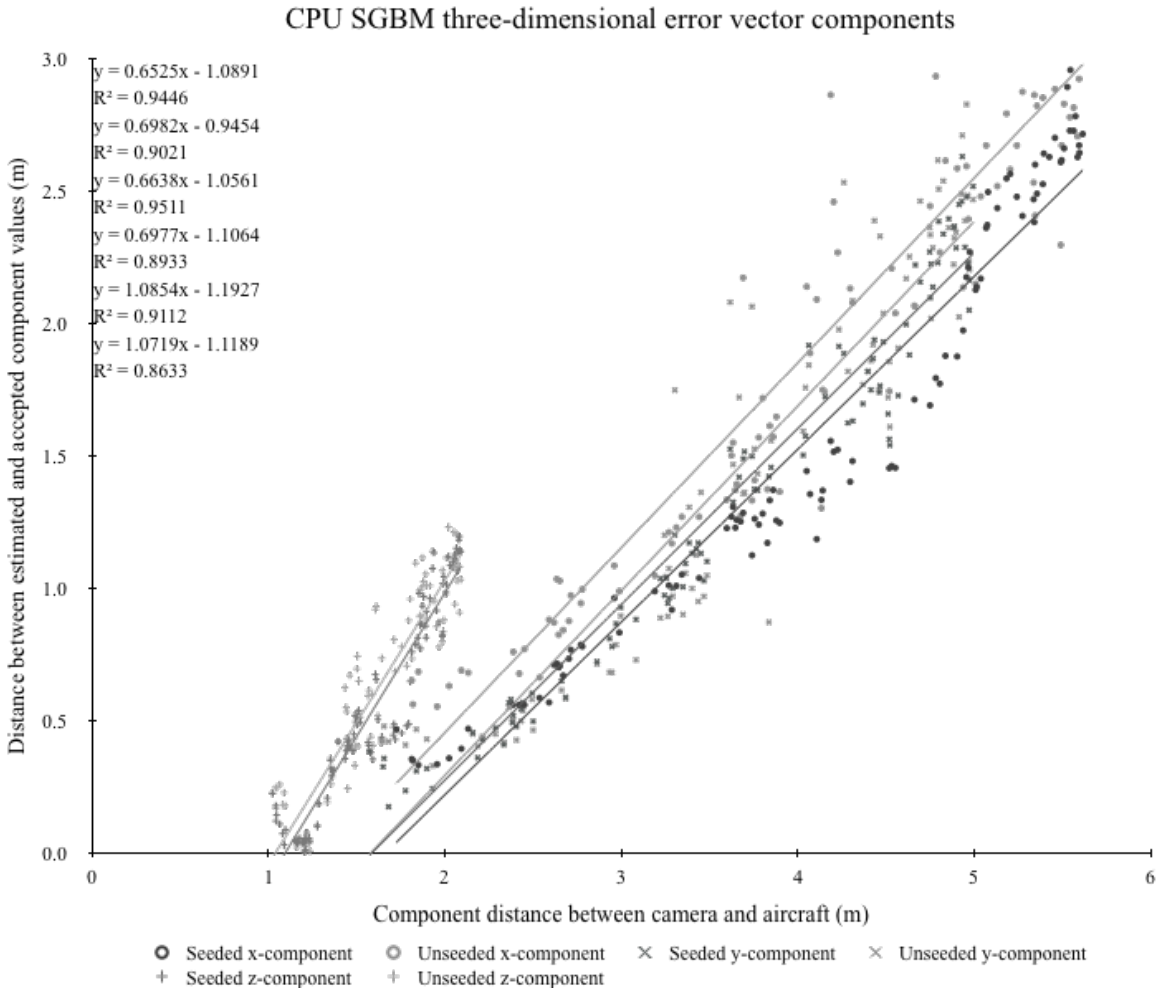


Figure 18. Relative position error by components for CPU SGBM

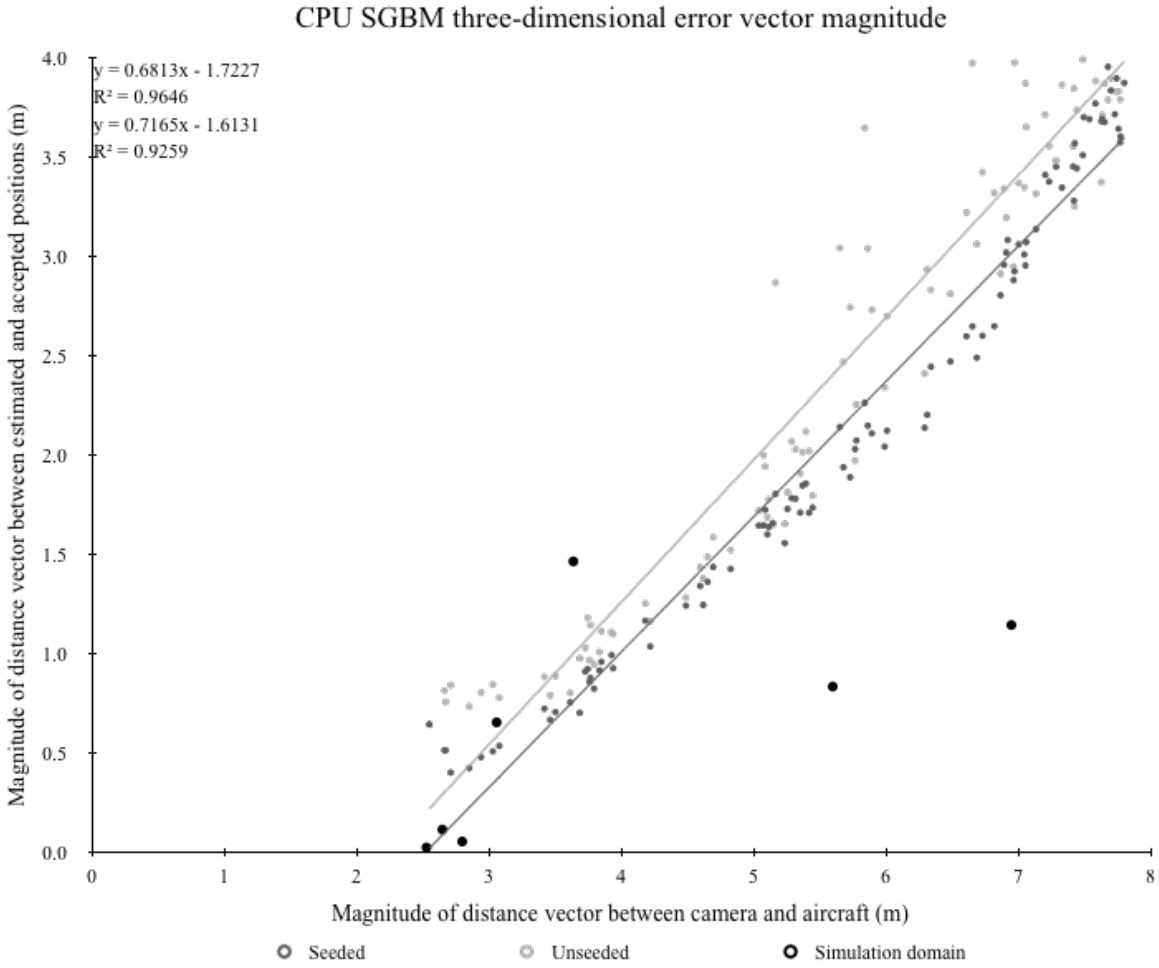


Figure 19. Relative position error magnitudes for CPU SGBM

Table 5. Linear regression analysis of CPU SGBM 3D error vector magnitudes

	Estimate	**Std. error**	t **value**	$\mathbf{Pr}(>\|t\|)$	R^2
Seeded intercept	-1.7227	0.0786	-22.10	$< 2.0 \times 10^{-16}$	0.9646
Seeded slope	0.6813	0.0132	51.69	$< 2.0 \times 10^{-16}$	
Unseeded intercept	-1.6131	0.1211	-13.32	$< 2.0 \times 10^{-16}$	0.9259
Unseeded slope	0.7165	0.0205	34.99	$< 2.0 \times 10^{-16}$	

Real world data analyzed by the CPU semi-global block matching stereo algorithm exhibit trends qualitatively similar to trends in the simulation domain results from previous work by Werner [42]. Seeding the iterative closest point algorithm appears to result in nominally lower errors. Note the increase in the z-component of the error vector as the magnitude of the distance vector decreases from 3 m towards 2 m. This increase corresponds to the occlusion of the imaged jet nose as it exits the bottom of the frame.

