

8-2016

Vision-Aided Navigation for Autonomous Vehicles Using Tracked Feature Points

Ahmed Saber Soliman Sayem

Follow this and additional works at: <https://commons.erau.edu/edt>



Part of the [Automotive Engineering Commons](#)

Scholarly Commons Citation

Sayem, Ahmed Saber Soliman, "Vision-Aided Navigation for Autonomous Vehicles Using Tracked Feature Points" (2016). *Dissertations and Theses*. 240.

<https://commons.erau.edu/edt/240>

This Thesis - Open Access is brought to you for free and open access by Scholarly Commons. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of Scholarly Commons. For more information, please contact commons@erau.edu.

VISION-AIDED NAVIGATION FOR
AUTONOMOUS VEHICLES USING TRACKED FEATURE POINTS

A Thesis

Submitted to the Faculty

of

Embry-Riddle Aeronautical University

by

Ahmed Saber Soliman Sayem

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Unmanned and Autonomous Systems Engineering

August 2016

Embry-Riddle Aeronautical University

Daytona Beach, Florida

VISION-AIDED NAVIGATION FOR
AUTONOMOUS VEHICLES USING TRACKED FEATURE POINTS

by

Ahmed Saber Soliman Sayem

A Thesis prepared under the direction of the candidate's committee chairman, Dr. Richard J. Prazenica, Department of Aerospace Engineering, and has been approved by the members of the thesis committee. It was submitted to the College of Engineering and was accepted in partial fulfillment of the requirements for the degree of Master of Science in Unmanned and Autonomous Systems Engineering.

THESIS COMMITTEE



Chairman, Dr. Richard J. Prazenica



Member, Dr. Hever Moncayo



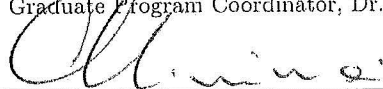
Member, Dr. Troy Henderson



Department Chair, Dr. Timothy A. Wilson
or Graduate Program Coordinator, Dr. Richard Stansbury

26 JUL 2016

Date



Dean of College of Engineering, Dr. Maj Mirmirani

07/26/2016

Date



Vice Chancellor, Academic Support, Dr. Christopher Grant

8/4/16

Date

ACKNOWLEDGMENTS

To Richard Skrabe who without him, this work would not have been possible. To my advisor Dr. Richard Prazenica who I can not express in words how much help I got from him to make this happens. To my wife, Amanda who supported and encouraged me to finish this thesis all the way till the end. To Dr. Ilteris Demirkiran, my mentor and my friend who was always there for me.

TABLE OF CONTENTS

	Page
LIST OF TABLES	vi
LIST OF FIGURES	vii
ABBREVIATIONS	ix
ABSTRACT	x
1 Introduction	1
1.1 Literature Review	2
1.2 Statement of Hypothesis	4
2 Feature Detection and Tracking	5
2.1 Definitions	5
2.1.1 Feature point	5
2.1.2 Types of features	5
2.2 Methodology	7
2.3 Feature Detection Algorithms	10
2.4 Feature Tracking Algorithms	20
3 Navigation Sensors and Algorithms	24
3.1 GPS	24
3.2 IMU	27
3.3 Vision-Based Navigation	28
3.3.1 Challenges	29
3.3.2 Techniques	30
3.4 Vision-Aided Sensor Fusion Algorithm	35
4 Implementation and Experimental Results	42
4.1 Experimental Description	42
4.2 Vision-based Navigation Results	43
4.3 Vision-Aided Navigation Filter Results	59
5 Conclusion and Recommendations	68
REFERENCES	70
A Matlab Code for GPS Lat-Lon-Alt Conversion to NED	73
B Matlab Code for IMU Navigation	75

C Matlab Code for Vision Algorithm 78

LIST OF TABLES

Table	Page
2.1 Detection Algorithms	10
4.1 Trajectory Mean Square Error	63

LIST OF FIGURES

Figure	Page
2.1 Blobs Example (“OpenCV-Python Tutorials”, 2015)	7
2.2 Feature Types. (Webb, Prazenica, Kurdila, & Lind, 2004)	9
2.3 Window Function	11
2.4 Harris Derivatives. (“OpenCV-Python Tutorials”, 2015)	13
2.5 Shi and Tomasi. (“OpenCV-Python Tutorials”, 2015)	14
2.6 FAST. (Rosten & Drummond, 2005)	15
2.7 Scaling Problem. (“OpenCV-Python Tutorials”, 2015)	16
2.8 Scaling Problem. (Lowe, 2004)	17
2.9 SURF Box Filter. (Bay, Tuytelaars, & Van Gool, 2006)	18
2.10 SURF Orientation. (Bay et al., 2006)	19
2.11 SURF Matching. (Bay et al., 2006)	19
2.12 Brightness Constancy	21
3.1 GPS Triangulation (HyperPhysics, 2015)	25
3.2 GPS Challenges (WIDE, 2005)	26
3.3 Epipolar Geometry (“OpenCV-Python Tutorials”, 2015)	31
3.4 Image Planes Orientation (“OpenCV-Python Tutorials”, 2015)	32
3.5 Vision-Aided Navigation Filter Block Diagram	36
3.6 IMU Coordinate System (Android, 2016)	40
4.1 Cheerson CX-20 Quadcopter	43
4.2 GPS Path for Quadcopter Flight Test	44
4.3 IMU-Based Estimation Results (Quadcopter)	45
4.4 Filtering Feature Points	46
4.5 Quadcopter Snapshots with Feature Points	47
4.6 Radial and Tangential Distortion (“OpenCV-Python Tutorials”, 2015)	49

Figure	Page
4.7 SURF 8-point (Kitti dataset)	50
4.8 Harris 8-point (Kitti dataset)	51
4.9 FAST 8-point (Kitti dataset)	52
4.10 SURF MSAC (Kitti dataset)	53
4.11 Harris MSAC (Kitti dataset)	54
4.12 FAST MSAC (Kitti dataset)	55
4.13 SURF MSAC (Quadcopter dataset)	56
4.14 FAST MSAC (Quadcopter dataset)	57
4.15 GPS Tracks from the Experimental Data Sets	60
4.16 Trajectory Estimation (Quadcopter Data Set)	61
4.17 Trajectory Estimation (Kitti Data Set)	62
4.18 Estimated Attitude and Attitude Rates (Quadcopter Data Set)	65
4.19 Total Feature Points vs. Matched Feature Points	66
4.20 Matched Feature Points vs. Filtered Feature Points	67

ABBREVIATIONS

IMU	Inertial Measurement Unit
INS	Inertial Navigation System/Sensor
GPS	Global Positioning System
FAST	Features from accelerated segment test
SIFT	Scale-invariant feature transform
SURF	Speeded-Up Robust Features
LoG	Laplacian of Gaussian
DoG	Difference of Gaussian
TOA	Time Of Arrival
TOT	Time Of Transmission
TOF	Time Of Flight
SVD	Singular Value Decomposition

ABSTRACT

Saber, Ahmed MSUASE, Embry-Riddle Aeronautical University, August 2016. Vision-Aided Navigation for Autonomous Vehicles using Tracked Feature Points.

This thesis discusses the evaluation, implementation, and testing of several navigation algorithms and feature extraction algorithms using an inertial measurement unit (IMU) and an image capture device (camera) mounted on a ground robot and a quadrotor UAV. The vision-aided navigation algorithms are implemented on data collected from sensors on an unmanned ground vehicle and a quadrotor, and the results are validated by comparison with GPS data. The thesis investigates sensor fusion techniques for integrating measured IMU data with information extracted from image processing algorithms in order to provide accurate vehicle state estimation. This image-based information takes the form of features, such as corners, that are tracked over multiple image frames. An extended Kalman filter (EKF) is implemented to fuse vision and IMU data. The main goal of the work is to provide navigation of mobile robots in GPS-denied environments such as indoor environments, cluttered urban environments, or space environments such as asteroids, other planets or the moon. The experimental results show that combining pose information extracted from IMU readings along with pose information extracted from a vision-based algorithm managed to solve the drift problem that comes from using IMU alone and the scale problem that comes from using a monocular vision-based algorithm alone.

1. Introduction

Mobile vehicle state estimation algorithms depend heavily on the Global Positioning System (GPS), which does not function well in indoor environments or cluttered urban environments, and is unavailable for space missions such as exploration of asteroids, moons, or other planets. The work of this thesis uses sensors that are independent of GPS in sensor fusion algorithms with the objective of providing a similar level of accuracy as GPS-based navigation algorithms. The sensors evaluated in this thesis include inertial measurement units (IMUs) and vision sensors. The sensors are part of a consumer-grade smart phone with monocular camera. Individually, these sensors do not typically provide the required accuracy. For example, IMU-based estimates of position and orientation, which require integration of noisy sensor measurements, are subject to drift over time. Vision-based state estimation has been proven to be quite effective for estimating angular motion, but it also suffers from a scale ambiguity issue because the range to features cannot typically be determined from a sequence of two-dimensional images. Therefore, the goal of this research is to minimize the error in the estimated position and orientation of a mobile vehicle using sensor fusion techniques that will integrate the IMU data and processed image data. In addition to improved navigation in GPS-denied environments, this work is also applicable to the mapping of unknown environments.

1.1 Literature Review

Previous work by (Webb et al., 2004) is used as a basis for the evaluation of algorithms discussed in this thesis. This work includes a vision-based approach that implements an implicit extended Kalman filter (IEKF), which is a variation of the extended Kalman filter that has been modified to accommodate measurements that are implicit functions of the vehicle states (Soatto, Frezza, & Perona, 1996). These implicit measurements are based on tracked feature points (corners) and the epipolar constraint (Prazenica, Hielsberg, Sharpley, & Kurdila, 2013). Another experiment conducted by (Jin, Favaro, & Soatto, 2001) employed the IEKF for estimating the three-dimensional motion of an object from a sequence of projections. The paper identifies nonlinear implicit system parameters and provides dynamic state estimation of the object (Soatto et al., 1996). It is important to note that, while this thesis will build on this related work, these previous studies did not consider the integration of IMU data into the state estimation filter, which is a key aspect of this thesis work. The work of (R. Hartley & Zisserman, 2003) provides the basis for the visual odometry step in the vision-aided navigation algorithm.

Integration of vision and inertial sensor data is still an active research field. The proposed work can be considered as an extension to the work done by (Sirtkaya, Seymen, & Alatan, 2013) which utilized a loosely coupled Kalman filter that uses vision (stereo) pose estimation as the measurement step for the filter and IMU sensor data for the model propagation step instead of an assumed kinematic model. The

latter work was only used to estimate the 2D position of a Kitti benchmark dataset (Geiger, Lenz, Stiller, & Urtasun, 2013) which corresponded to an image sequence obtained from a ground vehicle along with inertial sensor readings. SLAM-based implementations by (Weiss, Achtelik, Lynen, Chli, & Siegwart, 2012) provided another approach to address the scale problem by using a speed-controller to correct the drift error in Parallel Tracking and Mapping (PTAM) (Klein & Murray, 2007) (a variation of Visual SLAM (Thrun & Leonard, 2008)) and self-calibration of camera-inertial sensors on a MAV; however results were not shown for a maneuvering trajectory, only for hovering in place. The work done by (Kelly, Saripalli, & Sukhatme, 2008) provided more practical test results; however the cost of the sensor used was very high and the system utilized a stereo vision odometry configuration in contrast to the monocular and consumer-grade sensors used in this thesis. Similar work was done by (Chambers et al., 2014) using an unscented Kalman filter (UKF), which uses a deterministic sampling approach to propagate the model instead of linearization of the nonlinear model used in the EKF, and stereo camera pair to estimate position in the case where GPS was unavailable. Another system was implemented by (Fraundorfer et al., 2012) that utilized a forward-looking stereo camera for path planning and exploration along with a downward-looking monocular camera for autonomous landing and small scale mapping to replace a laser sensor that put many limitations on the weight and power required for MAV and UAV systems. (Mourikis & Roumeliotis, 2007) researched a tightly-coupled EKF implementation of vision-inertial fusion by augmenting vision pose information in the process model with IMU data and used

feature points when available for the update step, but this approach makes the EKF estimator more computationally expensive.

The work presented in this thesis combines some of the methods mentioned above along with enhancements to the feature tracking step by employing some of the work done by (Jin et al., 2001), which tried to enhance the outliers rejection mechanism, which yielded better estimation of camera motion. Also, the work done by (Shen, Mulgaonkar, Michael, & Kumar, 2013) is very similar to the intended goal of using a monocular camera (aside from using another fish-eye camera) with IMU sensors for MAV and UAV systems. While stereo vision does not suffer from the scale ambiguity problem in monocular cameras, it works poorly on distant features, which makes it less capable of retrieving robust information about camera motion.

1.2 Statement of Hypothesis

Applying sensor fusion techniques that combine IMU data with image-based feature extraction data will decrease the error in vehicle state estimation in GPS-denied environments compared to using an IMU or vision-based algorithm alone. The accuracy of the developed algorithms can be validated using sensor data collected from a ground vehicle and an unmanned aerial vehicle. In vehicle experiments, GPS data can be collected for the sole purpose of validating the navigation algorithms developed in this thesis.

2. Feature Detection and Tracking

2.1 Definitions

2.1.1 Feature point

A feature point in computer vision can be defined as a region/point of interest (ROI) of an image. It is considered the building block for a large number of computer vision algorithms used in modern applications. An algorithm that depends on features will only perform as well as the feature detector. Feature extraction/detection aims to reduce the amount of resources required to describe a large set of data. By reducing the number of variables to analyze, less computational power and memory will be required for intended applications like classification or tracking. Feature extraction/detection is a low level image processing operation, which is usually the first operation performed on an image. Filtering (e.g. with a Gaussian filter) is often performed to smooth the image prior to feature extraction.

2.1.2 Types of features

Edges

An edge in an image represents points with a strong gradient magnitude in one direction, which can be combined together to form boundaries of regions or lines.

Corners

A corner or interest point refers to a point-like feature in an image with a local two dimensional structure. Usually the point does not have to be an actual corner. For example, it could be a bright spot on a dark background, which corresponds to an interest point that can be uniquely tracked in other image frames. The term "corner" is often generically used to describe these points.

Blobs

As the name implies, blob detection is associated with a larger image region than point-like features. It is used to detect regions in the image that are too smooth to be detected by a corner detector. A blob (Fig. 2.1) is basically a region in a digital image that differs in brightness or color from surrounding regions.

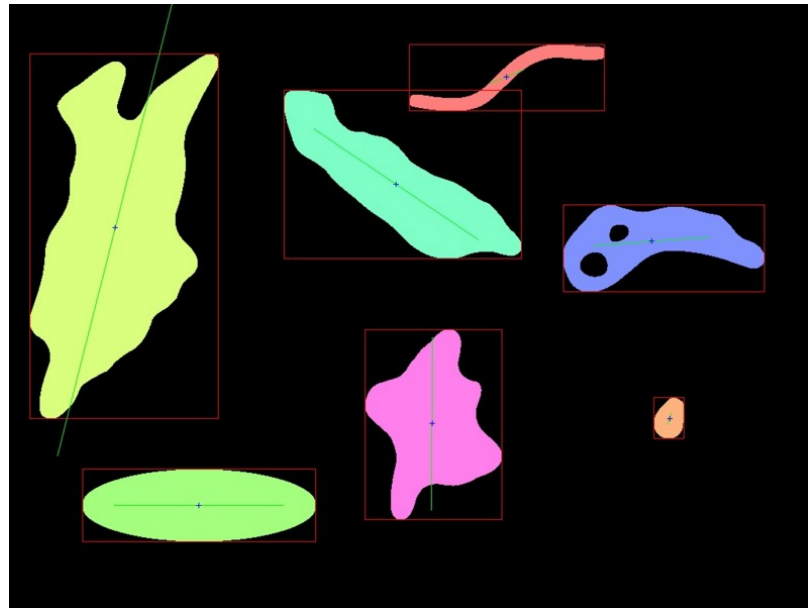


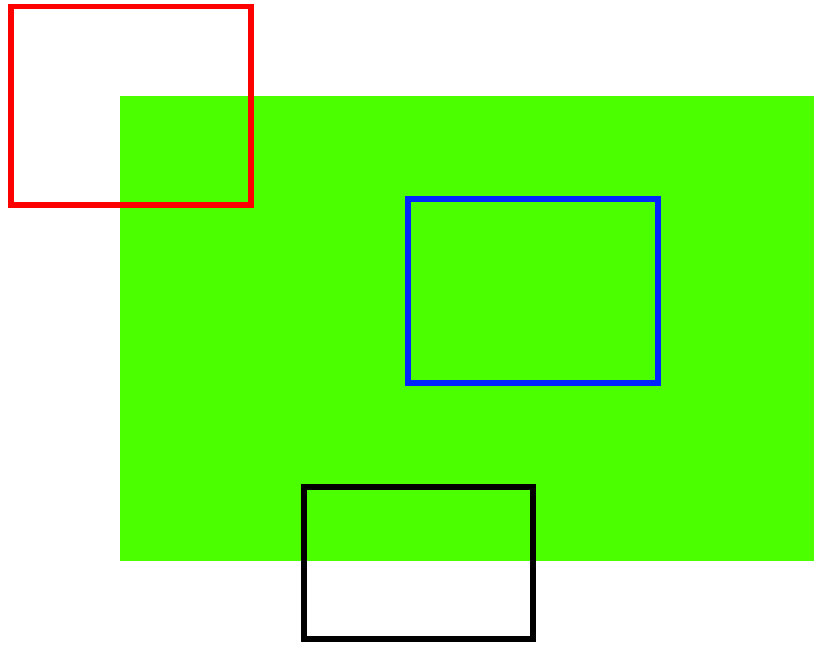
Figure 2.1. Blobs Example (“OpenCV-Python Tutorials”, 2015)

2.2 Methodology

The choice of which feature to track depends heavily on the application. In general, a feature should be invariant to morphological transforms in the image. That is scale, rotation and translation. If a feature descriptor does not change between two images after rotating, translating (in plane), and scaling (moving further/closer to) the camera, the feature point is a good choice. As can be seen in Figure 2.2a, the blue batch does not form a good feature point because moving it locally provides the same patch again, so it cannot be uniquely tracked. For the black batch, more information is provided; however, it is still not enough to uniquely identify the feature. That is moving it horizontally will give the same batch again, but in the vertical direction it will be different, which resembles an edge in the image. The red batch provides the

best feature to track, corresponding to a corner. If the batch is moved in any direction it will be different in appearance; hence it the best feature to track. It can be seen in the real image in Figure 2.2b that batches 'A' and 'B' are almost impossible to find uniquely in the image because they are repeated many times. Batches 'C' and 'D' can be found in multiple locations along the top of the building. The last two batches 'E' and 'F' can only be uniquely found once in the whole image, which makes them well suited for tracking in consecutive frames of the scene with different rotation and translation of the camera.