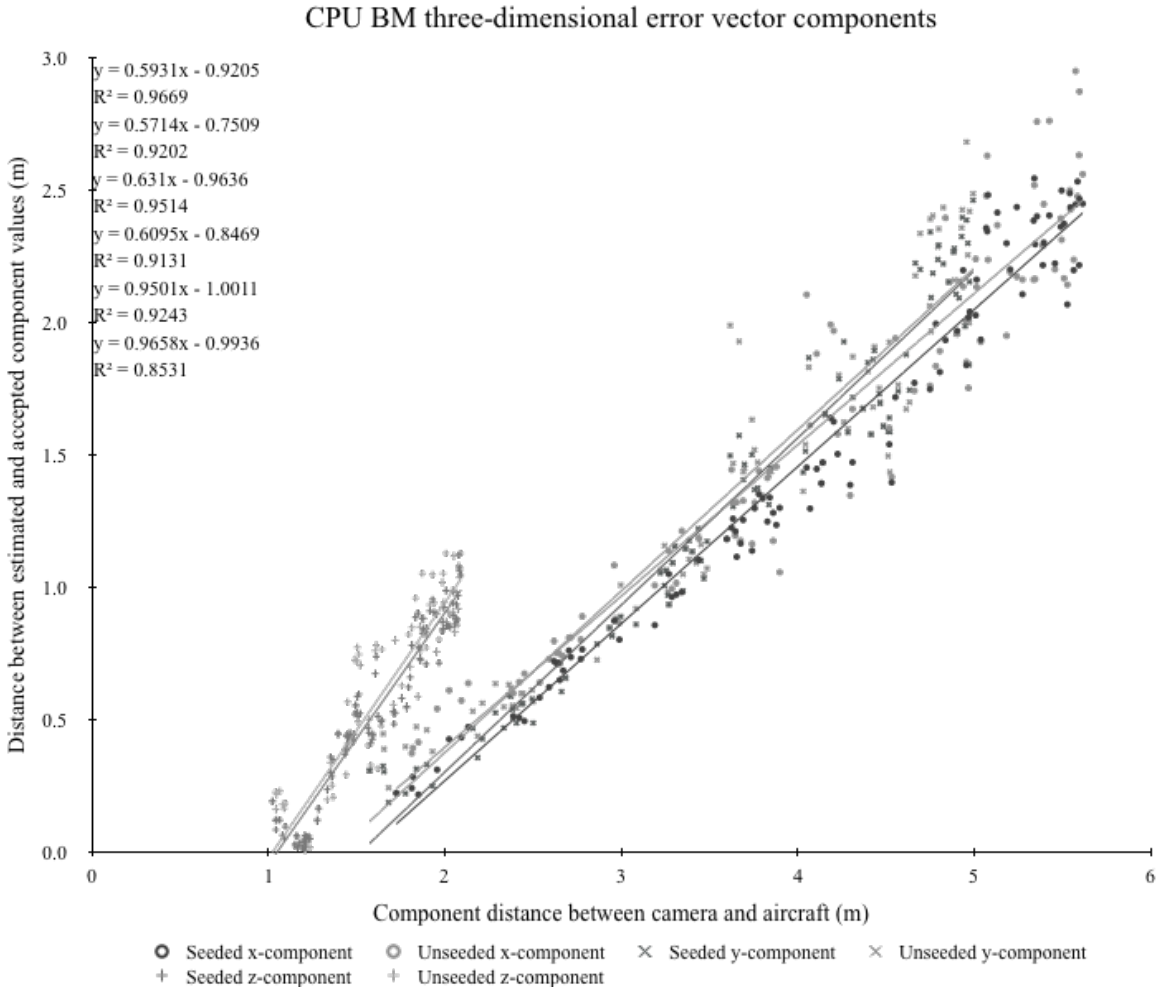


Figure 20. Relative position error by components for CPU BM

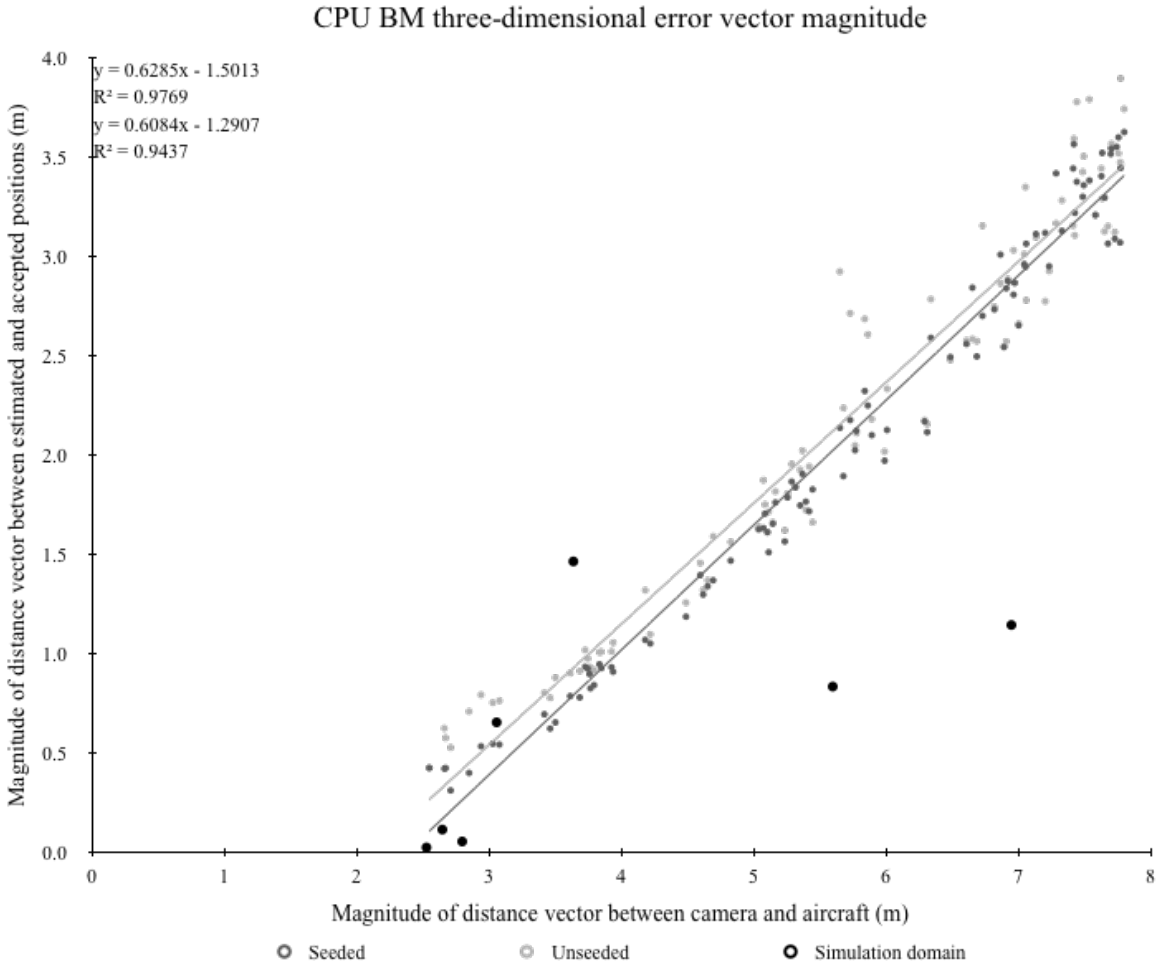


Figure 21. Relative position error magnitudes for CPU BM

Table 6. Linear regression analysis of CPU BM 3D error vector magnitudes

	Estimate	**Std. error**	t **value**	$\mathbf{Pr}(>\|t\|)$	R^2
Seeded intercept	-1.5013	0.0578	-25.98	$< 2.0 \times 10^{-16}$	0.9769
Seeded slope	0.6285	0.0098	64.32	$< 2.0 \times 10^{-16}$	
Unseeded intercept	-1.2907	0.0888	-14.54	$< 2.0 \times 10^{-16}$	0.9437
Unseeded slope	0.6084	0.0156	40.53	$< 2.0 \times 10^{-16}$	

For all three stereo algorithms, the R^2 value associated with the linear regression model for the seeded algorithm is higher than the R^2 value associated with the linear regression model for the unseeded algorithm. This fact suggests that seeding the iterative closest point algorithm leads to more predictable and less noisy relative position estimates.

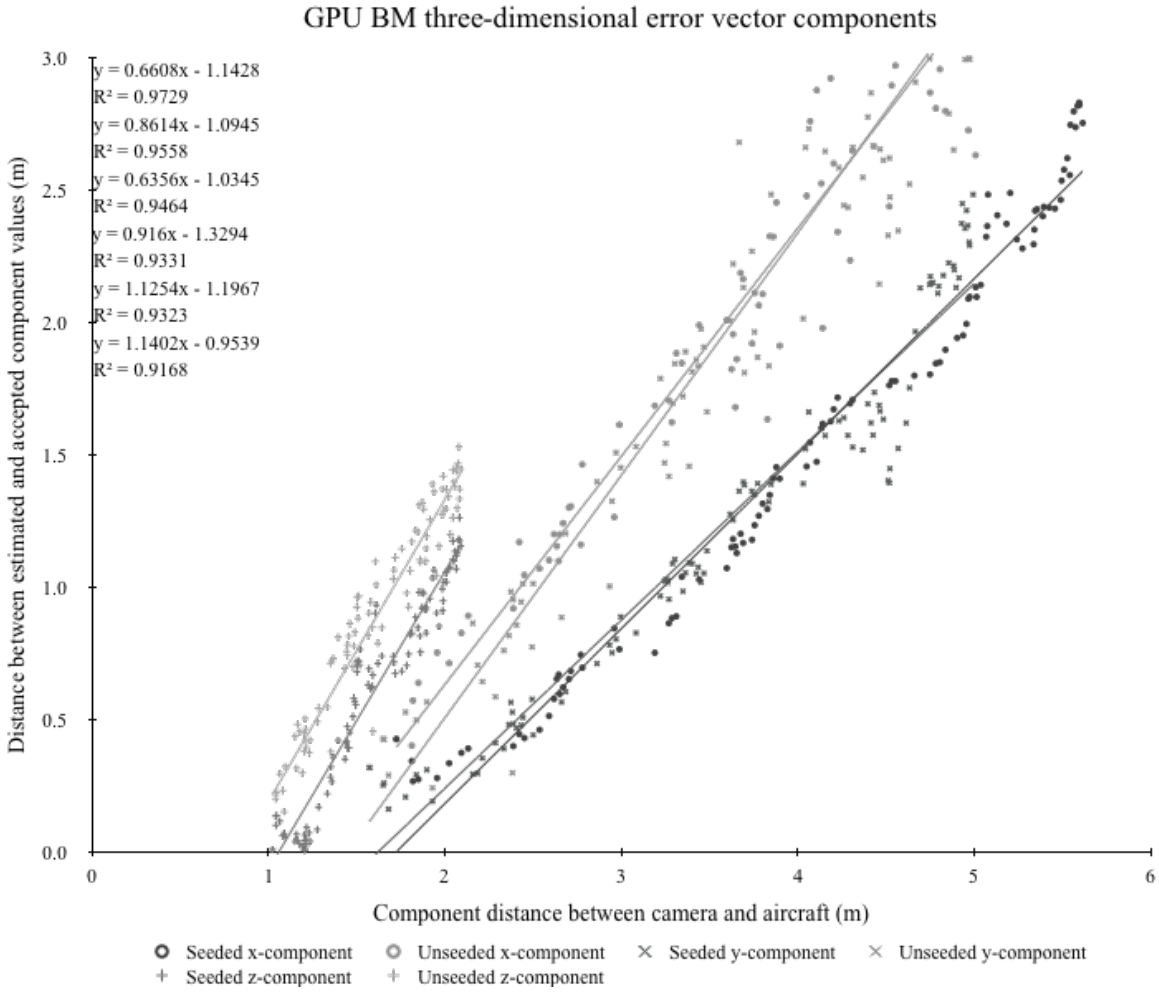


Figure 22. Relative position error by components for GPU BM

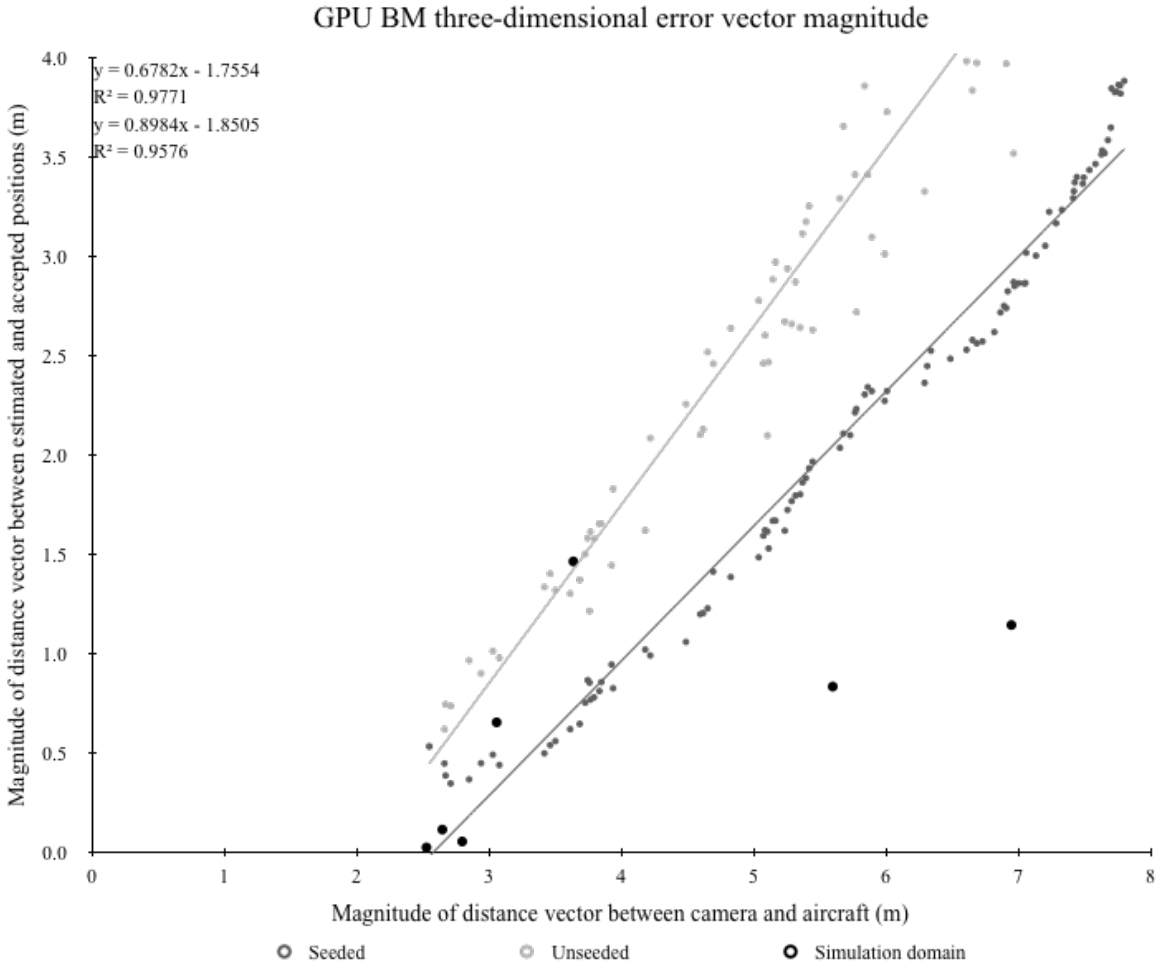


Figure 23. Relative position error magnitudes for GPU BM

Table 7. Linear regression analysis of GPU BM 3D error vector magnitudes

	Estimate	**Std. error**	t **value**	$\mathbf{Pr}(>\|t\|)$	R^2
Seeded intercept	-1.7554	0.0620	-28.32	$< 2.0 \times 10^{-16}$	0.9771
Seeded slope	0.6782	0.0105	64.71	$< 2.0 \times 10^{-16}$	
Unseeded intercept	-1.8505	0.1129	-16.39	$< 2.0 \times 10^{-16}$	0.9576
Unseeded slope	0.8984	0.0191	47.06	$< 2.0 \times 10^{-16}$	

V. Conclusion

Aerial refueling is essential to the USAF core mission of rapid global mobility. However, in-flight refueling is not available to remotely piloted aircraft or unmanned aerial systems. As reliance on drones for ISR and other USAF core missions grows, the ability to refuel such systems in-flight becomes increasingly critical.