The next step after choosing the corners in an image is to extract or describe them. A descriptor is basically information about the feature itself which uniquely identifies it; hence the descriptor makes it possible to track the feature across consecutive frames. Many applications depend on the quality of the descriptors to be able to extract useful information that can be used for navigation, object tracking, and other tasks.



(a) Image batch types



(b) Image batches

Figure 2.2. Feature Types. (Webb et al., 2004)

2.3 Feature Detection Algorithms

There are several algorithms for detecting one or more of the aforementioned feature types. Table 2.1 shows some of the algorithms that can be used for this purpose.

Table 2.1. Detection Algorithms

Algorithm	Feature Type
Canny	Edge
Sobel	Edge
Harris	Edge, Corner
Shi & Thomasi	Corner
FAST	Corner, Blob
LoG	Corner, Blob
MSER	Blob

This discussion focuses on corner and blob detection algorithms as they are more applicable to the navigation problem investigated here. Corner detection in general is a search process using a window (patch) that is moved on the image to detect intensity changes which would indicate an edge or a corner.

Harris

Harris invented a mathematical approach as described in Eq. 2.1 for determining whether a window in an image contains an edge or a corner.

$$E(u, v) = \sum_{x,y} w(x, y) [I(x + u, y + v) - I(x, y)]^2 \quad (2.1)$$

where $w(x, y)$ is the window function defined as either a rectangular window or Gaussian window which gives weights to pixels underneath (Fig. 2.3). I is the intensity function for x, y and the x, y pixel is shifted by u, v . For nearly constant patches, the intensity difference will be near 0. For very distinctive patches, the difference will be large. Hence, the Harris detector looks for patches with large values of E .

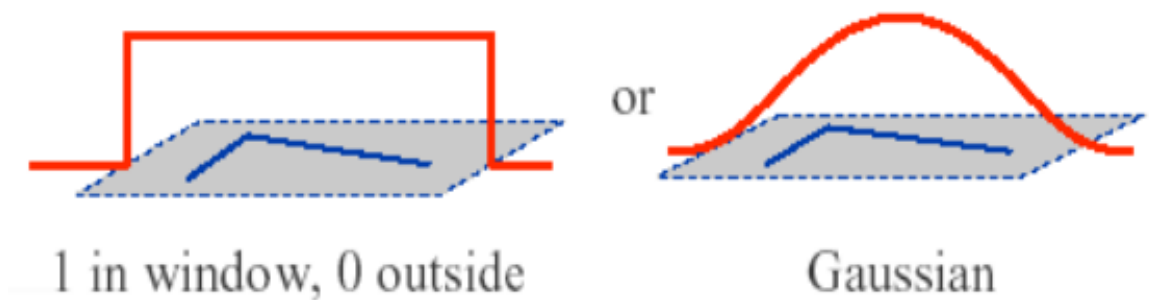


Figure 2.3. Window Function

For small shifts $[u, v]$, E can be approximated as Eq. 2.2:

$$E(u, v) \cong [u, v] M \begin{bmatrix} u \\ v \end{bmatrix} \quad (2.2)$$

where M is a 2x2 matrix computed from the image derivatives (Eq. 2.3):

$$M = \sum_{x,y} w(x,y) \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \quad (2.3)$$

where I_x, I_y are the gradient of the image in the x and y directions respectively. For each window, a score is calculated using the formula

$$R = \det(M) - k(\text{trace}(M))^2 \quad (2.4)$$

where, given the eigenvalues λ_1 and λ_2 ,

$$\det(M) = \lambda_1 \lambda_2$$

$$\text{trace}(M) = \lambda_1 + \lambda_2$$

and k is a constant between 0.04 and 0.06. Then if R is greater than a certain threshold, it is considered a corner or an edge depending on the threshold value. As can be seen in Figure 2.4, the vertical edge appears clearly in the X derivative component but does not show in the Y derivative component. For a flat image, both X and Y will not show any lines or gradients. As for corners, both the X and Y components will show the strong gradients which indicate a corner in the original image.

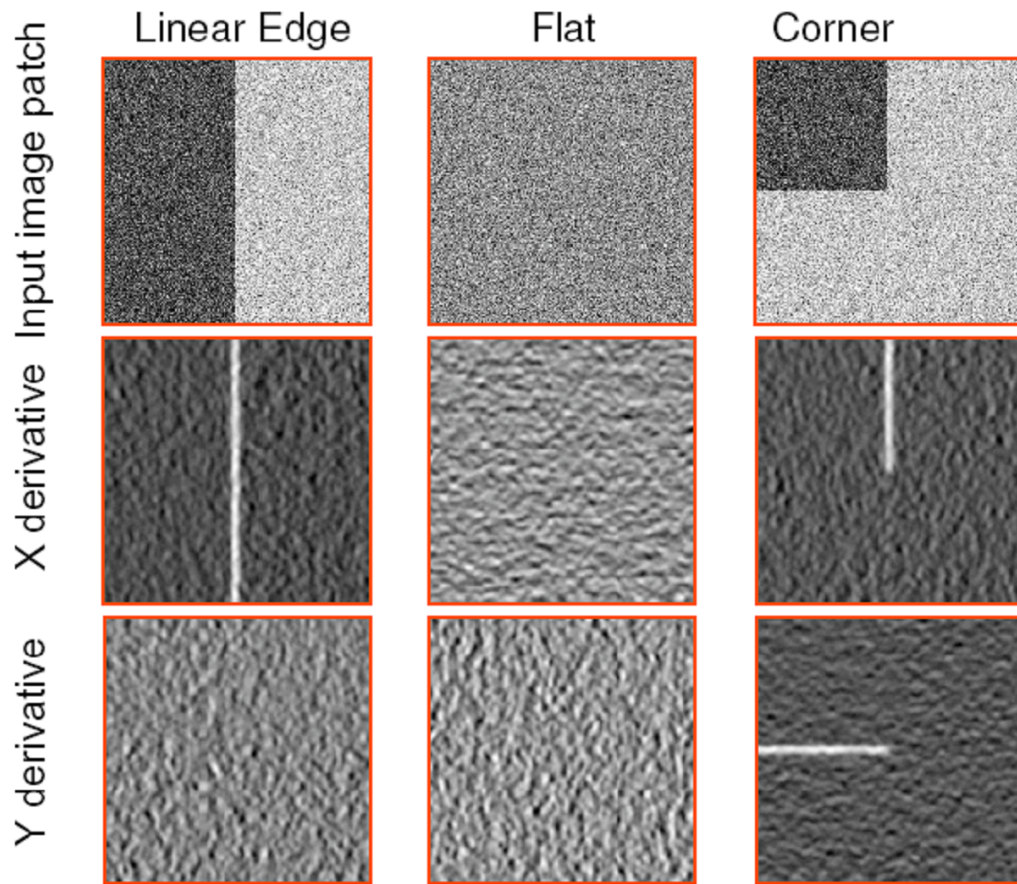


Figure 2.4. Harris Derivatives. (“OpenCV-Python Tutorials”, 2015)

Shi and Tomasi

Shi and Tomasi (1994) made a small change to the Harris algorithm in the scoring function. Instead of using Eq. 2.4, they modified it to Eq. 2.5, so as can be seen in Figure 2.5, only the green area is now considered a corner.

$$R = \min(M) = \min(\lambda_1, \lambda_2) \quad (2.5)$$

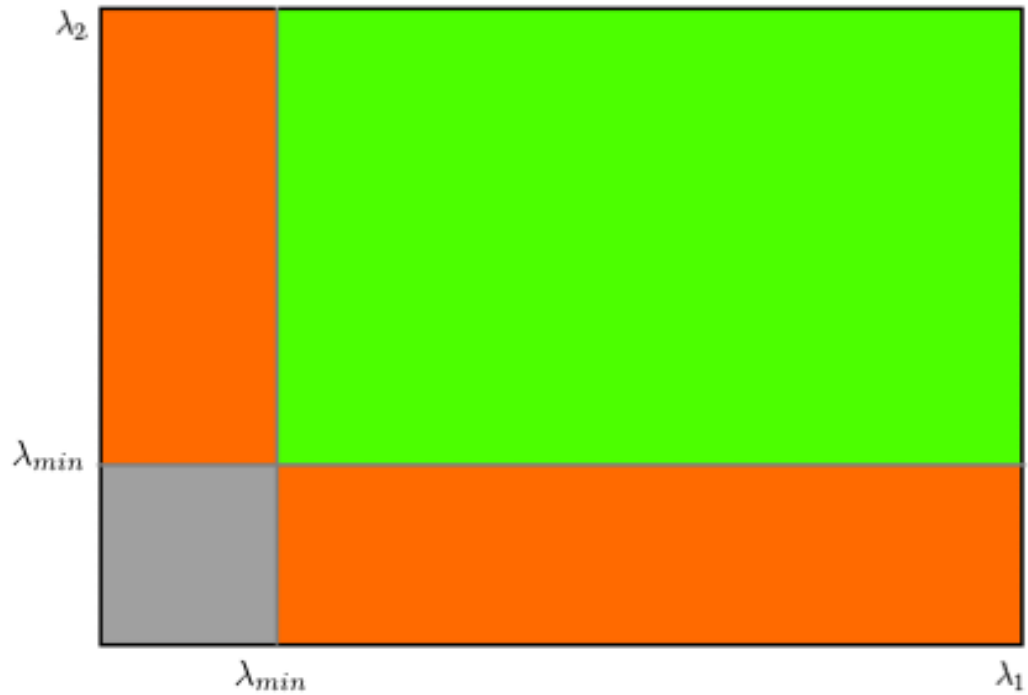


Figure 2.5. Shi and Tomasi. (“OpenCV-Python Tutorials”, 2015)

FAST

Features from accelerated segment test (FAST) is an algorithm developed by (Rosten & Drummond, 2005) with the goal to decrease the computation time for corner detection. It offers better computational efficiency over other detectors, which makes it very suitable for real-time feature tracking. The FAST operation consists of the following steps:

1. Select a pixel C in the image (see Figure 2.6) which is to be identified as an interest point or not. Let its intensity be I_c .
2. Select appropriate threshold value t .

3. Consider a circle of 16 pixels around the pixel under test.
4. The pixel C is a corner if there exists a set of n contiguous pixels in the circle (of 16 pixels) which are all brighter than $I_c + t$, or all darker than $I_c - t$.
5. A high-speed test was proposed to exclude a large number of non-corners. This test examines only the four pixels at 1, 9, 5 and 13 (first 1 and 9 are tested if they are both brighter or darker. If so, then 5 and 13 are checked). If C is a corner, then at least three of these must all be brighter than $I_c + t$ or darker than $I_c - t$.

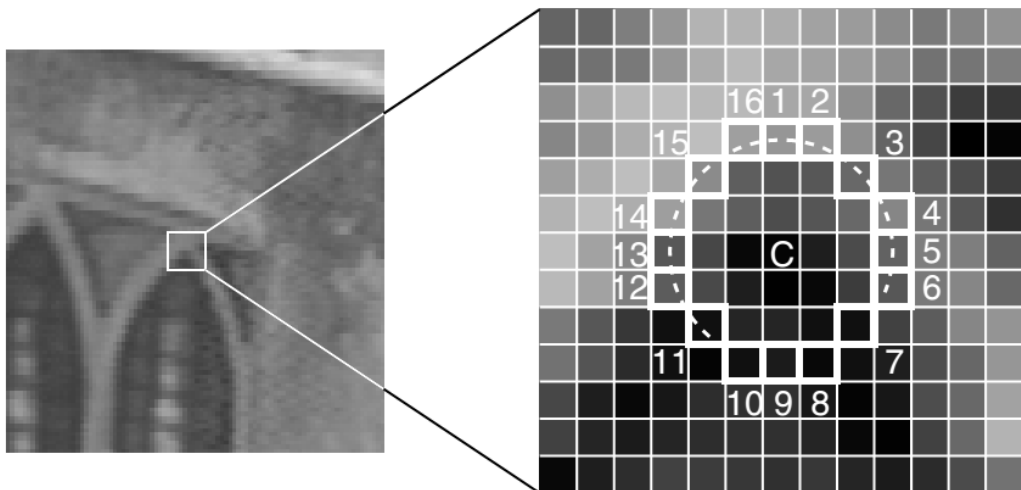


Figure 2.6. FAST. (Rosten & Drummond, 2005)

SIFT

While the previously discussed algorithms are rotation-invariant (that is, corners will still appear as corners when the image rotates), they are not scale-invariant as

can be seen in Figure 2.7. In the figure, the search window can detect the corner on the left without a problem, but if the image is scaled, the same window will no longer detect the corner.

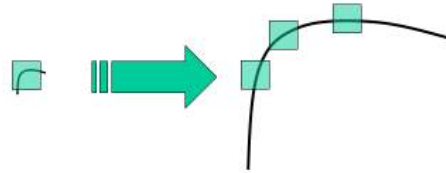


Figure 2.7. Scaling Problem. (“OpenCV-Python Tutorials”, 2015)

(Lowe, 2004) provided an algorithm to solve the scaling problem which aims to minimize the cost of feature extraction by applying the costly operations only on the locations that pass an initial test in a cascaded filtering approach. The major stages of the algorithm are:

1. **Scale-space extrema detection:** searching over all scales and image locations using the Difference of Gaussians (DoG) (Figure 2.8) method which consists of difference of Gaussian blurring of an image with two different σ ($\sigma, k\sigma$). DoG is an approximation of Laplacian of Gaussian (LoG) (Lindeberg, 1993), defined as subtraction of one blurred version of an image from another, less blurred version of the same image.

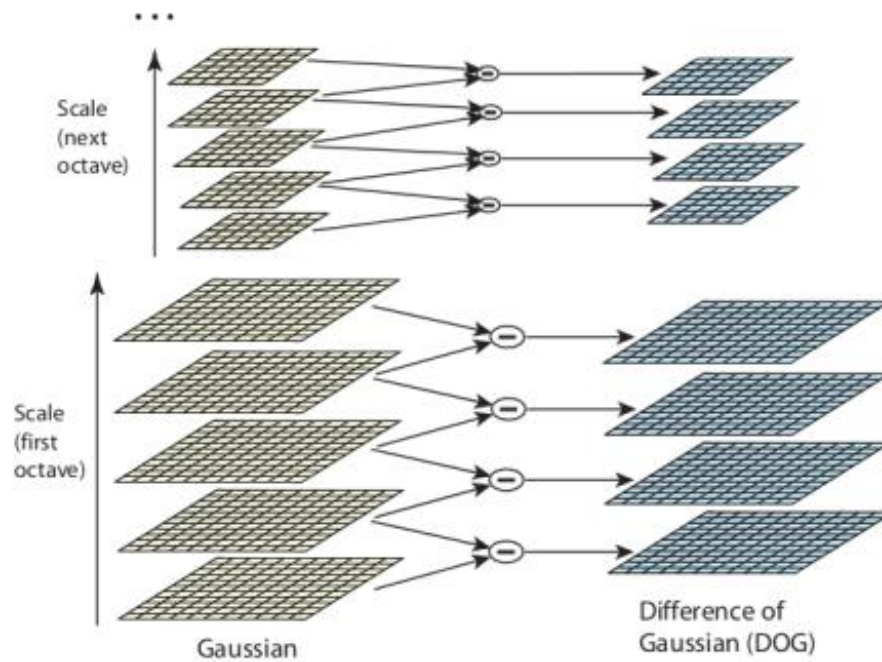


Figure 2.8. Scaling Problem. (Lowe, 2004)

2. **Keypoint localization:** selection of points is based on stability measures.
3. **Orientation assignment:** each keypoint location is assigned one or more orientation based on local image gradients.
4. **Keypoint descriptor:** the image gradients are measured at the selected scale around each point, then transformed into a representation that tolerates local shape distortion.

From the last stage, the algorithm got its name, as it transforms the image data into scale-invariant coordinates relative to local features.

SURF

The (Lowe, 2004) algorithm solved the scale problem but in a costly manner, which (Bay et al., 2006) tried to solve by an algorithm called Speeded-Up Robust Features, or in short **SURF**. Instead of DoG as an approximation of LoG that (Lowe, 2004) used, (Bay et al., 2006) employed the Box Filter method as in Figure 2.9. The main advantage of using this method is that the convolution with the box filter can be calculated with integral images in parallel on different scales.

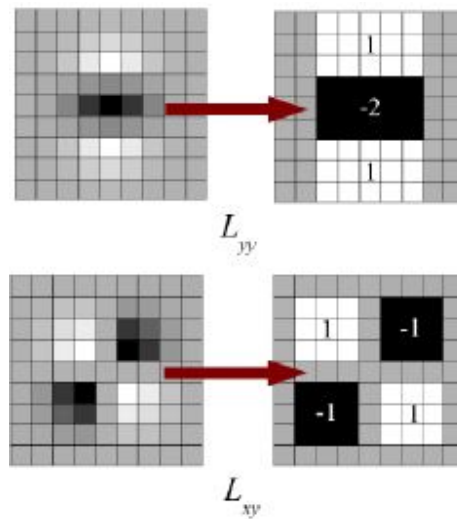


Figure 2.9. SURF Box Filter. (Bay et al., 2006)

(Bay et al., 2006) uses wavelet responses in the horizontal and vertical directions for a neighborhood of size 6 squares. These responses can be found very easily using integral images at any scale. (Bay et al., 2006) provides a way to omit finding rotation invariance, which speeds up the process and makes it robust up to ± 15 deg as illustrated in Figure 2.10.

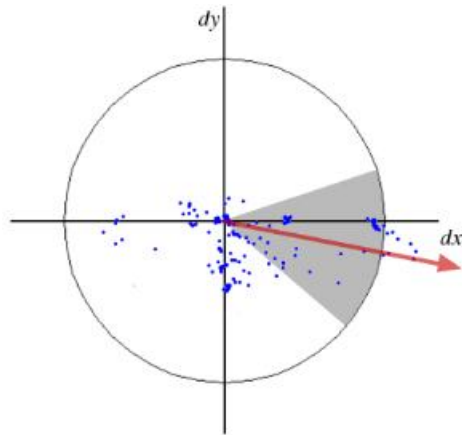


Figure 2.10. SURF Orientation. (Bay et al., 2006)

(Bay et al., 2006) added a significant improvement over SIFT by using the sign of Laplacian (the trace of M) for the underlying interest point in the matching stage (Figure 2.11).

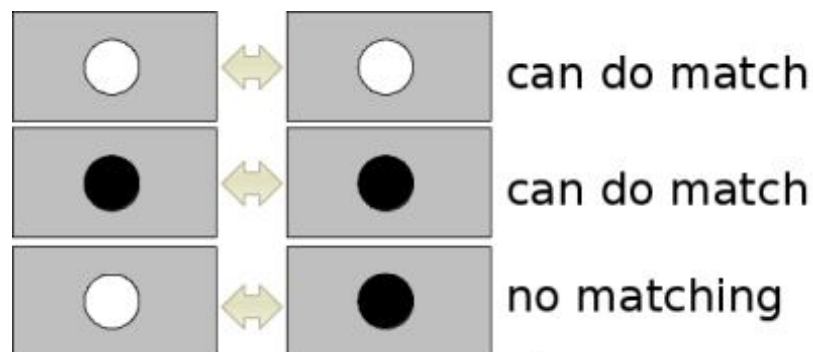


Figure 2.11. SURF Matching. (Bay et al., 2006)

(Bay et al., 2006) added optimization for each stage to improve the speed of SIFT which made it 3 times faster. However, the algorithm does not work well for view point or illumination change.