Research towards automated aerial refueling aims to solve this problem. Researchers have been investigating the AAR problem for over a decade, and approaches have utilized GPS, INS, monocular machine vision, and LiDaR. New capabilities in upcoming tankers introduce the possibility of a stereo vision solution.

5.1 Summary of findings

This investigation demonstrates that real world stereo imagery may be used to estimate relative position with accuracies comparable to accuracies observed in simulation. Real time relative position estimation is feasible, both on the GPU and on the CPU. Given rectified stereo images and associated region of interest masks, relative position estimation may be performed at rates between 10 Hz and 30 Hz. Changes to support real time execution exhibited no significant impact on position estimation accuracies compared to accuracies reported in simulation-based work conducted by Werner [42].

In order to support comparisons between stereo algorithms, all three functions were initialized with the same parameters. As a result, these algorithms could be ranked according to the ability of a linear regression to explain the position error vector magnitude as a function of distance between the cameras and the subject.

Block matching on the GPU exhibits the most predictable and least noisy relationship between these factors, and semi-global block matching on the CPU exhibits the

least predictable and most noisy relationship between these factors. However, these algorithms use different approaches in order to determine a stereo correspondence. Individually tuning the input parameters for these functions could result in different rankings.

Seeding the iterative closest point algorithm with the solution from the previous time step in order to propagate information improves accuracy. The magnitudes of relative position error vectors produced by the unseeded iterative closest point algorithm are greater than the magnitudes produced by the seeded algorithm on average.

5.2 Future work

One area of future work involves the application of filters. In order to reduce the amount of erroneous disparity values before projecting into a three dimensional point cloud, a speckle filter was applied. Despite there being no theoretical difference between disparity maps generated by different stereo algorithms, results indicate that speckle filter execution times differ statistically according to the originating stereo correspondence function. A model characterizing speckle filter execution time could help to explain these results.

Results suggest that the relative position estimates could be resilient to noise. Specifically, the results from high-fidelity simulation domain data in other work are predicted by a linear regression model on the more noisy data from this investigation. The tradeoff between omitting the speckle filter for faster execution times at a cost of greater noise could help to quantify the importance of the speckle filter.

Other filters could be applied at different stages of the relative position estimation process. For this investigation, the input is assumed to consist of rectified stereo images with associated region of interest masks. These masks are another type of filter.

Bounding boxes remove anomalous points that exceed sensible values. Clustering algorithms could be applied after point cloud generation in order to further filter the data. Isolating the points of interest while maintaining real time execution further supports stereo vision towards a solution to the AAR problem.

Similar to the speckle filter execution times, point cloud generation execution times exhibit different responses to the number of points generated depending on the stereo algorithm that produced the disparity map. Characterizing point cloud generation execution time as a function of the number of points generated may also help to place an upper bound on point cloud size for feasible real time execution. Implementing point cloud generation on the GPU could support real time execution as well.

Unlike the speckle filter execution times and point cloud generation execution times, the iterative closest point algorithm execution times exhibit no significant response to the number of points in the target point cloud. A characterization of iterative closest point algorithm execution times could help to explain these results.

While the size of the target point clouds varied, the size of the source point cloud remained constant. Changing the size of the source point cloud could aid in model development. Additionally, changing the number of iterations performed and other convergence criteria could help to construct a model of the tradeoff between execution time and accuracy.

Finally, large errors in long range position estimates are observed both in other work in the simulation domain and in this investigation using real world data. Errors reduce dramatically as the distance between cameras and the imaged object decreases. In order for stereo vision to be a viable solution to the AAR problem, efforts should be made to reduce these errors. Sensor fusion techniques or other modifications may also improve position estimation accuracies at greater imaging distances.

Appendix A. Computer Hardware

In the interest of reproducibility, this appendix reports the computer hardware on which software for this project is developed, tested, and executed. Keyboards, mouses, displays, cables, and other peripherals are also required.

Table 8. Data collection hardware

Item	Identification number	Details
Intel NUC	NUC5i5RYK	Core i5-5250U two core 1.6 GHz CPU
G.skill Ripjaws RAM	F3-1866C10D-16GRSL	DDR3L 2×8GB 1866 MHz 10-10-10-32 SO-DIMM RAM
Samsung 850 Evo M.2 SSD	MZ-N5E500BW	500GB M.2 SSD

Table 9. Data analysis hardware

Item	Identification number	Details
Fractal Design Define R4	FD-CA-DEF-R4-BL	ATX mid tower case
Corsair AX760 power supply	CP-9020045-NA	ATX12V 760W power supply
Asus Sabertooth Z97 Mark 1/USB 3.1 motherboard	90MB0LA0-M0AAY0	Intel LGA 1150 ATX motherboard
Intel Core i7-4770S	BX80646I74770S	65W four core 3.1 GHz CPU
2 × G.skill Sniper RAM	F3-1866C9D-16GSR	DDR3 2×8GB 1866 MHz 9-10-9-28 240-pin RAM
Nvidia Geforce Gtx 980	04G-P4-1982-KR	2048 CUDA cores Evga 4GB
Samsung 850 Evo SSD	MZ-75E1T0B/AM	1TB 2.5” SATA III SSD

Appendix B. Operating System

Software developed for this project is cross-platform or, more specifically, may run under recent versions of Linux, Mac, or Windows operating systems with the appropriate hardware. Due to time and resource constraints, compatibility on all platforms has not necessarily been tested. As a result, minor tweaks may be required before running software on some operating systems. All development decisions were made under the principle of producing cross-platform code.

In order to support an appropriate rate of progress, software compatibility is ensured only for Ubuntu 14.04.3 LTS. This operating system is largely open source, guaranteed support until April 2019, and free to download, install, and use [9]. The majority of development and testing occurred on this operating system.

The following steps present installation methods for the Ubuntu 14.04.3 LTS operating system used during this project. These steps serve as a record of the methodology employed when installing the operating system.

1. Obtain the Ubuntu 14.04.3 LTS Desktop 64-bit ISO image.

   This file may be available at the following URL:

   `http://releases.ubuntu.com/trusty/ubuntu-14.04.3-desktop-amd64.iso`

2. Place the ISO image on an installation device.

   (a) Burn the ISO image to a DVD.

   Detailed steps may be available at the following URL:

   `https://help.ubuntu.com/community/BurningIsoHowto`

   (b) Place the ISO image on a USB storage device.

   Detailed steps may be available at the following URLs:

   `https://help.ubuntu.com/community/Installation/FromUSBStick`

   `https://www.ubuntu.com/download/desktop/create-a-usb-stick-on-mac-osx`

3. Insert the installation media into the computer, and boot from the device.

   The easiest way to boot from the installation device may be from the BIOS boot menu.

4. Choose to install Ubuntu.

   - Welcome: For this project, the "English" option is chosen.
   - Wireless: In some cases, continuing without connecting to the Internet seems to cause problems with the installation.
   - Preparing to install Ubuntu: The options "has at least 6.6 GB available drive space," "is connected to the Internet," "Download updates while installing," and "Install this third-party software" are all checked. Omitting some combinations of the last three options may cause issues when using OpenCV for this project.
   - Installation type: Some machines used for development contain additional operating systems, while others only contain Ubuntu.
     - No additional operating systems: Choose "Erase disk and install Ubuntu."
     - Additional operating systems: Choose "Something else."
       * Create a partition for the root file system "/" with a size from 16000 MB to 24000 MB. The new partition should be of type "Primary" at the beginning of the partition space. Choose an "Ext4 journaling file system" with "/" as the mount point.
       * Create a partition for the swap with a size between one and 1.5 times the amount of RAM (e.g. a 48000 MB swap size for 32000 MB RAM). The new partition should be of type "Logical" at the beginning of the partition space. Choose to use as swap area.

        * Create a partition for "/home" using the remaining space on the storage device. The new partition should be of type "Logical" at the beginning of the partition space. Choose an "Ext4 journaling file system" with "/home" as the mount point.

    - Choose "Install now," and accept the changes written to the disk.
    - Where are you: For this project, the default selection is acceptable. Consult the IANA Time Zone Database (also referred to as the tz database or zoneinfo database) for other options.
    - Keyboard layout: For this project, the "English (US)" and "English (US)" options are chosen.
    - Who are you: The name, computer name, username, password, and password confirmation field are completed. The option to require a password to login is selected. The encryption option is not selected, because this option seems to cause problems for the resulting installation.