2.4 Feature Tracking Algorithms

Feature detection methods are usually combined with tracking these features over multiple frames in order to extract relevant information about camera pose or construct 3-D representations (structure from motion) of the scene. The main problem that feature tracking algorithms attempt to solve is the image registration problem (Zitov & Flusser, 2003). Image registration is the process of overlaying images (more than one) of the same scene taken at different times, from different locations and/or by different sensors (cameras). There are two main approaches for this purpose, area-based or feature-based, according to the nature of the images. Image registration can be divided into four main steps: feature detection, feature matching, transform model estimation, which defines the type and parameters of the mapping function between the reference and sensed images, and finally an image resampling and transformation step. Each step has its own set of problems and algorithms.

Challenges

Optimal output from the image registration algorithm faces a number of challenges. Some of these challenges include:

1. Choosing good features to track is essential (Shi & Tomasi, 1994) to make sure they can be matched in subsequent frames (i.e., the features should have scaling and rotation invariance).
2. Efficient tracking of points across frames.

3. Drift, which occurs due to small error accumulation in the model update.
4. Tracked points may appear/disappear due to occlusions or going outside the field of view.
5. Outlier rejection is required for optimal estimation.

KLT Tracking

The algorithm derived by Kanade-Lucas-Tomasi (KLT) (Tomasi & Kanade, 1992), which is based on the work of B. Lucas et al. (Lucas & Kanade, 1981) is the most commonly used method for feature-based tracking implementation using geometric deformation. The goal of KLT is to solve the structure from motion (SfM) problem, which means recovering scene geometry and camera motion from a sequence of images. A new method called the factorization method was introduced which can robustly estimate shape and rotation but not depth. Key assumptions of the KLT tracker (Fig. 2.12) are brightness constancy, small motion and spatial coherence.

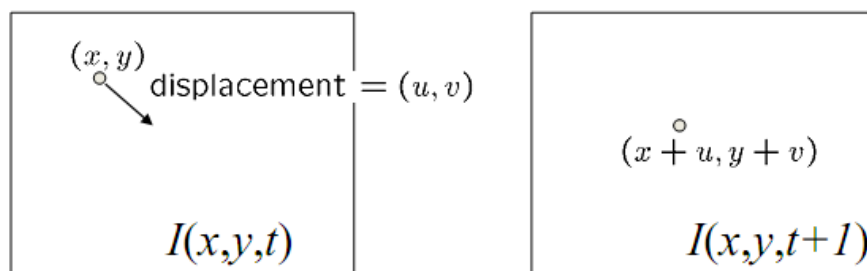


Figure 2.12. Brightness Constancy

The brightness constancy constraint can be expressed by Eq. 2.6:

$$I(x, y, t) = I(x + u, y + v, t + 1) \quad (2.6)$$

By defining $J(x) = I(x, y, t)$, and $I(x - d) = I(x + u, y + v, t + 1)$ Eq. 2.6 can be rewritten as follows in Eq. 2.7:

$$J(x) = I(x - d) + n(x) \quad (2.7)$$

where n is noise. The displacement vector d is then chosen to minimize the residue error defined by the following double integral over the given window W :

$$\epsilon = \int_W [I(x - d) - J(x)]^2 w dx \quad (2.8)$$

where w is a weighting function which can be a Gaussian-like function or in the simplest case it could be set to 1. When the displacement vector is small, the intensity function can be approximated by its Taylor series truncated to the linear term:

$$I(x - d) = I(x) - g \cdot d, \quad (2.9)$$

where g represents the image gradient. The error residue Eq. 2.8 can be written as:

$$\epsilon = \int_W [I(x) - g \cdot d - J(x)]^2 w dx = \int_W (h - g \cdot d)^2 w dx \quad (2.10)$$

where $h = I(x) - J(x)$. By differentiating Eq. 2.10 with respect to d , the minimization is obtained in closed form, which yields Eq. 2.11

$$\left(\int_W g g^T w dA \right) \mathbf{d} = \int_W h g w dA \quad (2.11)$$

Eq. 2.11 is the basic step of the tracking procedure. For every pair of adjacent frames, the left-side part in parenthesis can be computed from one frame, by estimating

gradients and computing their second order moments. The right-hand side can be computed from the difference between the two frames, which gives the solution for displacement \mathbf{d} .

Other Algorithms

Another algorithm introduced by (Jin et al., 2001) tries to minimize the cumulative error when trajectories are integrated over time by combining illumination properties with the geometric ones used by KLT. The standard cross correlation approach is also used in some applications like face matching. Cross correlation is normally used to solve the template matching problem, which is not applicable to the navigation application considered in this thesis.

3. Navigation Sensors and Algorithms

3.1 GPS

GPS was developed by the U.S. military conceptually in the late 1960's but the first GPS satellite was not launched until 1978. GPS uses specialized satellites to calculate the position of the receiver. Each satellite continuously broadcasts an epoch that is used by the receiver to calculate time of arrival (TOA). A message also is sent with the satellite location and time of transmission (TOT). The receiver needs the TOTs and TOAs from at least four satellites to compute time of flight (TOF) using speed of light and then calculate its position. The GPS satellites have atomic clocks that are synchronized together with a ground clock to ensure accurate timing. Fig. 3.1 shows an illustration of the 24 GPS satellites used to triangulate a GPS receiver using at least 3 satellites. One of them is used to measure distance to the receiver. The other two locate the detector on the intersection of two spheres, which intersect in two points. Using all three satellites, there is only one common intersection point between them, which is the receiver location.

Due to the nature of GPS, it is not available all the time under all conditions. For example, indoor environments block the satellite signals and prevent the triangulation process from being completed. Also the satellite signals are affected by weather conditions like rain or clouds that might block or scatter them. Buildings or tall ob-

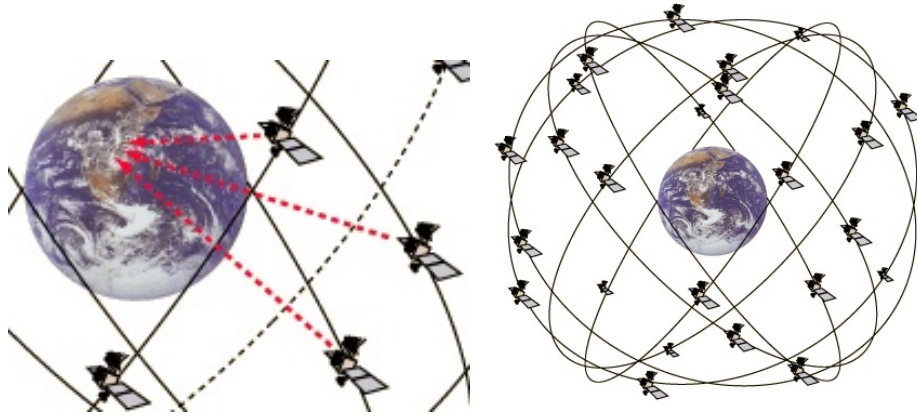


Figure 3.1. GPS Triangulation (HyperPhysics, 2015)

jects introduce a multi-path problem which can make the signal come from a different location due to reflections on these objects. They also can occlude satellite signals resulting in an insufficient number of tracked satellites, reducing accuracy or preventing a fix. The errors in reporting the location and the time from each satellite make the position calculation inaccurate. These problems can be summarized in Fig. 3.2. GPS is frequently used on mobile robots in conjunction with other sensors to detect its position. The GPS receiver update rate is relatively slower than other sensors (approximately 1s), and this gap is filled by higher update rate sensors such as an IMU. This technique is called sensor fusion and decreases the overall navigation error and adds more advantages over using a single sensor. The coordinate system used by GPS is defined by the World Geodetic System (WGS) standard. WGS 84 is the latest revision of the standard done in 1984, which puts its coordinate system origin at the Earth's center of mass with an error of less than 2 cm. The International Earth

Errors on GPS Signal

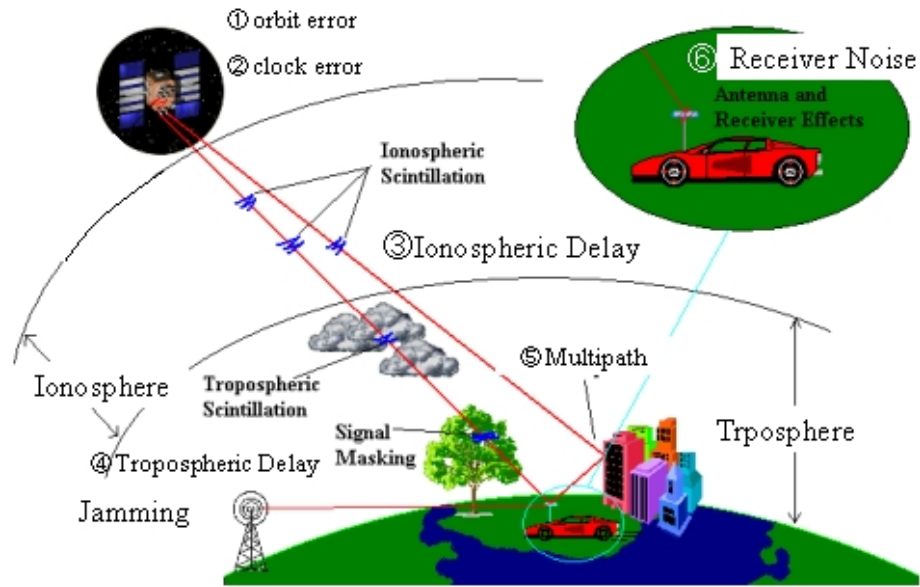


Figure 3.2. GPS Challenges (WIDE, 2005)

Rotation and Reference Systems Service (IERS) Reference meridian defines the zero longitude and latitude for the WGS 84 model.

Using the WGS 84 model, captured GPS coordinates (longitude, latitude and altitude) are converted to Northing-Easting-Down (NED) Cartesian coordinates, which are used as ground truth for our IMU-Vision filter output. A sample code for computing this conversion is provided in Appendix A. For the navigation studies performed in this thesis, GPS is used to provide ground truth data for comparison with the developed vision-aided navigation algorithms.

3.2 IMU

An Inertial Measurement Unit (IMU) is an electronic device that is used to calculate linear velocity, angular acceleration and orientation. An IMU consists of an accelerometer, a gyroscope and sometimes a magnetometer. Position can be calculated by integrating the acceleration readings twice; however it is not very accurate due to drift errors that accumulate from the integration of noisy data. An INS combines an IMU with GPS to help with position estimation when the GPS is not available (e.g., in tunnels or inside buildings). However, this approach still depends on acquiring a GPS signal at some point to correct the drift error in the IMU. This error is caused by sensor imperfections and vibrations, and when integrating the acceleration twice to compute position, this error amplifies and propagates as time progresses. An INS (Inertial Navigation System) combines a GPS receiver with an IMU utilizing the fast update rate of the IMU and minimizing the error propagation using GPS updates. INS is widely used for high accuracy applications. Without GPS however, the IMU error will keep drifting, which makes the INS inaccurate when used indoors or when GPS satellites are unavailable for an extended period of time. IMU motion estimation is accurate for a relatively small amount of time due to drift errors that accumulate due to integration of noise.

An IMU-based navigation algorithm is employed in this thesis for comparison with the vision-based and vision-aided navigation filter results. The Euler angles were estimated with integration of the rate gyro measurements. The noise was filtered

using normal integration for the first 4 gyroscope readings G as shown in Eq. 3.1.

The Runge-Kutta method was then used to integrate the remaining samples as shown in Eq. 3.2.

$$\psi_t = \psi_{t-1} + G_{\psi,t-1} \times dt, t \leq 4 \quad (3.1)$$

$$\psi_t = \psi_{t-1} + (G_{\psi,t-1} \times dt + 2 \times G_{\psi,t-2} \times dt + 2 \times G_{\psi,t-3} \times dt + G_{\psi,t-4} \times dt) / 6, t > 4 \quad (3.2)$$

This process filters the noise and gives a smoother curve. The accelerometer data are used along with the rotation angles from the gyroscope to estimate the trajectory.

$$RT_t = RT_{t-1} \times \begin{pmatrix} R_{3,3} & -T_{1,3} \\ 0 & 1 \end{pmatrix}^{-1} \quad (3.3)$$

where RT_t is a 4x4 matrix describing the homogeneous rotation and translation values in the world coordinate system. $R_{3,3}$ is a 3x3 matrix computed by concatenating three 3x3 (yaw, pitch, roll) matrices obtained via integration of the angular velocities about the IMU axes at time $t - 1$, and $T_{1,3}$ is a 3x1 vector describing the translation using linear velocities at time $t - 1$ multiplied by the time difference between the current and previous reading. The linear velocity vector is rotated into the current IMU coordinate system using rotation angles at time t .

3.3 Vision-Based Navigation

Using vision sensors for navigation is a relatively new field but it has received considerable attention from researchers. Vision sensors are considered a very good

alternative to GPS because they are more tolerant to error propagation. Vision-based navigation algorithms are mostly performed on a few image frames for each step, which makes the error more localized.

3.3.1 Challenges

There are several challenges and requirements associated with using vision sensors for navigation. These include:

- Processing speed can be a major problem due to the amount of information that is stored in each frame. Images are typically composed of two or even three dimensional arrays of data. This large amount of data makes image processing more expensive than processing other navigation sensor data.
- There is a scale ambiguity problem associated with monocular vision systems. This ambiguity causes the measured data in the image plane to be unit vectors only, with no depth information.
- Images can be corrupted by noise, which occurs due to many factors like vehicle vibrations, weather conditions, camera calibration issues, etc.
- The electrical power required to perform real-time image processing can make it difficult to implement on small mobile vehicles with limited electrical power resources.
- Feature detection and matching algorithms are very susceptible to noise.

3.3.2 Techniques

The main purpose of any vision-based navigation technique is the ability to estimate camera pose with respect to the environment using a sequence of images taken from a moving camera (monocular vision), or pair of cameras (stereo vision). This discussion focuses on classical techniques for monocular vision based on epipolar geometry.

Epipolar Geometry

Epipolar geometry is based on the essential constraint, which describes the relationship between the camera coordinates of a static feature point at two instants of time and the extrinsic properties of the camera (i.e., camera rotation and translation). The camera has to be calibrated to convert points measured in the image plane to equivalent pinhole camera measurements. These pinhole measurements take the form of unit vectors from the camera to the feature point, and the calibration process entails removing effects such as radial distortion. Figure 3.3 illustrates the epipolar constraint. Given measurements $\vec{\Pi}_r$ and $\vec{\Pi}_l$ of a static feature point in the left and right camera images, the epipolar constraint can be expressed as in Eq. 3.4:

$$\vec{\Pi}_r \cdot (\vec{T} \times R\vec{\Pi}_l) = 0 \quad (3.4)$$

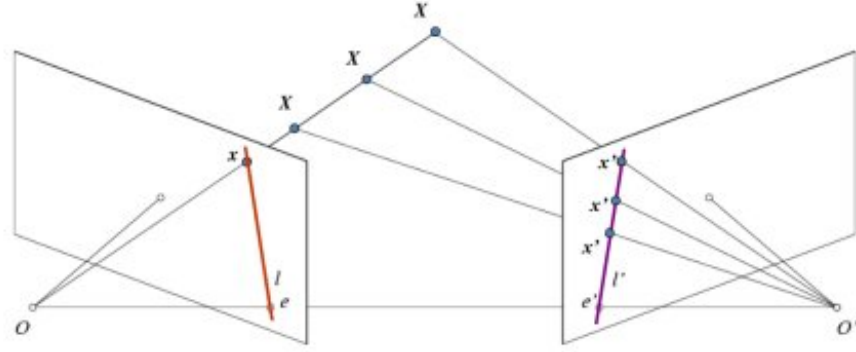


Figure 3.3. Epipolar Geometry (“OpenCV-Python Tutorials”, 2015)

where $\vec{\Pi}_r$ and $\vec{\Pi}_l$ are measured as

$$\vec{\Pi}_r = f \begin{pmatrix} u_{r,p} \\ v_{r,p} \\ 1 \end{pmatrix}, \vec{\Pi}_l = f \begin{pmatrix} u_{l,p} \\ v_{l,p} \\ 1 \end{pmatrix} \quad (3.5)$$

where f is the camera focal length, and u_p and v_p are the image coordinates for feature point p in homogeneous coordinates. The same principle can be applied to the unit vectors from the camera origin to the feature point as in Eq. 3.6:

$$\vec{O}_r \cdot (\vec{T} \times R\vec{O}_l) = 0 \quad (3.6)$$

In Eqs. 3.4 and 3.6, R represents the 3-D rotation matrix from the left image plane Π_l to the right image plane Π_r , which is computed from the frame-to-frame Roll ($d\phi$), Pitch ($d\theta$), and Yaw ($d\psi$) rotation matrices of the camera (Eqs. 3.8, 3.9 and 3.10) as follows:

$$R = R_{d\phi} R_{d\theta} R_{d\psi} \quad (3.7)$$

$$R_{d\phi} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(d\phi) & \sin(d\phi) \\ 0 & -\sin(d\phi) & \cos(d\phi) \end{pmatrix} \quad (3.8)$$

$$R_{d\theta} = \begin{pmatrix} \cos(d\theta) & 0 & -\sin(d\theta) \\ 0 & 1 & 0 \\ \sin(d\theta) & 0 & \cos(d\theta) \end{pmatrix} \quad (3.9)$$

$$R_{d\psi} = \begin{pmatrix} \cos(d\psi) & \sin(d\psi) & 0 \\ -\sin(d\psi) & \cos(d\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (3.10)$$

\vec{T} is the 3-D translation vector from the left image plane to the right image plane. \vec{T} is expressed in right image plane coordinates as shown in Fig. 3.4. Eq. 3.4 can be

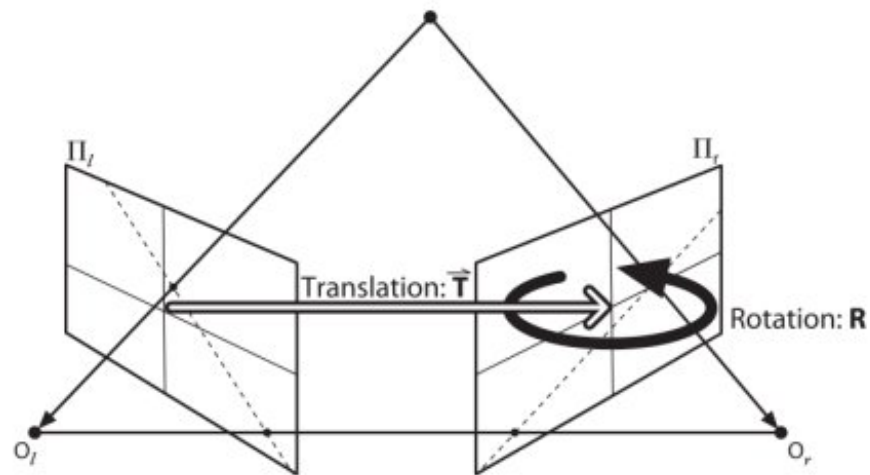


Figure 3.4. Image Planes Orientation (“OpenCV-Python Tutorials”, 2015)

applied to all measured feature points, where n_{fp} is the total number of valid feature points, resulting into Eq. 3.11:

$$\begin{pmatrix} u_{r,p} \\ v_{r,p} \\ 1 \end{pmatrix}^T E \begin{pmatrix} u_{l,p} \\ v_{l,p} \\ 1 \end{pmatrix} = 0, \quad p = 1, \dots, n_{fp} \quad (3.11)$$

where E denotes the essential matrix for feature point coordinates (fundamental matrix for normalized ones), which is defined as $\vec{T} \times R$.

These equations can be combined into a single equation (Eq. 3.12) (Soatto et al., 1996), which is the first step of the eight-point algorithm to compute the essential matrix. (Longuet-Higgins, 1981) introduced the eight-point algorithm, which finds the fundamental matrix. (R. I. Hartley, 1997) described a more practical approach to find the essential matrix which is the normalized eight-point algorithm.

$$C\vec{e} = 0, \quad C \in \Re^{n_{fp} \times 9} \quad (3.12)$$

where C_p , the p th row of C , is defined as Eq. 3.13:

$$C_p = [u_{r,p}u_{l,p} \quad u_{r,p}v_{l,p} \quad u_{r,p} \quad v_{r,p}u_{l,p} \quad v_{r,p}v_{l,p} \quad v_{r,p} \quad u_{l,p} \quad v_{l,p} \quad 1] \quad (3.13)$$

and \vec{e} is the stacked columns of E . Note that the scale ambiguity is apparent in 3.12 because, given a solution \vec{e} , any scalar multiple of \vec{e} would also satisfy the constraint. This implies that the translation vector \vec{T} can only be determined up to a scale factor (i.e., only the unit vector of translation can be computed).

In theory, the coplanarity constraint is true for all feature points, which makes the solution for E trivial given enough feature points (at least five). However, this is

not always practical in real world scenarios due to environment variables (like noise, vehicle vibration, etc) that affect the quality of the measured feature points. Another problem with the regular eight-point algorithm is that feature point coordinates are not necessarily normalized; hence the values of the first two coordinates have a much larger range than the third one for each feature point. Hartley proposed a way to transform the coordinate system of the feature points to normalize them, which makes Eq. 3.12 better-conditioned in practical use. With this approach, solving the equation requires at least 8 feature points because there are 9 equations (with 1 trivial equation). The resulting matrix may not satisfy the epipolar constraint due to noise in the feature points coordinates. Therefore, the last step of the algorithm is to find a matrix E' which minimizes the error of the resulting matrix E_{est} . The singular value decomposition of E_{est} is applied as in Eq. 3.14 :

$$E_{est} = USV^T \quad (3.14)$$

where U, V are orthogonal matrices and S is a diagonal matrix which contains the singular values of E_{est} . To compute E' , the largest two singular values s_1, s_2 of S are used to form S' in Eq. 3.15 :

$$S' = \begin{pmatrix} (s_1 + s_2)/2 & 0 & 0 \\ 0 & (s_1 + s_2)/2 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad (3.15)$$

Then E' is formed as in Eq. 3.16 :

$$E' = US'V^T \quad (3.16)$$

Rotation and Translation from Essential Matrix

Using the SVD of E , there are 4 possible solutions for rotation and translation, with only 1 physically feasible solution, as shown in Eq. 3.17

$$R_1 = UWV^T, R_2 = UW^TV^T, \vec{T}_{1,2} = \pm u_3 \quad (3.17)$$

where u_3 is the last column of the U matrix and W is defined as Eq. 3.18

$$W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (3.18)$$

The physically feasible solution is determined by testing the feature points against each one of the solutions. Only one combination of R and \vec{T} will project all points in front of the camera. Testing for the valid combination is done by back-projecting each pair of the feature points using each of the four combinations. If any of the 3D projected points have a negative depth value, the combination is rejected because this implies that the point exists behind the camera, which is not physically possible. Sometimes due to noise, all of the four combinations give negative depth values for some of the points. In this case, the combination is chosen that gives the highest percentage of positive depth values.

3.4 Vision-Aided Sensor Fusion Algorithm

An algorithm has been implemented for camera pose estimation which can be divided into two main stages. A block diagram of the algorithm is shown in Fig. 3.5.

An extended Kalman filter was used to integrate the vision-based motion estimation (using the epipolar constraint and the 8-point algorithm) with the IMU data. The IMU data were used in the propagation model as they give better accuracy than a regular kinematic model by utilizing measured vehicle states at each instance of time. The state estimation vector used in the Kalman filter is shown in Eq. 3.19.

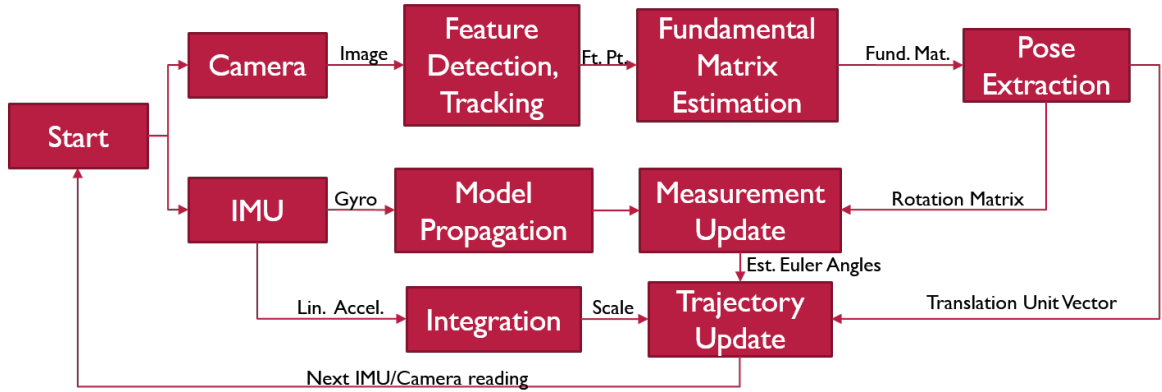


Figure 3.5. Vision-Aided Navigation Filter Block Diagram

$$X_{est} = [d\phi, d\theta, d\psi] \quad (3.19)$$

d_ψ , d_θ and d_ϕ are the rotation angles deltas (*rad*) around the x-axis, y-axis and z-axis in the camera frame between the previous and current timestamp. Only delta angles are estimated, and not the total values of the angles, to eliminate the error propagation from affecting the calculations. The EKF is divided into two main steps, a model propagation step and a measurement update step. The state estimates from the EKF at each time step are then used to update the inertial position and orientation of the vehicle.

Propagation Step

The IMU measurements are used to propagate the state estimation vector. The state is propagated using the rate gyro data as follows:

$$d\phi_t = G_{\phi,t-1} \times dt \quad (3.20)$$

$$d\theta_t = G_{\theta,t-1} \times dt \quad (3.21)$$

$$d\psi_t = G_{\psi,t-1} \times dt \quad (3.22)$$

The estimation error covariance matrix is propagated using

$$\tilde{P}_t = A\hat{P}_{t-1}A^T + Q \quad (3.23)$$

where \hat{P}_{t-1} is the covariance matrix from the EKF at the previous time step, Q is a 3x3 matrix representing the process noise, and A is a 3x3 Jacobian matrix resulting from linearization of the state propagation model:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (3.24)$$

Vision-Based Measurement Update

The second step in the extended Kalman filter is the measurement update. In this EKF implementation, the measurement update entails computing the pose vector using the vision-based algorithm as shown in Eq. 3.28. Due to the different update rate between the IMU (100ms) and camera (33.3ms), a post-processing step is done

to synchronize the camera frames to the IMU timestamp. The measurement is then computed between the camera frame in the previous and current IMU reading timestamps. The synchronization step assigns the closest camera frame timestamp to the IMU reading timestamp and ignores in-between frames. The vision-based algorithm is used to compute R from which the measurement can be extracted:

$$d_{\phi,m} = \text{atan2}(-R_{2,3}, R_{2,2}) \quad (3.25)$$

$$d_{\theta,m} = \text{asin}(R_{2,1}) \quad (3.26)$$

$$d_{\psi,m} = \text{atan2}(-R_{3,1}, R_{1,1}) \quad (3.27)$$

$$X_m = [d_{\phi,m}, d_{\theta,m}, d_{\psi,m}] \quad (3.28)$$

The Kalman gain matrix is updated using Eq. 3.29:

$$K_t = \tilde{P}_t C^T (C \tilde{P}_t C^T + R_m)^{-1} \quad (3.29)$$

The state estimate is then updated as follows:

$$X_{est} = X_p + K_t(X_m - X_p) \quad (3.30)$$

The estimation error covariance matrix is then updated using Eq. 3.31:

$$\hat{P}_t = [I - K_t C] \tilde{P}_t \quad (3.31)$$

C is 3x3 measurement matrix, and R_m is a 3x3 measurement noise matrix. Both Q and R_m are tuned to give more weight to the measurement update over the propagation model as shown in Equations 3.32 and 3.33:

$$Q = \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{pmatrix} \quad (3.32)$$

$$R_m = \begin{pmatrix} 0.05 & 0 & 0 \\ 0 & 0.05 & 0 \\ 0 & 0 & 0.05 \end{pmatrix} \quad (3.33)$$

The measurement matrix C is computed as

$$C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (3.34)$$

Inertial Position and Orientation Update

Following the EKF at each time step, the estimated angles deltas ($d\phi$, $d\theta$, $d\psi$) are then used in conjunction with the scale factor computed from the current velocity to update the inertial trajectory of the vehicle. The inertial rotation matrix $R_{I,t-1}$ from the previous time step is used to transform the previous IMU acceleration reading from the IMU-based reference frame coordinate system (shown in Figure 3.6) to the world coordinate system as shown in Eq. 3.35. The velocity estimate is updated by multiplying the acceleration vector by the time difference between the current and

previous IMU readings. The magnitude of the velocity estimate is used to estimate the transition scale factor as shown in Eq. 3.37. Inertial position and orientation estimates are then updated as shown in Equation 3.38, which uses homogeneous coordinates. The frame-to-frame rotation matrix R in 3.38 is computed using the estimated angle deltas $(d\phi, d\theta, d\psi)$ from the EKF.

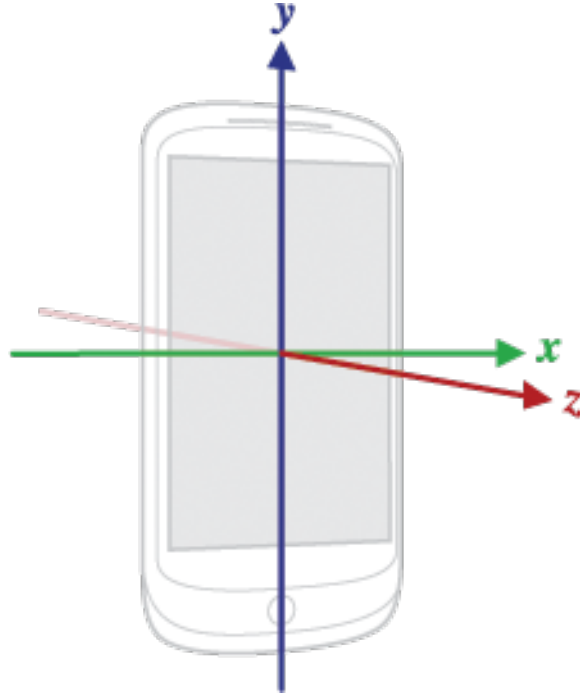


Figure 3.6. IMU Coordinate System (Android, 2016)

$$a_{t-1,world} = (R_{I,t-1})^T \times a_{t-1,r} \quad (3.35)$$

$$v_{t,world} = v_{t-1,world} + a_{t-1,world} \times dt \quad (3.36)$$

$$Scale_t = |v_{t,world}| \times dt \quad (3.37)$$

$$(R_I T_I)_t = (R_I T_I)_{t-1} \times \begin{pmatrix} R & -T \times Scale_t \\ 0 & 1 \end{pmatrix}^{-1} \quad (3.38)$$

Eq. 3.38 provides the updated inertial position estimate $T_{I,t}$ as well as the rotation matrix $R_{I,t}$, which represents the orientation of the vehicle relative to the inertial frame. The roll, pitch, yaw Euler angles can then be extracted as follows:

$$\phi = atan2(-R_I(2, 3), R_I(2, 2)) \quad (3.39)$$

$$\theta = asin(R_I(2, 1)) \quad (3.40)$$

$$\psi = atan2(-R_I(3, 1), R_I(1, 1)) \quad (3.41)$$

4. Implementation and Experimental Results

4.1 Experimental Description

The data used to test the navigation algorithms were captured using an Android-based smart phone (HTC-one M8) with 2 GB RAM, Qualcomm Snapdragon 801 quad-core 2.3 GHz CPU, high-resolution camera (1920x1080) at 30 fps, accelerometer with maximum range of 19.61 m/s^2 and a resolution of 0.01 m/s^2 , gyroscope with maximum range of 2291.8 deg/s and 0.6 deg/s resolution, magnetometer with $200 \mu\text{T}$ maximum range and $0.01 \mu\text{T}$ resolution, and a GPS sensor with a maximum error of 5 meters. Sensor properties were acquired using an Android app installed on the phone. The phone was mounted on a Cheerson CX-20 quadcopter as shown in Fig. 4.1. The quadcopter weighs 980 grams (2.2 lbs) without the phone and battery (SpecOut, 2016). It has a span of 509 mm (20 inches), and its dimensions are 360x360x200 mm. The quadcopter is equipped with GPS for a return to home (RTH) feature. With a 2700 mAh/11v LiPo (180 grams) battery, the Cheerson CX-20 can average 15 minutes of flight time without payload (phone/camera). It comes with a GoPro camera mount, which has been altered to hold a smart-phone mount. It has a maximum speed of 8 m/s or 18 mph. A 2.4 Ghz wireless controller is used to control the quadcopter with an operating range of 1500 meters. Testing was conducted in Losco Regional Park in Jacksonville, Florida. Fig. 4.2 shows a 30 seconds part of the

GPS route of the test. The Kitti dataset (Geiger et al., 2013) was also used for the vision algorithm validation. The Kitti dataset was collected using a standard station wagon with two high-resolution color and grayscale video cameras. Accurate ground truth was provided by a Velodyne laser scanner and a GPS localization system.



Figure 4.1. Cheerson CX-20 Quadcopter

An IMU-based navigation solution was computed for the quadcopter data set. As shown in Figures 4.3a and 4.3b, the results show that the IMU-based trajectory starts by following the correct path matching the GPS locations; however it starts to drift away over time.

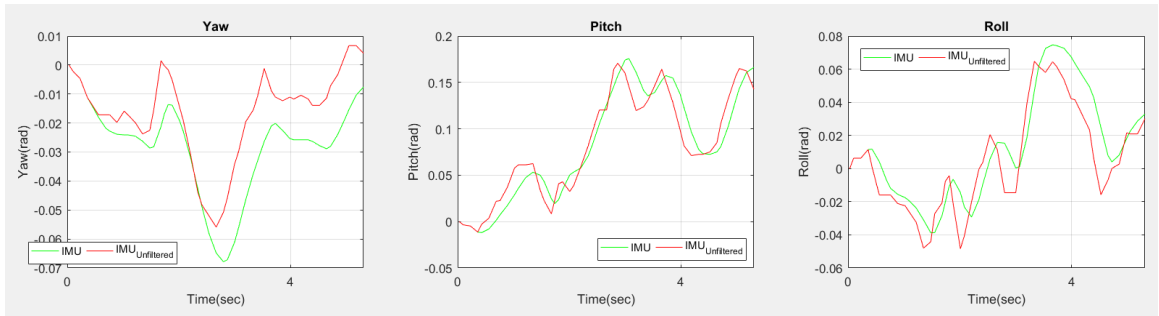
4.2 Vision-Based Navigation Results

The vision-based navigation algorithm starts by extracting initial feature points using any of the methods mentioned in Chapter 2. The FAST, SURF, and Harris corner detectors were all implemented in the vision-based navigation algorithm to determine the best choice of feature detection algorithm. During testing, the SURF method provided the best results for both the Kitti dataset and the quadcopter data

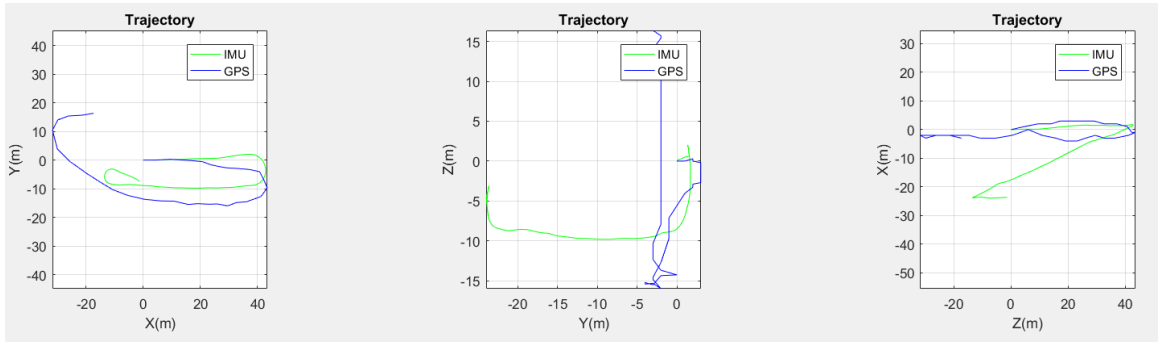


Figure 4.2. GPS Path for Quadcopter Flight Test

as shown in Figures 4.14 and 4.12. The FAST method provided similar results in the Kitti dataset but did not perform as well for the quadcopter test data (Figures



(a) Yaw



(b) Trajectory

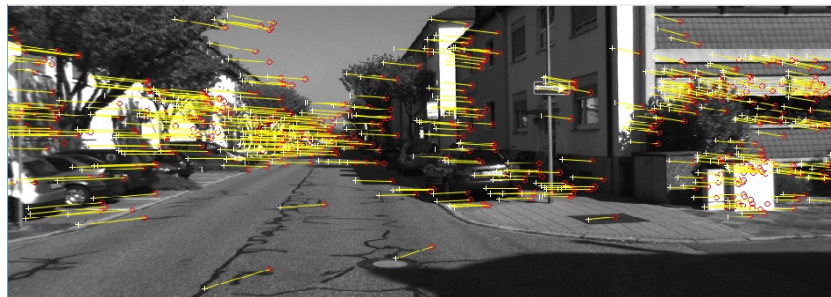
Figure 4.3. IMU-Based Estimation Results (Quadcopter)

4.13 and 4.10). The detected feature points are used to initialize the KLT tracker, which tracks the feature points in subsequent frame(s). A threshold was implemented to enforce updating the tracker points when there are not enough inliers for the fundamental matrix estimator.