5. After installation, install updates presented by the Ubuntu Software Update application.

6. Finally, execute the following Terminal commands.

```
sudo apt-get update
sudo apt-get upgrade
```

Appendix C. Software dependencies

All of the software developed for this project depend on the OpenCV 3.0.0 release. Before installing OpenCV, several dependencies are required. The `build-essential`, `cmake`, `git`, `libgtk2.0-dev`, `pkg-config`, `libavcodec-dev`, `libavformat-dev`, and `libswscale-dev` packages are required [31].

The `python-dev`, `python-numpy`, `libtbb2`, `libtbb-dev`, `libjpeg-dev`, `libpng-dev`, `libtiff-dev`, `libjasper-dev`, and `libdc1394-22-dev` packages are recommended [31].

OpenCV automatically makes use of the `libwebp-dev`, `libopenexr-dev`, `doxygen`, `libv4l-dev`, `libavresample-dev`, `libeigen3-dev`, `libgphoto2-dev`, `python3-dev`, `python3-numpy`, `libgstreamer1.0-dev`, and `libgstreamer-plugins-base1.0-dev` packages when they are present [31].

Additionally, OpenCV makes use of the `libopenni-sensor-primesense-dev`, `libopenni-dev`, `libxine-dev`, and `libgdal-dev` packages so long as they are specifically set to "ON" when generating compilation files with the CMake command [31].

The `freeglut3-dev`, `mesa-utils`, `libqt4-dev`, and `libvtk5-dev` packages provide graphics and visualization functionalities leveraged by OpenCV. This combination of packages results in warning messages during the one-time process of generating compilation files for OpenCV, but these warning messages can be ignored. To avoid these inconsequential warning messages, investigate using Qt 5 and VTK 6.

The following terminal command installs the dependencies used by this project for the OpenCV 3.0.0 release. A description for most of these packages may be viewed by appending the package name to the end of the "`https://apps.ubuntu.com/cat/applications/`" URL.

```
sudo apt-get install build-essential cmake git libgtk2.0-dev
pkg-config libavcodec-dev libavformat-dev libswscale-dev
python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev
libtiff-dev libjasper-dev libdc1394-22-dev libwebp-dev
libopenexr-dev libavresample-dev libeigen3-dev libgphoto2-dev
libv4l-dev libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
doxygen python3-dev python3-numpy libopenni-dev
libopenni-sensor-primesense-dev libxine-dev libgdal-dev
freeglut3-dev mesa-utils libqt4-dev libvtk5-dev
```

Additionally, the CUDA API must be installed for data analysis. This process changes frequently and depends on the operating system. See the Nvidia CUDA website for details.

After installing the above packages and CUDA, the OpenCV 3.0.0 release can be downloaded, compiled, and installed. According to the Filesystem Hierarchy Standard [26], the `/usr/local` directory is the appropriate location for software like the OpenCV 3.0.0 release. So long as no other versions of OpenCV are present, the source code may be placed in `/usr/local/src` and the installation prefix may be set to `/usr/local`. If the user does not have appropriate privileges to install OpenCV in `/usr/local` or multiple versions of OpenCV must exist on the system, then any other directory may be used.

The following terminal commands retrieve the appropriate version of the source code using `wget` and `unzip`. If the source directory is not `/usr/local/src`, then the `sudo` command may not be necessary.

```
cd /usr/local/src/
sudo wget https://github.com/Itseez/opencv/archive/3.0.0.zip
sudo unzip 3.0.0.zip
cd opencv-3.0.0/
```

Alternatively, the following terminal commands may be used to retrieve the appropriate version of the source code in the git repository.

```
cd /usr/local/src/
sudo git clone https://github.com/Itseez/opencv.git
cd opencv/
sudo git checkout 3.0.0
sudo mv opencv/ opencv-3.0.0/
```

After retrieving the appropriate version of the source code, the compilation files must be generated. The following terminal commands may be used to generate these compilation files. These commands must be executed from the `opencv-3.0.0/` working directory. The final two characters of the `cmake` command, `..`, indicate that the `CMakeLists.txt` file in the `opencv-3.0.0/` directory should be used to generate the compilation files.

```
sudo mkdir build
cd build/
sudo cmake -D CMAKE_BUILD_TYPE=RELEASE
-D CMAKE_INSTALL_PREFIX=/usr/local -D WITH_GDAL=ON
-D WITH_OPENMP=ON -D WITH_OPENNI=ON -D WITH_OPENGL=ON
-D WITH_QT=ON -D WITH_TBB=ON -D WITH_XINE=ON ..
# also include the flags -D WITH_CUBLAS and -D WITH_NVCUVID
# if the CUDA API is installed
```

Finally, the following terminal commands compile and install OpenCV.

```
sudo make -j 4 # on a four core CPU
sudo make install
```

Data collection and analysis also rely on the SQLite relational database management software [37], the Tesseract optical character recognition (OCR) software [38], and Point Cloud Library (PCL) [33]. The following terminal command installs SQLite and Tesseract.

```
sudo apt-get install sqlite3 libsqlite3-dev tesseract-ocr
libtesseract-dev libleptonica-dev
```

To install PCL, first ensure that the following packages are installed: `libcf0`, `libeigen3-dev`, `libflann-dev`, `libflann1.8`, `libgl2ps-dev`, `libgl2ps0`, `libhdf5-7`, `libnetcdf-dev`, `libnetcdfc++4`, `libnetcdfc7`, `libnetcdff5`, `libopenni2-0`, `libqhull6`, `libopenni2-dev`, `libqhull-dev`, `libusb-1.0-0-dev`, `libusb-1.0-doc`, `libvtk5-dev`, `libvtk5-qt4-dev`, `libvtk5.8`, `libvtk5.8-qt4`, `libxss-dev`, `tcl8.6-dev`, `tk8.6-dev`, `x11proto-scrnsaver-dev`.

```
sudo apt-get install libcf0 libeigen3-dev libflann-dev libflann1.8
libgl2ps-dev libgl2ps0 libhdf5-7 libnetcdf-dev libnetcdfc++4
libnetcdfc7 libnetcdff5 libopenni2-0 libopenni2-dev libqhull-dev
libqhull6 libusb-1.0-0-dev libusb-1.0-doc libvtk5-dev libvtk5-qt4-dev
libvtk5.8 libvtk5.8-qt4 libxss-dev tcl8.6-dev tk8.6-dev
x11proto-scrnsaver-dev
```

Next, download the source code, generate compilation files, compile, and install.

```
wget https://github.com/PointCloudLibrary/pcl/archive/pcl-1.8.0rc1.zip
unzip pcl-1.8.0rc1.zip
cd pcl-pcl-1.8.0rc1/ && mkdir build && cd build/
cmake -D CMAKE_BUILD_TYPE=Release -D CMAKE_INSTALL_PREFIX=/home/usrnm
-D BUILD_CUDA=ON -D BUILD_GPU=ON ..
make -j 4
make install
```

Appendix D. Software development

The project software is developed with the Git distributed version control system [18]. Development takes place at two levels, a macro level and a micro level. Macro level development follows the "forking workflow," and micro level development follows the "gitflow workflow" [2].