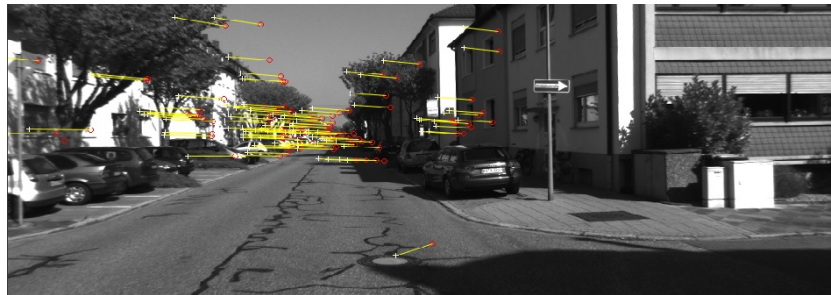
Filtering of Tracked Features

One important addition to the algorithm is a safeguard against the case of no (or little) motion between image frames. This mechanism checks for the distance

between each pair of tracked points against the mean distance of all tracked points. The standard deviation of this measure in the case of no motion will be very small; hence the current frame is skipped from further processing if there are not enough points after pruning feature points with small distances as illustrated in Fig. 4.4. This measure also removes features tracked near the image vanishing point, which is defined as the intersection point of the projection of a set of parallel lines in space on the image plane. This filtering process provides the next step of the algorithm with good features to estimate the fundamental matrix.



(a) Unfiltered feature points (835 points)

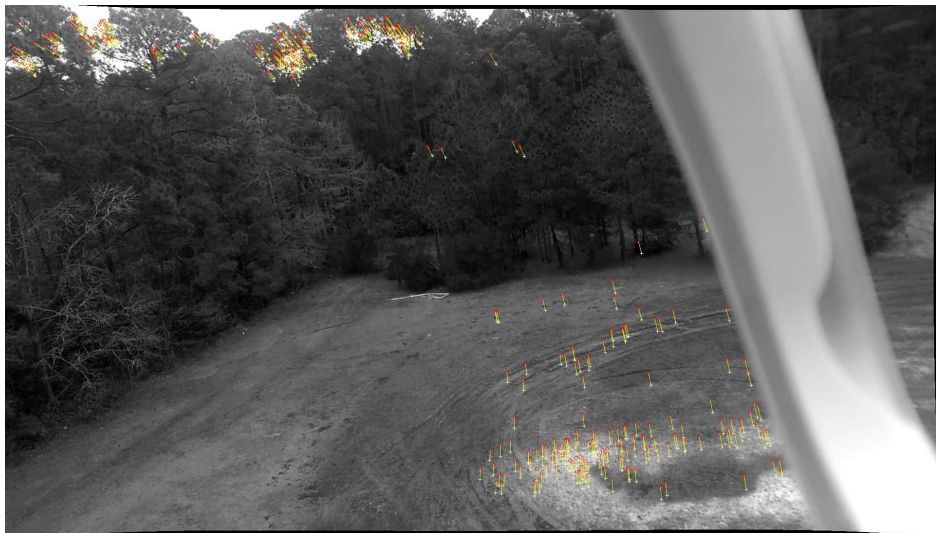


(b) Filtered feature points (249 points)

Figure 4.4. Filtering Feature Points



(a) Quadcopter snapshot (15th second)



(b) Quadcopter snapshot (23th second)

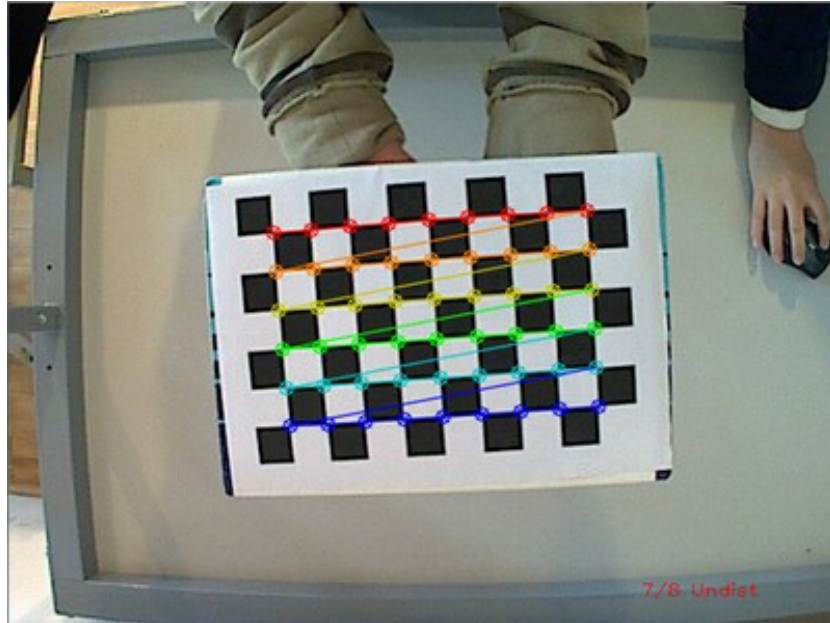
Figure 4.5. Quadcopter Snapshots with Feature Points

Fundamental Matrix Estimation

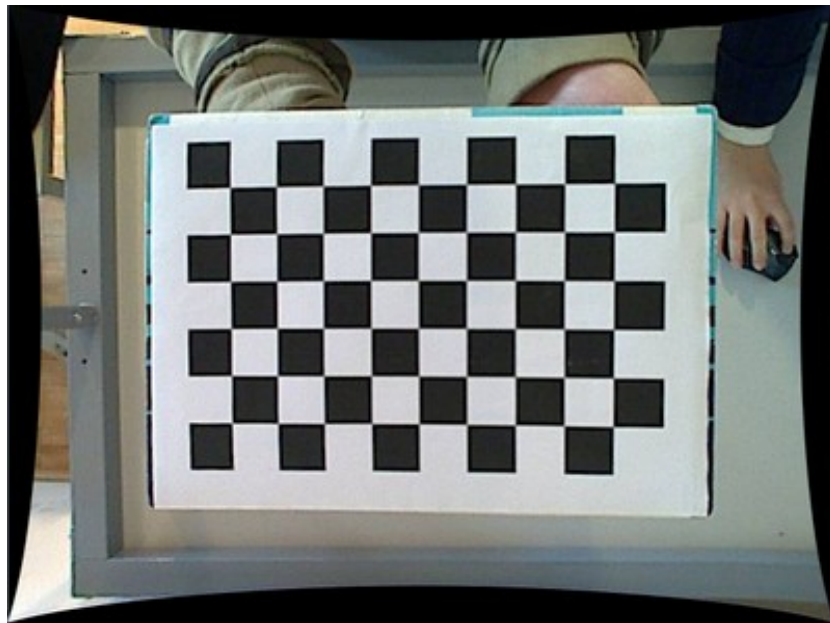
Even though the filtered points resulting from the last step are sufficient for motion estimation, there can still be some outlier points which cause the regular eight-

point algorithm to compute inaccurate estimates of the essential matrix, which in turn results in poor approximation of the camera pose, as illustrated in Figures 4.7, 4.8 and 4.9. In the regular 8-point algorithm, all feature points (maximum of 1000 points) are used to compute the fundamental matrix. Camera frames are normalized using the camera calibration parameters prior to computation of the fundamental matrix. The Kitti dataset was captured using a camera with 718.856 focal length and (607.1928, 185.2157) principal point. The HTC smartphone that was used on the quadcopter has a camera with 1512 focal length and (947.76, 541.55) principal point. Radial lens distortion values were estimated for the smart-phone camera using the MATLAB cameraCalibrator tool to (0.159827593819772, -0.359668203419746) along with tangential distortion values of (-0.000480568550162411, -0.00259254233756110). The radial and tangential distortion calibration yields a camera correction corresponding to the image shown in Figure 4.6. A chess board with a 6x9 array of 25mm squares was used as the calibration pattern.

Using random selection algorithms in combination with the 8-point algorithm makes it more robust and less affected by outliers points, as shown in Figures 4.10 and 4.13 for 1000 trials. Random sample consensus (RANSAC) is an iterative method to estimate the parameters of a mathematical model from a set of observed data which contains outliers. Using a variation of RANSAC, M-estimator Sample Consensus (MSAC), an initial fundamental matrix F is set to zero. Then a number of fundamental matrices f (N trials) are estimated using 8 randomly selected pairs of feature points using the normalized 8-point algorithm. In each trial, the fitness of



(a) Original Image



(b) Undistorted Image

Figure 4.6. Radial and Tangential Distortion (“OpenCV-Python Tutorials”, 2015)

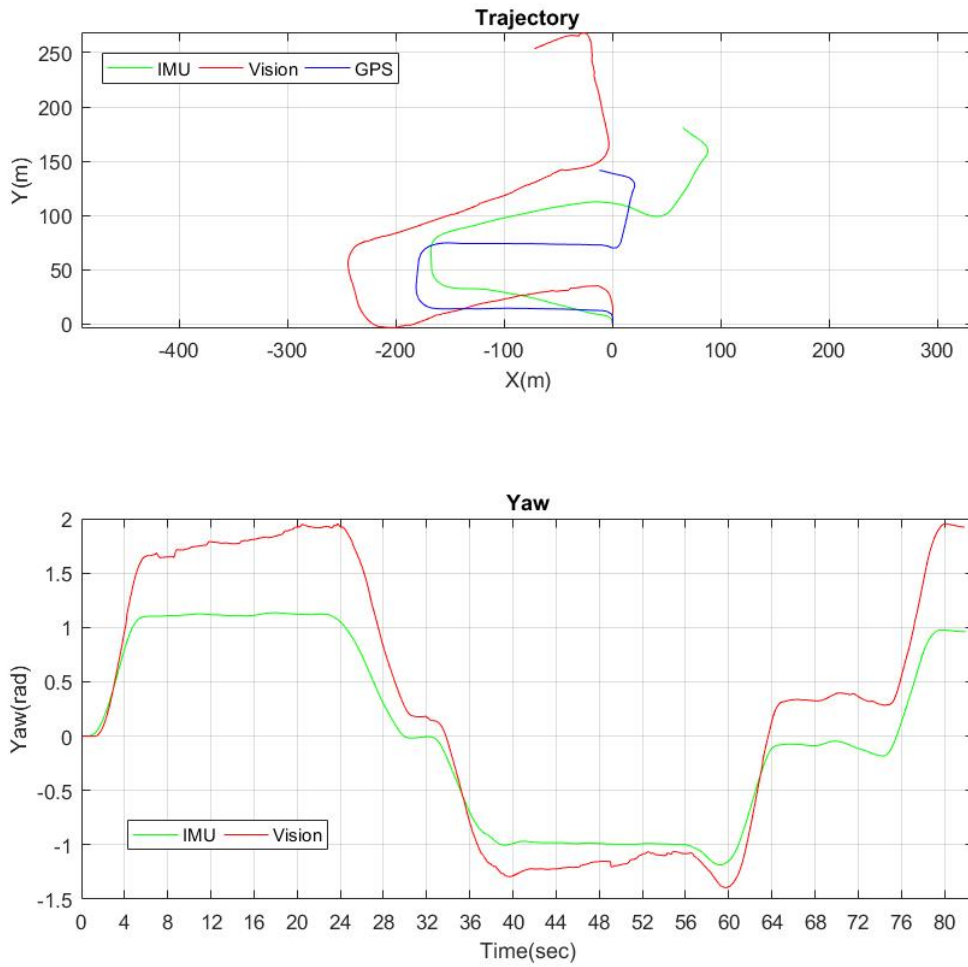


Figure 4.7. SURF 8-point (Kitti dataset)

the estimated f is computed for all feature points (u_i, v_i) using the following fitness function:

$$\sum_i^{n_{fpt}} \min(d(u_i, v_i), t) \quad (4.1)$$

where t is a specified threshold and n_{fpt} is the total number of feature points. If the fitness of f is less than the fitness of F where F denotes the current best estimate of

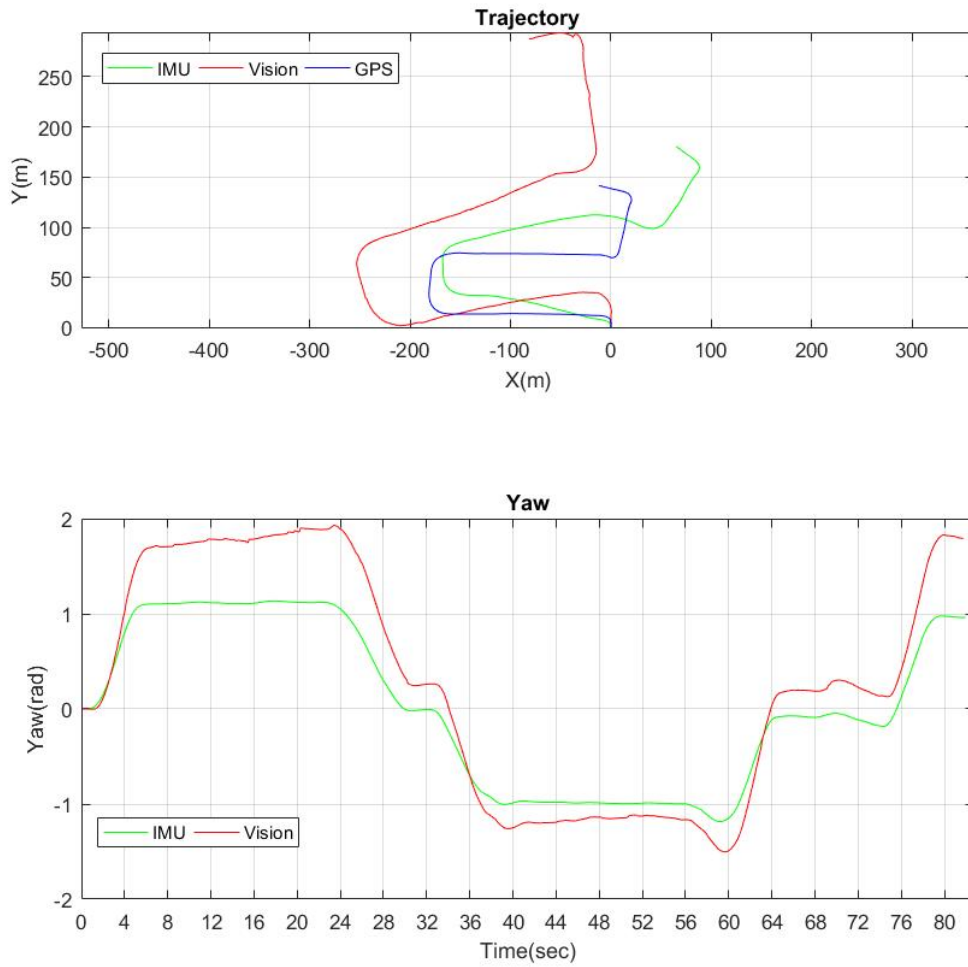


Figure 4.8. Harris 8-point (Kitti dataset)

the fundamental matrix, f is considered a better estimate and is used for evaluating the next trials. The distance d is computed using the Algebraic method defined as

$$d(u_i, v_i) = (v_i F u_i^T)^2 \quad (4.2)$$

MSAC is used because it generally converges much quicker than RANSAC. The generated F matrix gives a better estimation of camera motion between each pair of

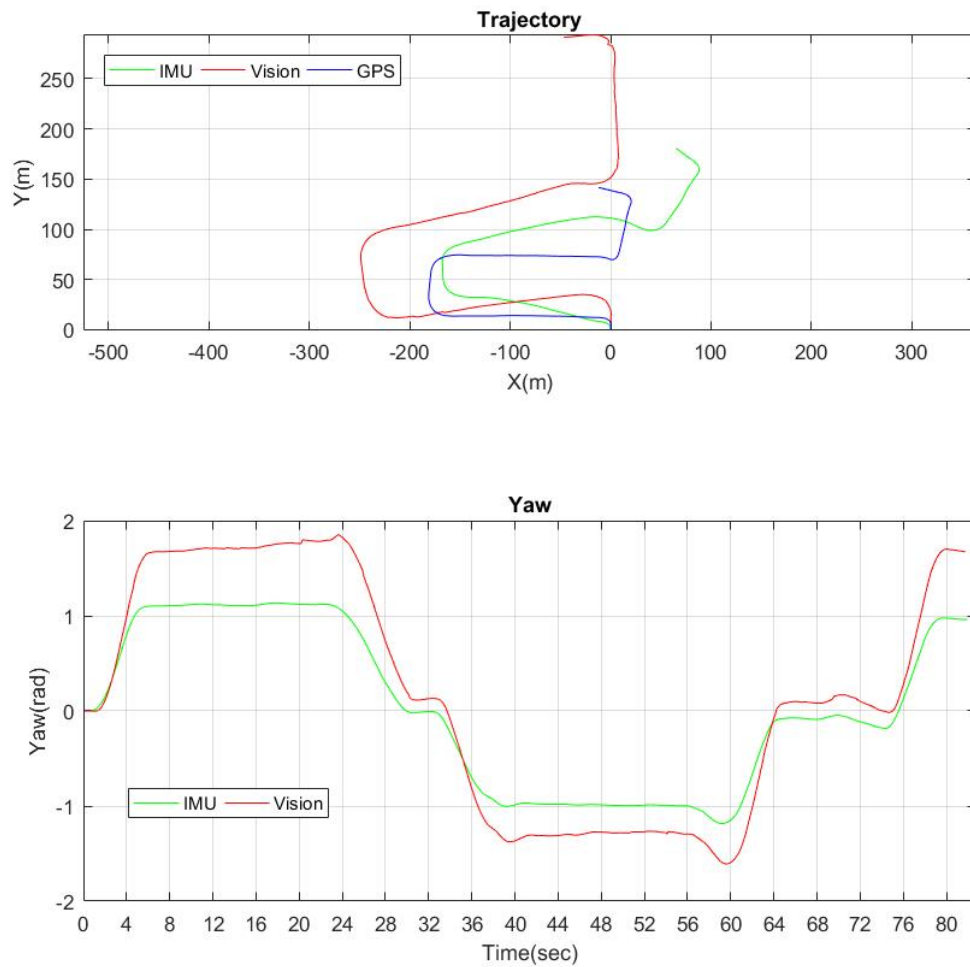


Figure 4.9. FAST 8-point (Kitti dataset)

frames and minimizes the overall error in total motion between all frames. However, in order to get accurate results, the random selection process has to be repeated more than once (trials) to ensure robustness, which adds extra computational overhead for each fundamental matrix estimation step.

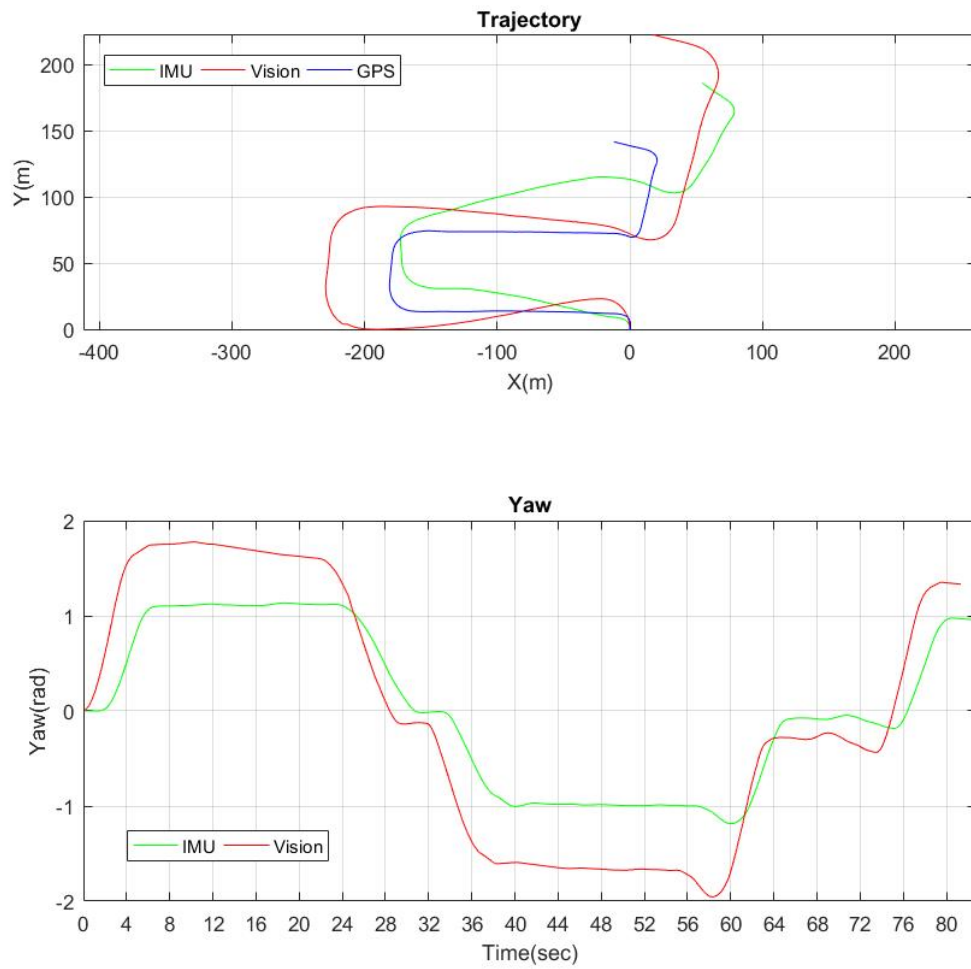


Figure 4.10. SURF MSAC (Kitti dataset)

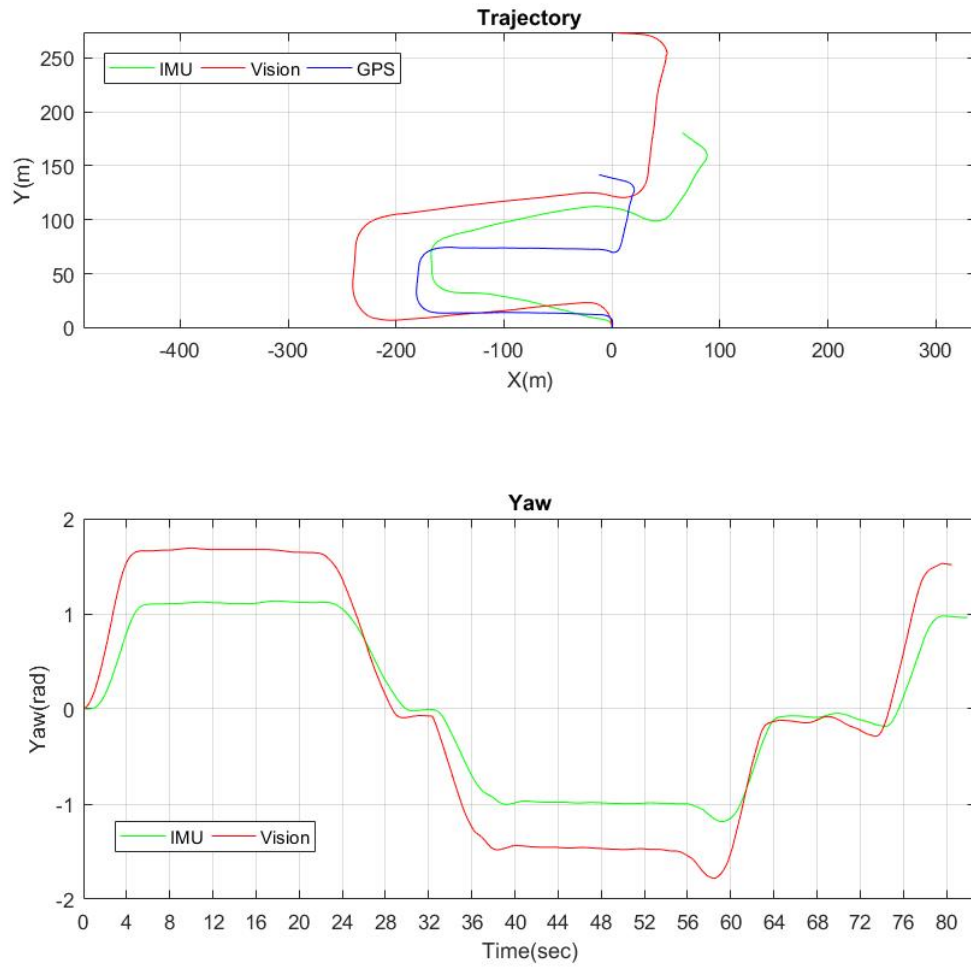


Figure 4.11. Harris MSAC (Kitti dataset)

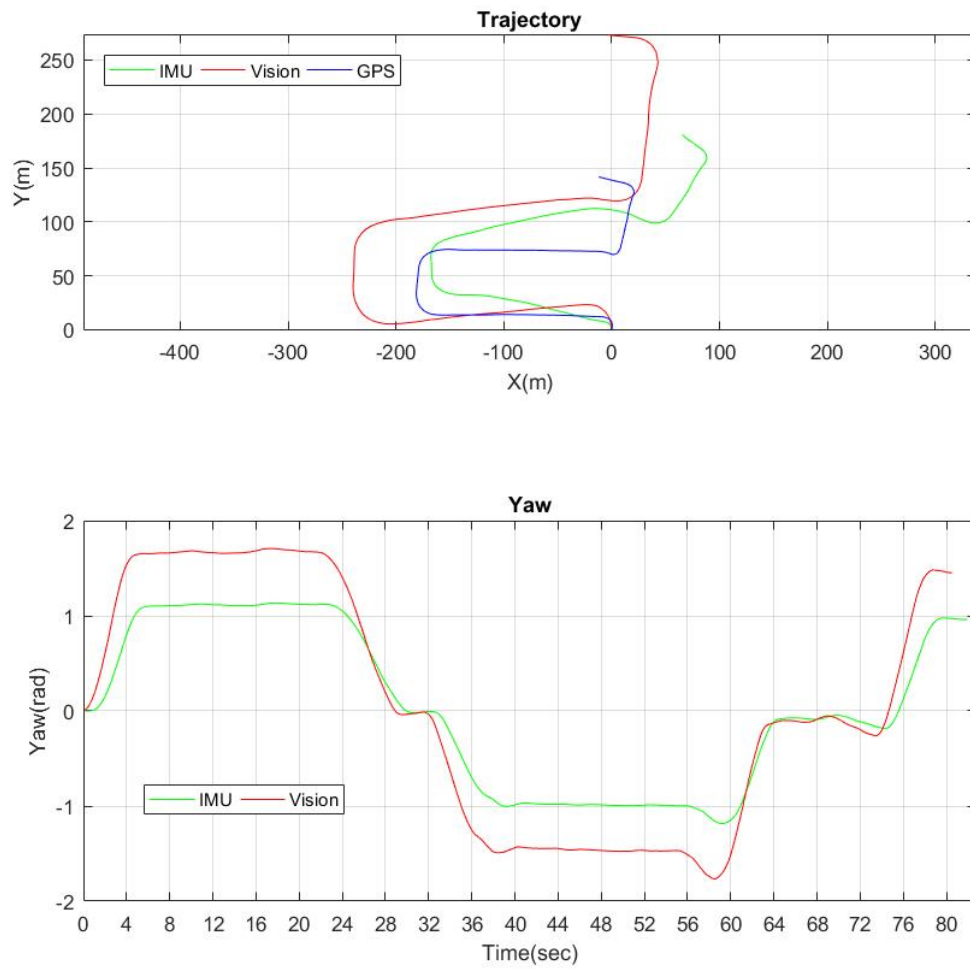


Figure 4.12. FAST MSAC (Kitti dataset)

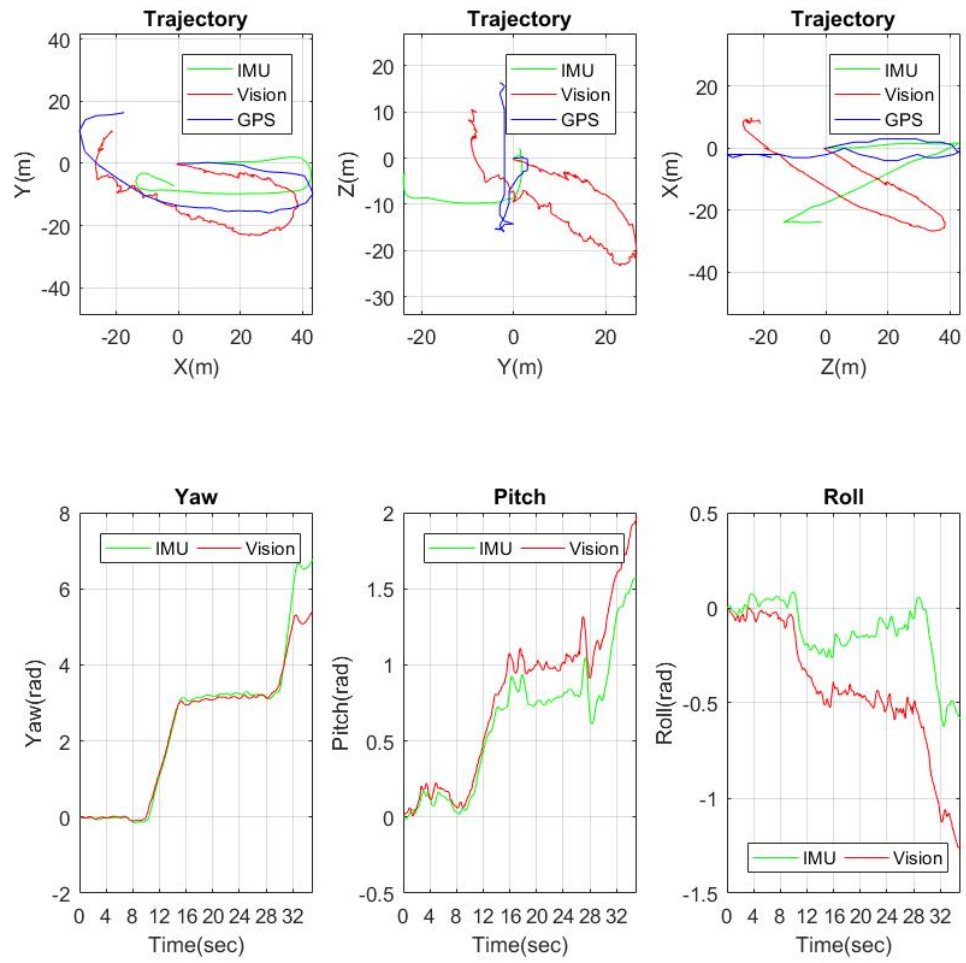


Figure 4.13. SURF MSAC (Quadcopter dataset)

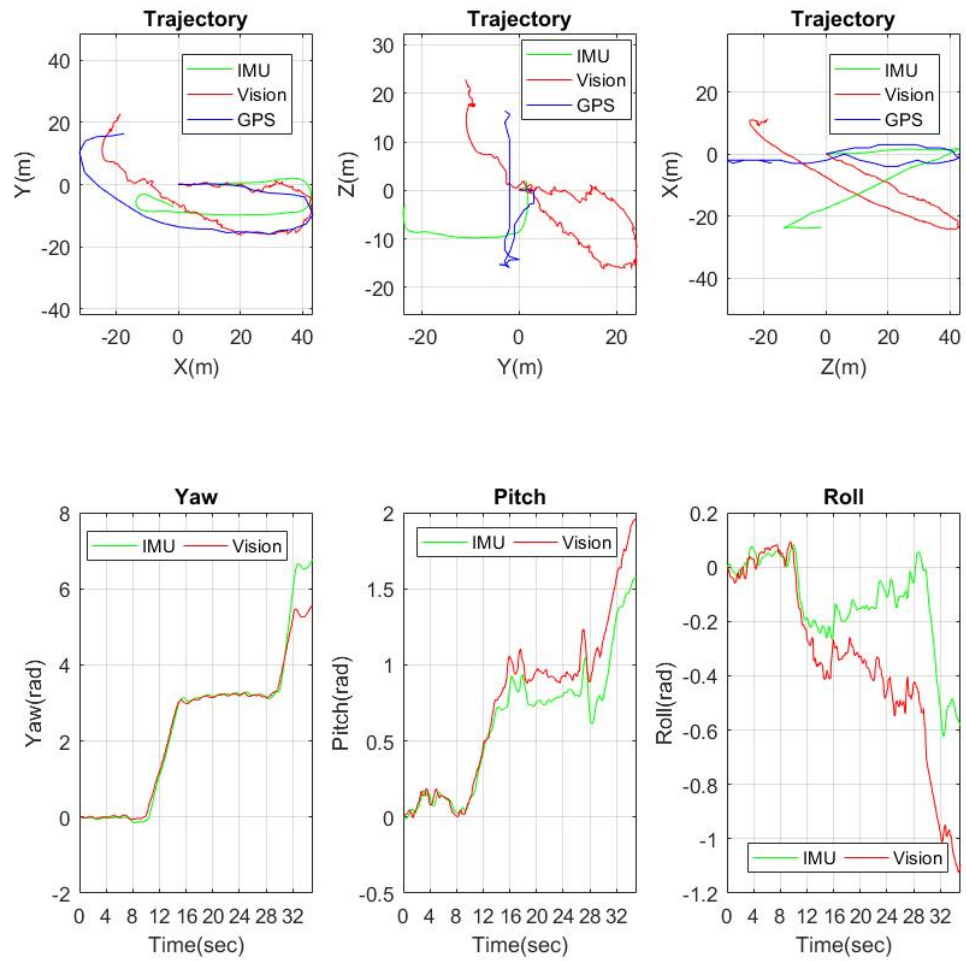


Figure 4.14. FAST MSAC (Quadcopter dataset)

Camera Pose Extraction

As discussed in Chapter 3, the rotation matrix and translation vector between the current and previous frame can be extracted from the fundamental matrix. Combining this information with the last estimated camera pose provides an estimate of the

current rotation/translation of the vehicle in world coordinates (up to a scale factor). This step has another threshold that filters most of the noise using the fact that the rotation change between two consecutive frames is usually very small; hence when encountering large changes, the algorithm discards this information as an outlier and uses the previous pose as the current one. In other words, in this case, the algorithm assumes that the vehicle is rotating with the previous rate with zero translation. As shown in Fig. 4.13, the MSAC method applied to SURF features provided a trajectory that closely matches the GPS output. However, there is one main issue (scale) which is not possible to retrieve using only monocular vision algorithms. This scale factor is solved by using acceleration values from the IMU to estimate the velocity change between each two frames. After that, the scale factor is multiplied by the unit vector computed from the fundamental matrix to get the metric translation between frames. The current camera pose is calculated using Eq. 4.3.

$$(R_I T_I)_t = (R_I T_I)_{t-1} \times \begin{pmatrix} R & -T \\ 0 & 1 \end{pmatrix}^{-1} \quad (4.3)$$

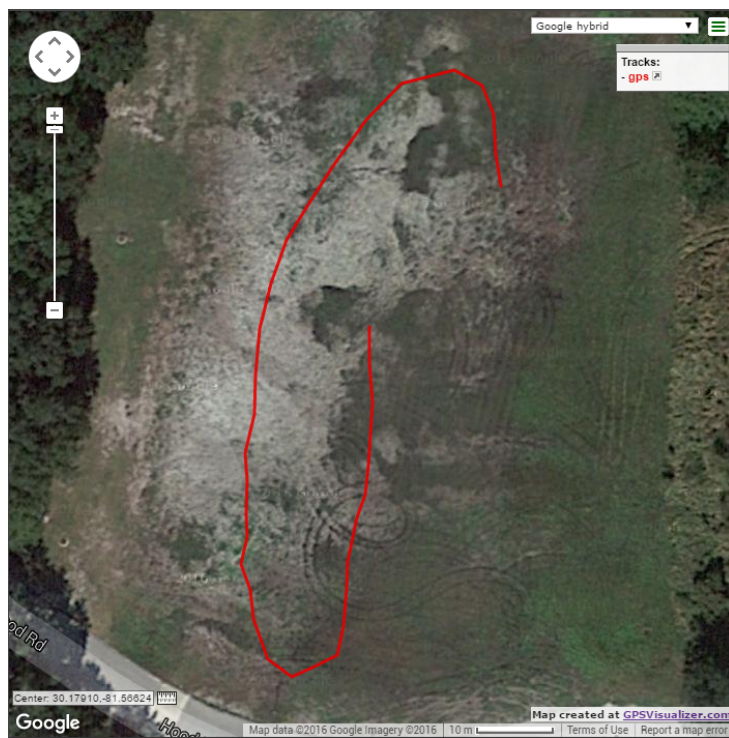
where $(R_I T_I)_t$ is a 4x4 matrix describing the homogeneous rotation and translation values in the world coordinate system. R is a 3x3 matrix that describes the rotation from the previous frame to the current frame coordinate system, and T is a 3x1 vector describing the translation (after multiplying it with the scale factor) from the previous frame to the current frame in the previous frame coordinate system.