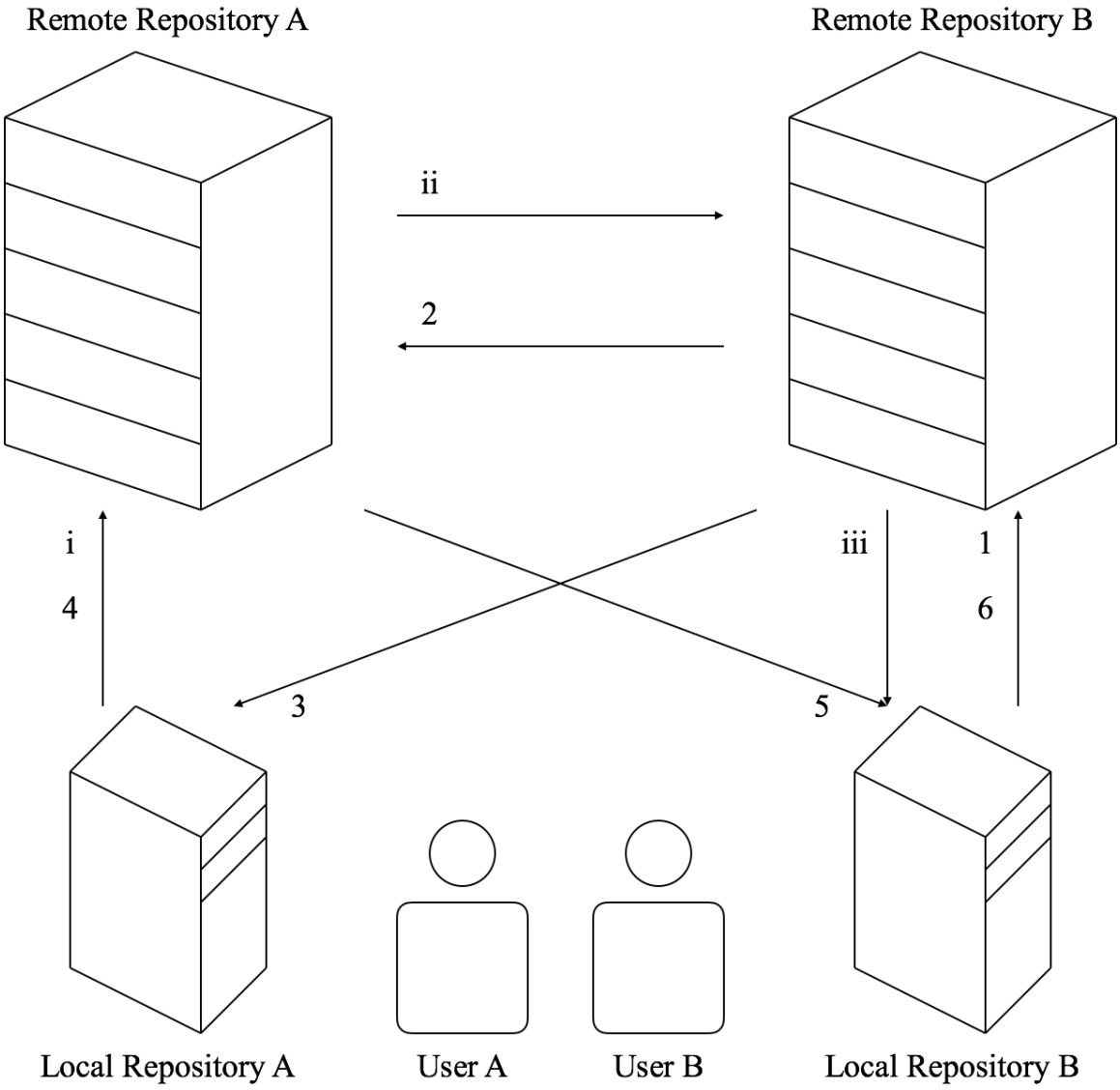


Figure 24. An illustration of the forking workflow used during software development

The macro level forking workflow begins with three initialization steps. Initialization step i consists of User A initializing Local Repository A and pushing it to Remote Repository A.

```
usera$ mkdir /path/to/repository/ && cd /path/to/repository/
usera$ git init
usera$ git config user.name User A
usera$ git config user.email ua@example.com
usera$ git remote add origin https://usera@example.com/usera/
repository.git
# Create a README file
usera$ git add README.txt
usera$ git commit -m "README"
usera$ git push -u origin master
```

Initialization step ii consists of User B forking Remote Repository A in order to create Remote Repository B. This step is performed through the remote repositories server interface. Initialization step iii consists of User B cloning Remote Repository B in order to create Local Repository B.

```
userb$ git clone https://userb@example.com/userb/repository.git
userb$ cd repository/
userb$ git config user.name User B
userb$ git config user.email ub@example.com
userb$ git remote add upstream https://userb@example.com/usera/
repository.git
```

After the three initialization steps, the macro level forking workflow consists of a repeating sequence of six steps. Before step 1 of the macro level workflow, User B makes changes to Local Repository B according to the micro level gitflow workflow.

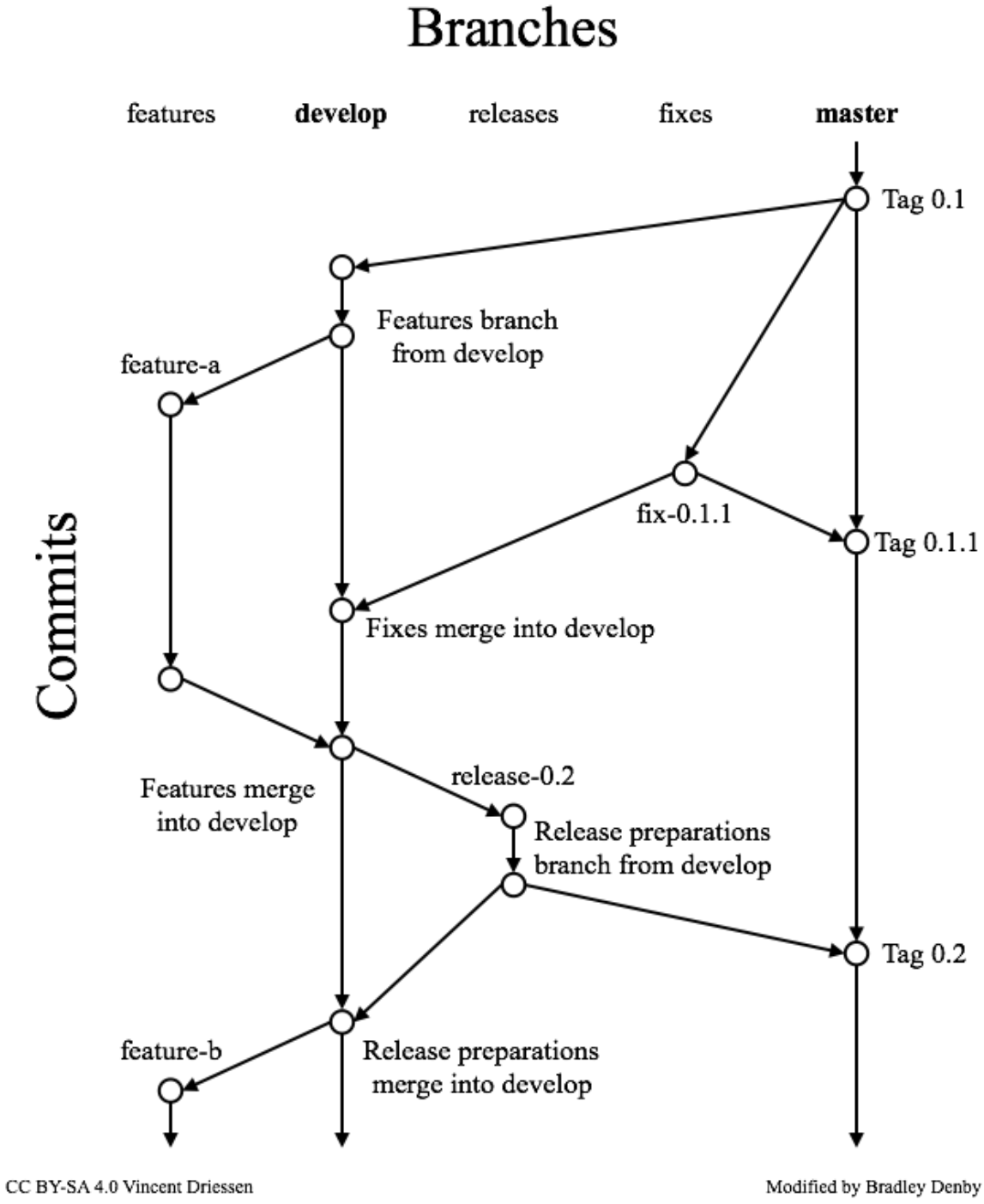


Figure 25. An illustration of the gitflow workflow used during software development

Specifically, User B checks out a branch (a feature branch, a release branch, a fix branch, or develop) in which changes will be made.

```
# Create a new feature branch
userb$ git checkout -b feature-a develop
# Create a new fix branch
userb$ git checkout -b fix-0.1.1 master
# Create a new release branch
userb$ git checkout -b release-0.2 develop
```

After making changes in Local Repository B according to the micro level gitflow workflow and committing them to a branch, User B performs step 1 of the macro level forking workflow. User B pushes changes made to Local Repository B to Remote Repository B.

```
userb$ git push origin branch-name
```

Step 2 of the macro level forking workflow consists of User B submitting a pull request from the updated branch in Remote Repository B to an appropriate target branch in Remote Repository A. See Figure 25 for an illustration of appropriate merges. Step 2 is performed through the remote repositories server interface.