4.3 Vision-Aided Navigation Filter Results

The vision-aided navigation filter was evaluated using two experimental data sets. The first was the Kitti dataset (2011'09'26'drive'0022 subset), with the GPS route shown in Fig. 4.15a with 82 seconds of running time and a total of 800 1242x375 gray-scale image frames. GPS/IMU data were measured and synchronized with the frames. The second test was performed on the data collected from the Cheerson CX-20 quadcopter. A 35 seconds subset of data was used and synchronized with GPS/IMU data and a total of 332 image frames. The GPS route from this dataset is shown in Fig. 4.15b. The vision-aided navigation filter results are shown in Figures 4.16 and 4.17 after several tuning attempts using different combinations of process/measurement noise values. The yaw angle drift from the IMU is corrected by the yaw angle estimation using the vision pose measurement, which decreased the error between the estimated vehicle trajectory and the actual path calculated using GPS. It is worth noting that the GPS sensor used was a consumer-grade GPS device integrated into a smart phone, which has mean position error of 3 meters. The results also show that the EKF smooths the estimated trajectory compared to the raw GPS measurements. The SURF and FAST feature detection methods were used in all tests to compare the performance using two different categories of feature detector. FAST provided comparable results to the SURF method with less computational time for both tests. The main tuning parameter was the number of frames to use for tracking. Tracking



(a) Kitti Data Set

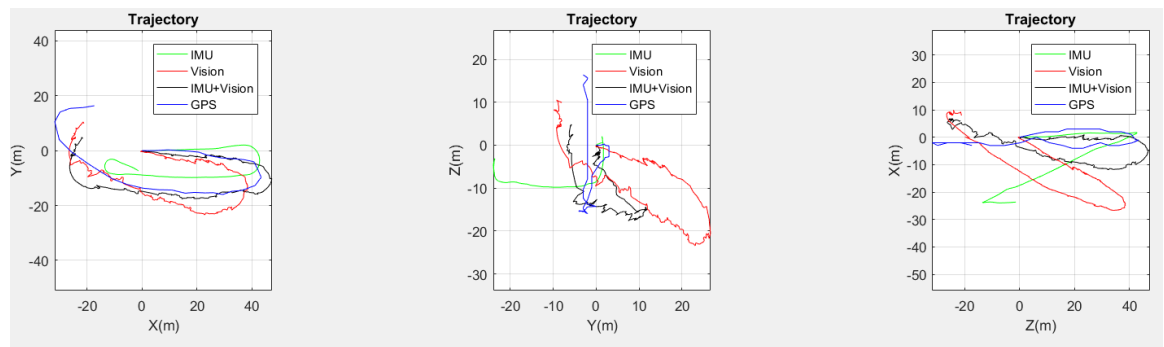


(b) Cheerson Quadcopter Data Set

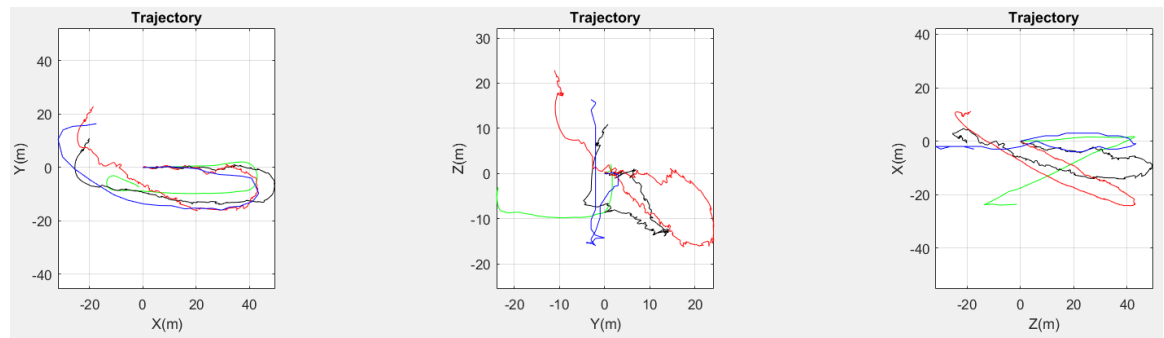
Figure 4.15. GPS Tracks from the Experimental Data Sets

feature points over 2 or 3 frames gave the best results. This parameter relies heavily on the frame rate of the processed image feed and the speed of the vehicle as well.

Figures 4.16 and 4.17 show the comparison between the IMU, Vision and Vision-aided navigation filter output trajectory against the GPS path for the quadcopter and Kitti data sets. While the filtered path does not match the GPS path completely, it follows the GPS track better than the IMU alone and more smoothly than the vision output alone. The figures also show the difference between using the SURF and FAST methods.



(a) SURF



(b) FAST

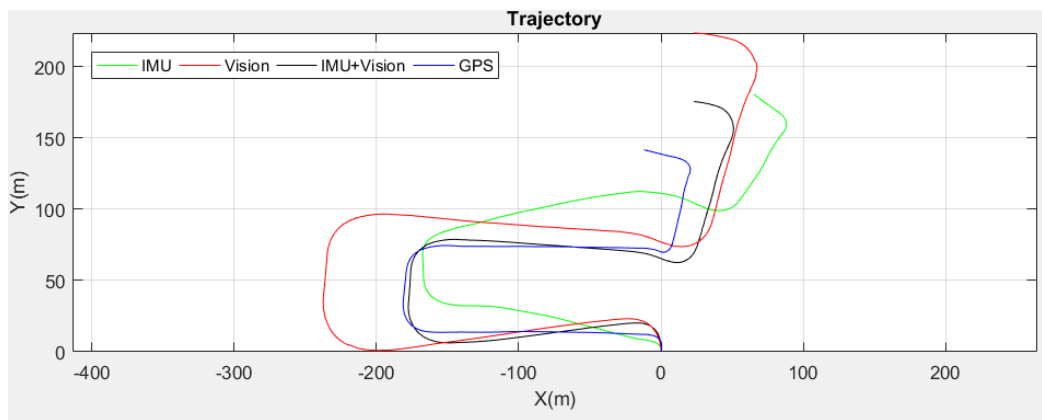
Figure 4.16. Trajectory Estimation (Quadcopter Data Set)

Table 4.1 shows the mean square error (MSE) for the quadcopter and the Kitti datasets, which is the distance between the 3-D position calculated by GPS and the 3-D position calculated by the vision-aided navigation filter defined as follows:

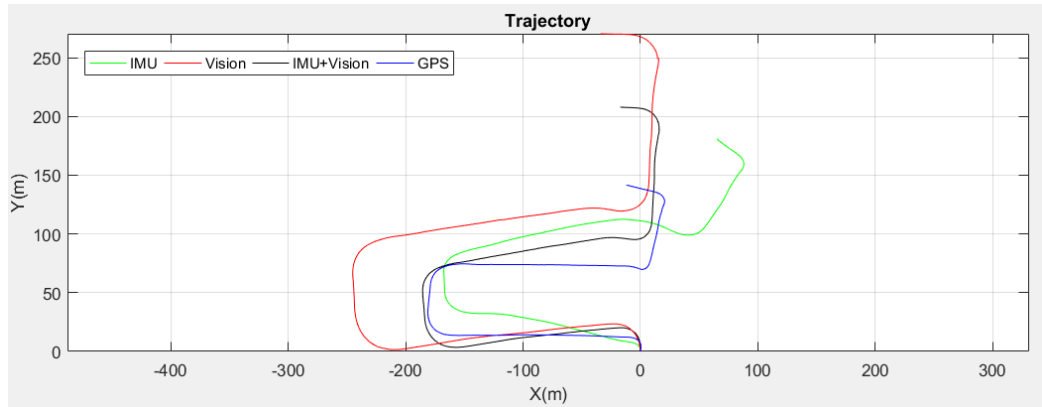
$$MSE = \frac{\sum_{t=0}^N (Pos_{gps,t} - (\sum_{ts=t-range/2}^{t+range/2} Pos_{filter,ts})/range)^2}{N+1} \quad (4.4)$$

where $range$ is the ratio between GPS update rate (1s) and filter update rate (100ms).

Ten filter runs were performed for the MSAC implementations and the average MSE



(a) SURF



(b) FAST

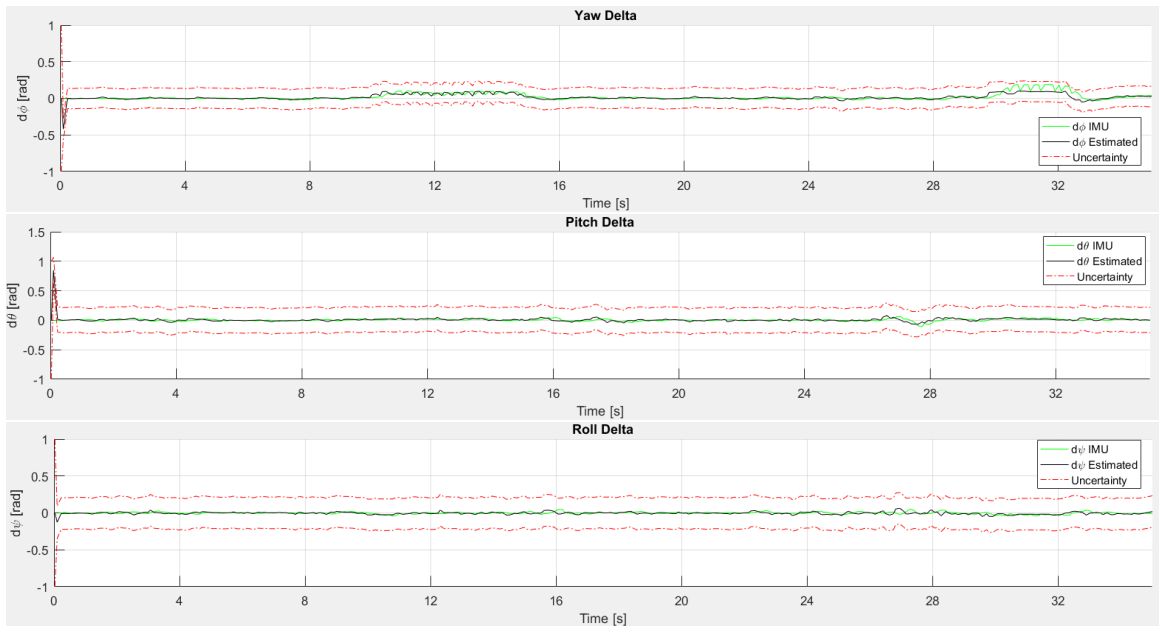
Figure 4.17. Trajectory Estimation (Kitti Data Set)

value was computed. The error becomes much larger when the regular normalized 8-point algorithm is used alone for both datasets. More than 90 percent of running time of the filter is taken by the fundamental matrix estimation step when using the FAST method, while only 1.2 percent of the total time was consumed by the feature point detection step and 8.6 percent for the feature point tracking step. Using the SURF method, 68 percent of the running time was consumed by the fundamental matrix estimation, approximately 25 percent was used by the feature point detection, and the tracking step only used 6 percent with a 1.3X increase in total running time over using the FAST method for the quadcopter dataset.

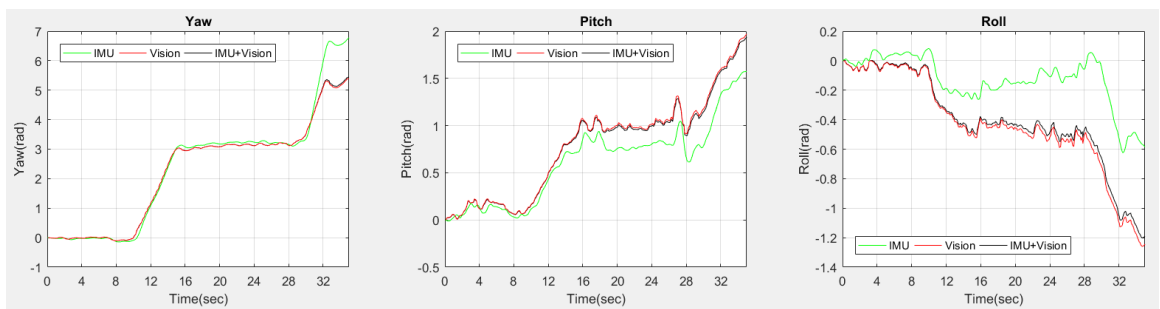
Table 4.1. Trajectory Mean Square Error

Dataset	Feature Detector	MSE(meters²)
Quadcopter	SURF (MSAC)	60.2223
Quadcopter	SURF (Norm8)	90.4586
Quadcopter	FAST (MSAC)	62.0694
Quadcopter	FAST (Norm8)	73.5585
Kitti	SURF (MSAC)	343.7344
Kitti	SURF (Norm8)	7.523E+03
Kitti	FAST (MSAC)	507.1884
Kitti	FAST (Norm8)	3.405E+03

Figures 4.18a and 4.18b show that both the IMU and vision-based algorithms provided similar results most of the time, but due to the drift in the IMU calculation it passes the π boundary earlier than the vision algorithm and because the EKF noise was tuned towards favoring the measurements over the IMU, the filtered output from the vision-aided navigation filter more closely follows the vision curve instead of the IMU curve. A bounded change to the pitch and roll angles is also observed, as expected.



(a) Estimated States vs. Time



(b) Attitude vs. Time

Figure 4.18. Estimated Attitude and Attitude Rates (Quadcopter Data Set)

Fig. 4.19 shows the number of feature points extracted in each frame against the number of matched features between each pair of frames using the SURF method. The whole image resolution (1920x1080 pixels) is used, which provides high quality feature points. A limit of 1000 feature points was used as it provided enough information to estimate the fundamental matrix.

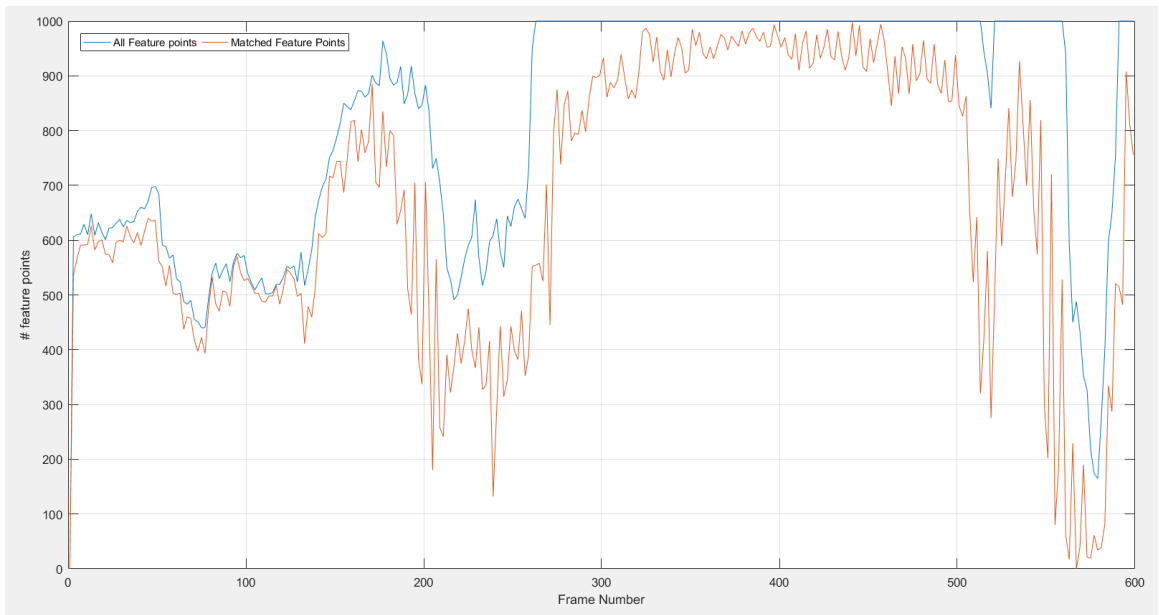


Figure 4.19. Total Feature Points vs. Matched Feature Points

Figure 4.20 shows the effect of using the distance between each pair of matched points to prune out small distances to provide higher quality features to calculate the fundamental matrix. A threshold of 0.7 of the mean distance was used here.

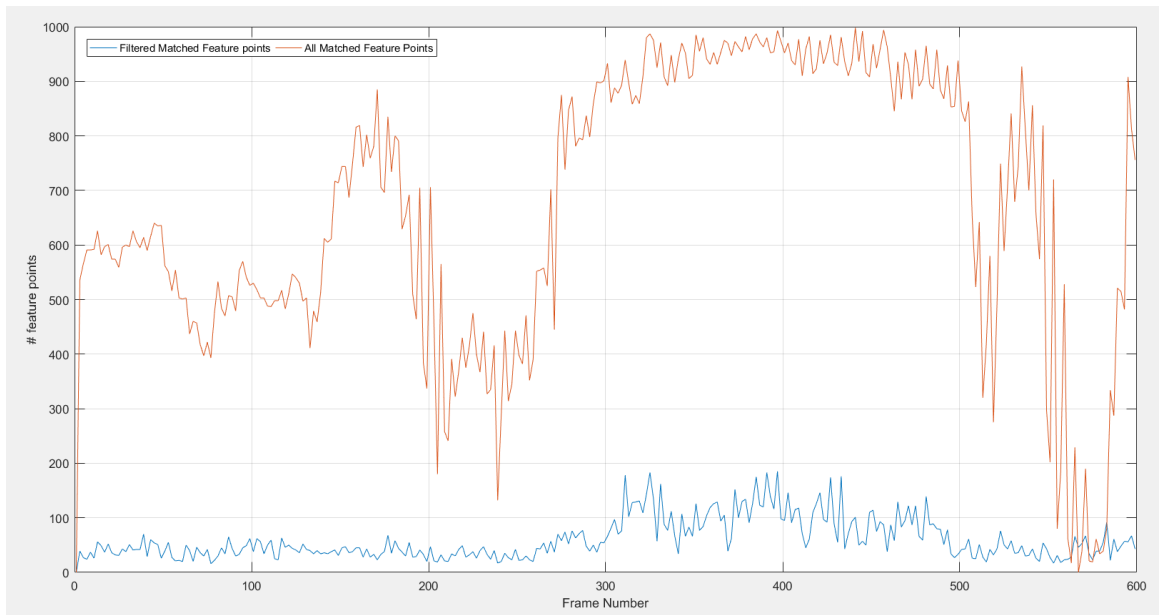


Figure 4.20. Matched Feature Points vs. Filtered Feature Points

5. Conclusion and Recommendations

In this thesis, a vision-aided navigation algorithm was developed that provides sensor fusion of data from an inertial measurement unit (accelerometers and rate gyros) with information extracted from a monocular vision sensor. The algorithm, which takes the form of an extended Kalman filter implementation, utilizes the IMU data for the state propagation step and vision-based information for the measurement update step. This vision-based information corresponds to the frame-to-frame camera rotation and translation, which is computed using tracked feature points and the classical eight-point algorithm. The vision-aided navigation filter was implemented on experimental data obtained from a ground vehicle and a quadcopter UAV. The navigation results were then compared with those obtained using an IMU-based solution (i.e., using only the IMU data to estimate the vehicle motion) and a vision-based solution that used the eight-point algorithm alone to estimate the vehicle motion.