In step 3 of the macro level forking workflow, User A fetches the changes submitted by User B from Remote Repository B to Local Repository A.

```
usera$ git fetch https://usera@example.com/userb/
repository.git branch-name
usera$ git checkout FETCH_HEAD
```

User A tests the code for acceptability. Upon a successful test, the changes are merged into the indicated target branch in Local Repository A.

```
usera$ git checkout target-branch
usera$ git merge --no-ff FETCH_HEAD
# omit --no-ff if merging develop into develop
```

Step 4 of the macro level forking workflow consists of User A pushing the merged changes from Local Repository A to Remote Repository A. The associated pull request should be automatically resolved after step 4.

```
usera$ git push origin target-branch
```

Once the merged changes have been pushed to Remote Repository A, User B performs step 5 of the macro level forking workflow. The merged changes are pulled from Remote Repository A to Local Repository B.

```
userb$ git checkout target-branch
userb$ git pull upstream target-branch
```

Finally, in step 6 of the macro level forking workflow, User B pushes the merged changes from Local Repository B to Remote Repository B.

```
userb$ git push origin target-branch
```

If the original branch is not a permanent branch (i.e. the original branch is not the develop or master branch), then it should be removed from the repositories.

```
userb$ git branch -d branch-name
userb$ git push origin --delete branch-name
```

Appendix E. Network time protocol server

References such as [40] and [23] outline procedures for building a stratum 1 network time protocol (NTP) server with a Raspberry Pi computer. However, these references may be scattered, out of date, or lack specifics. Thus, the following information provides a specific record of the methodology employed when constructing the NTP server for the data collection environment local area network (LAN).

The materials listed in Table 10 were used in order to construct the NTP server. Additional materials may be required. Examples of such materials include Ethernet cables, a Power over Ethernet (PoE) switch, a PoE splitter, and a weatherproof garden case for outdoor cabling. Before configuring the NTP server, the materials must be constructed into a computer. This process includes some soldering. Assembly instructions are available from the purchase site [1]. After assembly, the Raspberry Pi may be configured for WiFi network access and access through Secure Shell (SSH).

Table 10. NTP server materials

Item	**Adafruit.com PID**
Raspberry Pi 2 Model B	2358
Adafruit Raspberry Pi B+/Pi 2 case	2258
8 GB SD card with Raspbian Jessie	2767
SMA to μFL/μ.FL/IPX/IPEX RF adapter	851
3-5 V 28 dB 5 m SMA External active GPS antenna	960
CR1220 12 mm 3 V lithium coin cell battery	380
2 pack brass M2.5 standoffs for Pi HAT	2336
Adafruit Ultimate GPS HAT for Raspberry Pi	2324
GPIO header for Pi HAT 2×20 short female	2243
USB WiFi 802.11 b/g/n antenna for Pi	1030

After the materials have been assembled, the following steps configure the device as an NTP server.

1. Enable PPS

```
$ lsmod | grep pps # there should be no output
$ sudo apt-get install pps-tools
$ sudo reboot
$ lsmod | grep pps # there should be no output
$ sudo nano /boot/config.txt
  # add the following line
  dtoverlay=pps-gpio,gpiopin=4
$ sudo nano /etc/modules
  # add the following line
  pps-gpio
$ sudo reboot
$ lsmod | grep pps # output should include pps_gpio and pps_core
$ sudo ppstest /dev/pps0
```

2. Compile NTP from source

```
$ cd Downloads/
$ wget http://archive.ntp.org/ntp4/ntp-4.2/ntp-4.2.8p4.tar.gz
$ wget http://archive.ntp.org/ntp4/ntp-4.2/ntp-4.2.8p4.tar.gz.md5
$ md5 ntp-4.2.8p4.tar.gz
$ cat ntp-4.2.8p4.tar.gz.md5
$ mkdir ntp-configure-prefix
$ sudo apt-get install libcap-dev libevent-dev libssl-dev
$ cd ntp-4.2.8p4
$ ./configure --prefix=/home/pi/Downloads/ntp-configure-prefix/
  --enable-ATOM --enable-NMEA --enable-SHM --enable-linuxcaps
```

```
$ make -j 4
$ make install
```

3. Stop, remove, and replace the existing NTP service

```
$ systemctl | grep ntp
$ sudo systemctl stop ntp.service
$ cd Downloads/ntp-configure-prefix/
$ sudo cp bin/* /usr/bin/
$ sudo cp sbin/* /usr/sbin/
$ sudo cp -r share/doc/ntp/. /usr/share/doc/ntp/
$ sudo cp -r share/doc/sntp /usr/share/doc/
$ cd /usr/share/man/man1/
$ sudo rm ntpdc.1.gz ntpq.1.gz ntpsweep.1.gz ntptrace.1.gz
  sntp.1.gz
$ cd /home/pi/Downloads/ntp-configure-prefix/
$ sudo cp share/man/man1/* /usr/share/man/man1/
$ sudo rm /usr/share/man/man5/ntp.conf.5.gz
$ sudo cp share/man/man5/* /usr/share/man/man5/
$ cd /usr/share/man/man8/
$ sudo rm ntpd.8.gz ntp-keygen.8.gz ntp-wait.8.gz
$ cd /home/pi/Downloads/ntp-configure-prefix/
$ sudo cp share/man/man8/* /usr/share/man/man8/
$ sudo cp -r share/ntp /usr/share/
$ sudo systemctl start ntp.service
```

4. Configure the serial port so that it may be used for GPS

```
$ sudo nano /boot/cmdline.txt
  # remove 'console=ttyAMA0,115200' to read:
  dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2
  rootfstype=ext4 elevator=deadline rootwait
$ sudo systemctl stop serial-getty@ttyAMA0.service
$ sudo systemctl disable serial-getty@ttyAMA0.service
$ sudo systemctl mask serial-getty@ttyAMA0.service
$ sudo reboot
$ stty -F /dev/ttyAMA0 raw 9600 cs8 clocal -cstopb
$ cat /dev/ttyAMA0
$ sudo nano /etc/udev/rules.d/99-gps.rules
  # add the following lines
  KERNEL=="ttyAMA0",SYMLINK+="gps0"
  KERNEL=="pps0",SYMLINK+="gpspps0"
```

5. Edit the NTP configuration file `/etc/ntp.conf`

```
$ sudo nano /etc/ntp.conf
```

- Add the following lines

```
server 127.127.20.0 mode 18 minpoll 4 maxpoll 4 noselect
fudge 127.127.20.0 flag1 1
```

- Wait approximately 24 hours, then run the following commands

```
$ ntpq -pn
$ ntpq -p
```

- Make note of the offset (e.g. −524.105)

- Edit the previously inserted lines so that the `time2` value approximately eliminates the offset

```
server 127.127.20.0 mode 18 minpoll 4 maxpoll 4 prefer
fudge 127.127.20.0 time2 +0.500 flag1 1
```

- Optionally, include the following line if the local clock is available

```
server 127.127.1.0
```

- Comment out the other servers

Additional documentation, such as details for driver 20, is available from the NTP manual [30]. After some time, the NTP server may be checked by running the command line function `$ date` and confirming that the output is correct. In order to subscribe another computer on the LAN to the NTP server time, the computer's local `/etc/ntp.conf` file must be edited. The NTP server's IP address must be entered into the computer's local `/etc/ntp.conf` file using appropriate formatting.

Bibliography

1. Adafruit Industries. *Adafruit Ultimate GPS HAT for Raspberry Pi*, Nov 2015. `https://learn.adafruit.com/adafruit-ultimate-gps-hat-for-raspberry-pi?view=all`.

2. Atlassian. *Comparing workflows*, Oct 2015. `https://www.atlassian.com/git/tutorials/comparing-workflows/`.

3. Tracy J. Barnidge and Joseph L. Tchon. 3d display considerations for rugged airborne environments. In *Proc. SPIE*, 2015.

4. Paul J. Besl and Neil D. McKay. Method for registration of 3-d shapes. In *Proc. SPIE*, volume 1611, pages 586–606, 1992.

5. Glen Bever, Peter Urschel, and Curtis E. Hanson. Comparison of relative navigation solutions applied between two aircraft. Technical Report NASA/TM-2002-210728, H-2498, NAS 1.15:210728, NASA Dryden Flight Research Center, Jun 2002.

6. S. Birchfield and C. Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(4):401–406, Apr 1998.

7. Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, Inc., 2nd edition, 2013.

8. Sean M. Calhoun, John Raquet, and Joe Curro. Flight test evaluation of image rendering navigation for close-formation flight. In *Proceedings of the 25th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS 2012)*, pages 826–832, Nashville, TN, Sept 2012.

9. Canonical. *Releases*, Oct 2015. `https://wiki.ubuntu.com/Releases`.

10. Joseph A. Curro II. Automated aerial refueling position estimation using a scanning lidar. Master's thesis, Air Force Institute of Technology, Mar 2012.

11. Boguslaw Cyganek and J Paul Siebert. *An introduction to 3D computer vision techniques and algorithms*. John Wiley & Sons, 2011.

12. Olivier Faugeras, Bernard Hotz, Herve Mathieu, Thierry Vieville, Zhengyou Zhang, Pascal Fua, Eric Theron, Laurent Moll, Gerard Berry, Jean Vuillemin, Patrice Bertin, and Catherine Proy. Real time correlation-based stereo: algorithm, implementations and applications. Technical report, INRIA, 1993.

13. Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient belief propagation for early vision. *International Journal of Computer Vision*, 70(1):41–54, 2006.

14. United States Air Force. Kc-x tanker modernization program, Mar 2011. https://www.fbo.gov/index?id=b0305f7a7e5d6958dc613bf08c4c3151.

15. United States Air Force. Air force core missions, Aug 2013. http://www.af.mil/News/ArticleDisplay/tabid/223/Article/466868/air-force-core-missions.aspx.

16. Mario Fravolini, Marco Mammarella, Giampiero Campa, Marcello Napolitano, and Mario Perhinschi. Machine vision algorithms for autonomous aerial refueling for uavs using the usaf refueling boom method. In Anthony Finn and Lakhmi C. Jain, editors, *Innovations in Defence Support Systems 1*, volume 304 of *Studies in Computational Intelligence*, pages 95–138. Springer Berlin Heidelberg, 2010.

17. Mario L. Fravolini, Giampiero Campa, and Marcello R. Napolitano. Evaluation of machine vision algorithms for autonomous aerial refueling for unmanned aerial vehicles. *Journal of Aerospace Computing, Information, and Communication*, 4(9):968–985, Sept 2007.

18. Git. *Git reference manual*, Oct 2015. https://git-scm.com/docs.

19. R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

20. Richard I. Hartley. Theory and practice of projective rectification. *International Journal of Computer Vision*, 35(2):115–127, 1999.

21. Richard I. Hartley and Peter Sturm. Triangulation. *Computer Vision and Image Understanding*, 68(2):146–157, 1997.

22. H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(2):328–341, Feb 2008.

23. Thomas Kocourek. *Raspberry Time Broadband Ham Net*, Nov 2015. http://www.satsignal.eu/ntp/Raspberry_Time%20-%20Broadband%20Ham%20Net.pdf.

24. K. Konolige. Projected texture stereo. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 148–155, May 2010.

25. Kurt Konolige. Small vision systems: Hardware and implementation. In Yoshiaki Shirai and Shigeo Hirose, editors, *Robotics Research*, pages 203–212. Springer London, 1998.

26. The Linux Foundation. *Filesystem Hierarchy Standard*, June 2015. http://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf.

27. M. Mammarella, G. Campa, M.R. Napolitano, M.L. Fravolini, Yu Gu, and M.G. Perhinschi. Machine vision/gps integration using ekf for the uav aerial refueling problem. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(6):791–801, Nov 2008.

28. Marco Mammarella, Giampiero Campa, Marcello Napolitano, and Mario Fravolini. Comparison of point matching algorithms for the uav aerial refueling problem. *Machine Vision and Applications*, 21(3):241–251, 2010.

29. Marco Mammarella, Giampiero Campa, Marcello Napolitano, Brad Seanor, Mario Fravolini, and Lorenzo Pollini. Gps / mv based aerial refueling for uavs. *Guidance, Navigation, and Control and Co-located Conferences*, Aug 2008.

30. Network Time Protocol. *Network Time Protocol Documentation*, Nov 2015. `http://doc.ntp.org/4.1.2/`.

31. OpenCV. *OpenCV 3.0.0-dev documentation*, Jan 2016. `http://docs.opencv.org/3.0-beta/index.html`.

32. Bradford W. Parkinson and Per K. Enge. *Differential GPS*, pages 3–50. Progress in Astronautics and Aeronautics. American Institute of Aeronautics and Astronautics, Jan 1996.

33. Point Cloud Library. *PCL 1.8.0 documentation*, Jan 2016. `http://docs.pointclouds.org/trunk/classpcl_1_1_iterative_closest_point.html`.

34. Steven M. Ross. Formation flight control for aerial refueling. Master's thesis, Air Force Institute of Technology, Mar 2006.

35. Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1-3):7–42, 2002.

36. Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: Exploring photo collections in 3d. *ACM Trans. Graph.*, 25(3):835–846, Jul 2006.

37. SQLite. *SQLite documentation*, Oct 2015. `http://www.sqlite.org/docs.html`.

38. Tesseract OCR. *Tesseract OCR documentation*, Dec 2015. `https://github.com/tesseract-ocr/tesseract/wiki`.

39. Justin S. Tharp. On the integration of medium wave infrared cameras for vision-based navigation. Master's thesis, Air Force Institute of Technology, Mar 2015.

40. Aaron Toponce. *Building a stratum 1 NTP server with a Raspberry Pi*, Nov 2015. `http://xmission.com/blog/2014/05/28/building-a-stratum-1-ntp-server-with-a-raspberry-pi`.

41. Michael J. Veth. *Fusion of Imaging and Inertial Sensors for Navigation*. PhD thesis, Air Force Institute of Technology, Sept 2006.

42. Kyle P. Werner. Precision relative positioning for automated aerial refueling from a stereo imaging system. Master's thesis, Air Force Institute of Technology, Mar 2015.

43. W.R. Williamson, M.F. Abdel-Hafez, Ihnseok Rhee, Eun-Jung Song, J.D. Wolfe, D.F. Chichka, and J.L. Speyer. An instrumentation system applied to formation flight. *Control Systems Technology, IEEE Transactions on*, 15(1):75–85, Jan 2007.

44. Qingxiong Yang, Liang Wang, and N. Ahuja. A constant-space belief propagation algorithm for stereo matching. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1458–1465, June 2010.

45. Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2):119–152, 1994.

46. Zhengyou Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1330–1334, Nov 2000.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
24–03–2016	Master's Thesis	Sept 2014 — Mar 2016

4. TITLE AND SUBTITLE

Towards automated aerial refueling:
Real time position estimation with stereo vision

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Denby, Bradley D.

5d. PROJECT NUMBER

16-359

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-16-M-252

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Ba T Nguyen
Aerospace Systems Directorate
Air Force Research Laboratory
2210 8TH ST
WPAFB OH 45433-7765
(937) 938-4617 ba.nguyen@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RQ

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT Aerial refueling is essential to the United States Air Force (USAF) core mission of rapid global mobility. However, in-flight refueling is not available to remotely piloted aircraft (RPA) or unmanned aerial systems (UAS). As reliance on drones for intelligence, surveillance, and reconnaissance (ISR) and other USAF core missions grows, the ability to automate aerial refueling for such systems becomes increasingly critical. New refueling platforms include sensors that could be used to estimate the relative position of an approaching aircraft. Relative position estimation is a key component to solving the automated aerial refueling (AAR) problem. Analysis of data from a one-seventh scale, real world refueling scenario demonstrates that the relative position of an approaching aircraft can be estimated at rates between 10 Hz and 30 Hz using stereo vision. Linear regression models on position estimate accuracies predict results reported by other research in the simulation domain, suggesting that real world accuracies are comparable to simulation domain accuracies reported by others. Further, by seeding the position estimation algorithm with previous position estimates, subsequent errors in position estimation are reduced.

15. SUBJECT TERMS

LaTeX, Thesis, aerial refueling, AAR, stereo vision, position estimation, relative navigation, computer vision

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Maj Brian G. Woolley, PhD, AFIT/ENG
U	U	U	U	103	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x4618; brian.woolley@afit.edu