The experimental results show that, even though the vision-aided navigation filter managed to solve partially some of the problems discussed (the scale problem from vision and the drift error from IMU), due to the randomness of feature selection for estimation of the fundamental matrix, the algorithm output was not guaranteed to be accurate for all cases considered even after introducing the extra safeguard of distance checks to account for noisy movement. Out of 20 test runs, 1 run yielded

inaccurate estimates of camera pose from vision, which was corrected by the IMU fusion; however, the position estimation was affected, causing the overall trajectory to drift away from the correct path. One possible solution is to tune the Kalman filter parameters but that requires considerable trial and error.

Another observation from the results is that with smooth movement the algorithm provides better estimation, which can be seen clearly from the Kitti dataset results. Several feature detection algorithms were implemented for use in the vision-aided navigation filter. These included SURF, FAST and the Harris corner detector. Overall, the best pose estimation was achieved using the SURF feature detection method. While the algorithm is not ready for real time implementation, it provides a practical approach which can be tuned to fit into a semi-real time implementation given the increase of processing power in consumer-grade mobile devices. The algorithm was implemented using Matlab but can easily be ported into embedded device programming such as Java or C++, which can fit on a mobile device similar to the one used for the test. As a final note, the work done here addresses a monocular camera in contrast to much of the work that has been done in this field which used stereo vision.

REFERENCES

- Android. (2016). *Developers sensors api*. Retrieved from https://developer.android.com/guide/topics/sensors/sensors_overview.html
- Bay, H., Tuytelaars, T., & Van Gool, L. (2006). Surf: Speeded up robust features. In A. Leonardis, H. Bischof, & A. Pinz (Eds.), *Computer vision – eccv 2006: 9th european conference on computer vision, graz, austria, may 7-13, 2006. proceedings, part i* (pp. 404–417). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/11744023_32 doi: 10.1007/11744023_32
- Chambers, A., Scherer, S., Yoder, L., Jain, S., Nuske, S., & Singh, S. (2014, June). Robust multi-sensor fusion for micro aerial vehicle navigation in gps-degraded/denied environments. In *2014 american control conference* (p. 1892-1899). doi: 10.1109/ACC.2014.6859341
- Fraundorfer, F., Heng, L., Honegger, D., Lee, G. H., Meier, L., Tanskanen, P., & Pollefeys, M. (2012, Oct). Vision-based autonomous mapping and exploration using a quadrotor mav. In *2012 ieee/rsj international conference on intelligent robots and systems* (p. 4557-4564). doi: 10.1109/IROS.2012.6385934
- Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013, September). Vision meets robotics: The kitti dataset. *Int. J. Rob. Res.*, *32*(11), 1231–1237. Retrieved from <http://dx.doi.org/10.1177/0278364913491297> doi: 10.1177/0278364913491297
- Hartley, R., & Zisserman, A. (2003). *Multiple view geometry in computer vision* (2nd ed.). New York, NY, USA: Cambridge University Press.
- Hartley, R. I. (1997, Jun). In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*(6), 580-593. doi: 10.1109/34.601246
- HyperPhysics, G. S. U. (2015). *Gps*. Retrieved from <http://hyperphysics.phy-astr.gsu.edu/hbase/mechanics/imgmech>
- Jin, H., Favaro, P., & Soatto, S. (2001). Real-time feature tracking and outlier rejection with changes in illumination. In *Computer vision, 2001. iccv 2001. proceedings. eighth ieee international conference on* (Vol. 1, p. 684-689 vol.1). doi: 10.1109/ICCV.2001.937588
- Kelly, J., Saripalli, S., & Sukhatme, G. S. (2008). Combined visual and inertial navigation for an unmanned aerial vehicle. In C. Laugier & R. Siegwart (Eds.), *Field and service robotics: Results of the 6th international conference* (pp. 255–264). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from

- http://dx.doi.org/10.1007/978-3-540-75404-6_24 doi: 10.1007/978-3-540-75404-6_24
- Klein, G., & Murray, D. (2007, November). Parallel tracking and mapping for small AR workspaces. In *Proc. sixth IEEE and ACM international symposium on mixed and augmented reality (ISMAR'07)*. Nara, Japan.
- Lindeberg, T. (1993). Detecting salient blob-like image structures and their scales with a scale-space primal sketch: A method for focus-of-attention. *International Journal of Computer Vision*, 11(3), 283–318. Retrieved from <http://dx.doi.org/10.1007/BF01469346> doi: 10.1007/BF01469346
- Longuet-Higgins, H. C. (1981, Sep 10). A computer algorithm for reconstructing a scene from two projections. *Nature*, 293(5828), 133–135. Retrieved from <http://dx.doi.org/10.1038/293133a0> doi: 10.1038/293133a0
- Lowe, D. G. (2004, November). Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2), 91–110. Retrieved from <http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94> doi: 10.1023/B:VISI.0000029664.99615.94
- Lucas, B. D., & Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th international joint conference on artificial intelligence - volume 2* (pp. 674–679). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Retrieved from <http://dl.acm.org/citation.cfm?id=1623264.1623280>
- Mourikis, A. I., & Roumeliotis, S. I. (2007, April). A multi-state constraint kalman filter for vision-aided inertial navigation. In *Proceedings 2007 IEEE international conference on robotics and automation* (p. 3565–3572). doi: 10.1109/ROBOT.2007.364024
- Opencv-python tutorials [Computer software manual]. (2015). OpenCV. Retrieved from <http://opencv-python-tutroals.readthedocs.org/>
- Prazenica, R. J., Hielsberg, M., Sharpley, R., & Kurdila, A. (2013, Aug 15). In (chap. 3-D Implicit Terrain Mapping and Path Planning for Autonomous MAV Flight in Urban Environments). American Institute of Aeronautics and Astronautics. Retrieved from <http://dx.doi.org/10.2514/6.2013-4792> (0) doi: 10.2514/6.2013-4792
- Rosten, E., & Drummond, T. (2005, Oct). Fusing points and lines for high performance tracking. In *Tenth IEEE international conference on computer vision (iccv'05) volume 1* (Vol. 2, p. 1508–1515 Vol. 2). doi: 10.1109/ICCV.2005.104
- Shen, S., Mulgaonkar, Y., Michael, N., & Kumar, V. (2013, May). Vision-based state estimation for autonomous rotorcraft MAVs in complex environments. In *Robotics and automation (icra), 2013 IEEE international conference on* (p. 1758–1764). doi: 10.1109/ICRA.2013.6630808
- Shi, J., & Tomasi, C. (1994, Jun). Good features to track. In *Computer vision and pattern recognition, 1994. proceedings cvpr '94., 1994 IEEE computer society conference on* (p. 593–600). doi: 10.1109/CVPR.1994.323794

Sirtkaya, S., Seymen, B., & Alatan, A. A. (2013, July). Loosely coupled kalman filtering for fusion of visual odometry and inertial navigation. In *Information fusion (fusion), 2013 16th international conference on* (p. 219-226).

Soatto, S., Frezza, R., & Perona, P. (1996, Mar). Motion estimation via dynamic vision. *IEEE Transactions on Automatic Control*, *41*(3), 393-413. doi: 10.1109/9.486640

SpecOut. (2016). *Cheerson cx-20*. Retrieved from <http://drones.specout.com/1/70/Cheerson-CX-20>

Thrun, S., & Leonard, J. J. (2008). Springer handbook of robotics. In B. Siciliano & O. Khatib (Eds.), (p. 871-889). Springer Berlin Heidelberg.

Tomasi, C., & Kanade, T. (1992, November). Shape and motion from image streams under orthography: A factorization method. *Int. J. Comput. Vision*, *9*(2), 137-154. Retrieved from <http://dx.doi.org/10.1007/BF00129684> doi: 10.1007/BF00129684

Webb, T., Prazenica, R., Kurdila, A., & Lind, R. (2004, Aug 16).

In (chap. Vision-Based State Estimation for Autonomous Micro-Air Vehicles). American Institute of Aeronautics and Astronautics. Retrieved from <http://dx.doi.org/10.2514/6.2004-5349> (0) doi: 10.2514/6.2004-5349

Weiss, S., Achtelik, M. W., Lynen, S., Chli, M., & Siegwart, R. (2012, May). Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments. In *Robotics and automation (icra), 2012 ieee international conference on* (p. 957-964). doi: 10.1109/ICRA.2012.6225147

WIDE. (2005). *Advanced topics for marine technology*. Retrieved from <http://www.soi.wide.ad.jp/class/20050026/slides/01/img/61.png>

Zitov, B., & Flusser, J. (2003). Image registration methods: a survey. *Image and Vision Computing*, *21*(11), 977 - 1000. Retrieved from <http://www.sciencedirect.com/science/article/pii/S0262885603001379> doi: [http://dx.doi.org/10.1016/S0262-8856\(03\)00137-9](http://dx.doi.org/10.1016/S0262-8856(03)00137-9)

A. Matlab Code for GPS Lat-Lon-Alt Conversion to NED

```

a = 6378137.0;           % Semi-major axis of WGS 84 ellipsoid
f_inv = 298.257223563; % Reciprocal of flattening for WGS 84 ellipsoid
e2 = 6.69437999014e-3; % First eccentricity squared for WGS 84 ellipsoid
% define a local NED reference frame
lat0_d = lat(1)*pi/180; % initial geodetic latitude
long0_d = lon(1)*pi/180; % initial geodetic longitude
h0 = alt(1);           % initial height above ellipsoid (m)
chi = sqrt(1 - e2*(sin(lat0_d)^2));
x0_ECEF_GPS = (a/chi + h0)*cos(lat0_d)*cos(long0_d);
y0_ECEF_GPS = (a/chi + h0)*cos(lat0_d)*sin(long0_d);
z0_ECEF_GPS = (a/chi*(1-e2) + h0)*sin(lat0_d);
R_GE_3 = [cos(long0_d), sin(long0_d), 0;
          -sin(long0_d), cos(long0_d), 0;
          0, 0, 1];
R_GE_2 = [cos(lat0_d), 0, sin(lat0_d);
          0, 1, 0;
          -sin(lat0_d), 0, cos(lat0_d)];
R_GE_1 = [0, 0, 1; 0, 1, 0; -1, 0, 0];
R_GE = R_GE_1*R_GE_2*R_GE_3;

```

```
r0_ECEF_GPS = [x0_ECEF_GPS, y0_ECEF_GPS, z0_ECEF_GPS]';  
r0_ECEF = r0_ECEF_GPS;  
  
x0_ECEF = r0_ECEF(1);  
y0_ECEF = r0_ECEF(2);  
z0_ECEF = r0_ECEF(3);  
  
for i=1:length(lat)  
    lat_d = lat(i)*pi/180;  
    long_d = lon(i)*pi/180;  
    h = alt(i);  
    chi = sqrt(1 - e2*(sin(lat_d)^2));  
    x_ECEF_GPS = (a/chi + h)*cos(lat_d)*cos(long_d);  
    y_ECEF_GPS = (a/chi + h)*cos(lat_d)*sin(long_d);  
    z_ECEF_GPS = (a/chi*(1-e2) + h)*sin(lat_d);  
    pos_ECEF = [x_ECEF_GPS - x0_ECEF,  
                y_ECEF_GPS - y0_ECEF,  
                z_ECEF_GPS - z0_ECEF]';  
  
    pos_NED(:,i) = R_GE*pos_ECEF;  
end
```

B. Matlab Code for IMU Navigation

```

dt = time(i) - time(i-1);

if(i<5)

    gyro_ang_x = [gyro_ang_x; wrap2PI(gyro_ang_x(end)+p_gyro(i-1)*dt)];
    gyro_ang_y = [gyro_ang_y; wrap2PI(gyro_ang_y(end)+q_gyro(i-1)*dt)];
    gyro_ang_z = [gyro_ang_z; wrap2PI(gyro_ang_z(end)+r_gyro(i-1)*dt)];

else

    gyro_ang_x = [gyro_ang_x; wrap2PI(gyro_ang_x(end)+(p_gyro(i-1)*dt +
    2*p_gyro(i-2)*dt + 2*p_gyro(i-3)*dt+p_gyro(i-4)*dt)/6)];
    gyro_ang_y = [gyro_ang_y; wrap2PI(gyro_ang_y(end)+(q_gyro(i-1)*dt +
    2*q_gyro(i-2)*dt + 2*q_gyro(i-3)*dt+q_gyro(i-4)*dt)/6)];
    gyro_ang_z = [gyro_ang_z; wrap2PI(gyro_ang_z(end)+(r_gyro(i-1)*dt +
    2*r_gyro(i-2)*dt + 2*r_gyro(i-3)*dt+r_gyro(i-4)*dt)/6)];

end

R_psi = [cos(gyro_ang_z(end) -gyro_ang_z(end-1)), sin(gyro_ang_z(end)
-gyro_ang_z(end-1)), 0;
-sin(gyro_ang_z(end) -gyro_ang_z(end-1)),
cos(gyro_ang_z(end) -gyro_ang_z(end-1)), 0;
0, 0, 1];

R_theta = [cos(gyro_ang_y(end) -gyro_ang_y(end-1)), 0,

```

```

-sin(gyro_ang_y(end) -gyro_ang_y(end-1));
0, 1, 0;
sin(gyro_ang_y(end) -gyro_ang_y(end-1)), 0, cos(gyro_ang_y(end)
-gyro_ang_y(end-1));
R_phi = [1, 0, 0;
0, cos(gyro_ang_x(end) -gyro_ang_x(end-1)), sin(gyro_ang_x(end)
-gyro_ang_x(end-1));
0, -sin(gyro_ang_x(end) -gyro_ang_x(end-1)), cos(gyro_ang_x(end)
-gyro_ang_x(end-1))];
R_EBtemp = R_psi*R_theta*R_phi;
R_EB = R_EB*R_EBtemp';

a_meas_B = [ax_meas(i-1), ay_meas(i-1), az_meas(i-1)]';
a_meas_E = R_EB'*a_meas_B;

newT = [xVelIMU*dt,yVelIMU*dt,zVelIMU*dt]';
TrIMU(:, :, i) = TrIMU(:, :, i-1) / [R_EBtemp, -newT; 0, 0, 0, 1];
rotZYX = rotm2eul(TrIMU(1:3, 1:3, i));
y = rotZYX(2);
z = rotZYX(1);
x = rotZYX(3);

zRotVIMU = [zRotVIMU; z ];
xRotVIMU = [xRotVIMU; x ];
yRotVIMU = [yRotVIMU; y ];
xVIMU = [xVIMU; (TrIMU(1, 4, i-1))];
yVIMU = [yVIMU; (TrIMU(2, 4, i-1))];

```

```
zVIMU = [zVIMU; (TrIMU(3,4,i-1))];  
xVelIMU = xVelIMU+a_meas_E(1)*dt;  
yVelIMU = yVelIMU+a_meas_E(2)*dt;  
zVelIMU = zVelIMU+a_meas_E(3)*dt;
```


C. Matlab Code for Vision Algorithm

```
cImgT1In = readFrame(inMov);  
cImgT1In = rgb2gray(cImgT1In);  
imagePoints1 = detectSURFFeatures(cImgT1In);  
% Create the point tracker  
tracker = vision.PointTracker('NumPyramidLevels', 5,  
    'MaxBidirectionalError', 0.1);  
% % Initialize the point tracker  
imagePoints1 = imagePoints1.Location;  
initialize(tracker, imagePoints1, cImgT1In);  
%% for all frames  
total_frame_cnt = 0;  
while(inMov.CurrentTime - time_off < time_sync)  
    if(inMov.CurrentTime - time_off > 30)  
        break;  
    end  
    skip_frame = 0;  
    if(mod(frame_cnt,2) == 0 || curSize < 20)  
  
        cImgT2In = readFrame(inMov);  
        cImgT2In = rgb2gray(cImgT2In);
```

```

imagePoints1 = detectSURFFeatures(cImgT2In);

% reset the point tracker

imagePoints1 = imagePoints1.Location;

if(size(imagePoints1) > 0)

    setPoints(tracker, imagePoints1 );

end

frame_cnt = frame_cnt + 1;

end

frame_cnt = frame_cnt + 1;

cImgT2In = readFrame(inMov);

cImgT2In = rgb2gray(cImgT2In);

[imagePoints2, validIdx] = step(tracker, cImgT2In);

matchedPoints1 = imagePoints1(validIdx, :);

matchedPoints2 = imagePoints2(validIdx, :);

curSize = size(matchedPoints1,1);

if(size(matchedPoints1,1) < 20)

    skip_frame = 1;

end

if(skip_frame == 0)

    dis_sq = (matchedPoints1(:,1) - matchedPoints2(:,1)).^2 +

    (matchedPoints1(:,2) - matchedPoints2(:,2)).^2;

    mean_dis_sq = mean(dis_sq)

    if(mean_dis_sq < 50) % stopped or very little motion

        skipped_stopped = skipped_stopped+1

        skip_frame = 1;

```

```

end

filteredValidIdx = dis_sq >= (mean_dis_sq*1);

inlierPoints1 = (matchedPoints1(filteredValidIdx, :))';
inlierPoints2 =
(cameraAdjR*matchedPoints2(filteredValidIdx, :))';

if(length(inlierPoints1) < 16)
    skipped_notenough = skipped_notenough + 1
    skip_frame = 1;
end

if(skip_frame == 0)
    [fMatrix, epipolarInliers, status] =
estimateFundamentalMatrix(...
inlierPoints1, inlierPoints2, 'NumTrials',
1000, 'Method', 'MSAC', 'DistanceThreshold', 1e-4);
    if(status == 0)
        % Find epipolar inliers
        inlierPoints1 = inlierPoints1(epipolarInliers, :);
        inlierPoints2 = inlierPoints2(epipolarInliers, :);

        if(length(inlierPoints1) < 16)
            skipped_notenough = skipped_notenough + 1
            skip_frame = 1;
        end
    end
end

```

```

        end

    else

        skipped_notenough = skipped_notenough + 1

        skip_frame = 1;

    end

end

if(skip_frame == 0)

    [fMatrix, epipolarInliers] = estimateFundamentalMatrix(...
    inlierPoints1, inlierPoints2, 'NumTrials',
    100, 'Method', 'MSAC', 'DistanceThreshold', 1e-4);

    inlierPoints1 = (inlierPoints1(epipolarInliers, :))';
    inlierPoints2 =
    (cameraAdjR*inlierPoints2(epipolarInliers, :))';

end

end

if(skip_frame == 0)

    [R, t] = cameraPose(fMatrix, cameraParamsL,
    inlierPoints1, inlierPoints2);

    rotZYX = rotm2eul(R);

    y = rotZYX(2);

    z = rotZYX(1);

    x = rotZYX(3);

else

    z = 1;

    t = [0 0 0];

end

end

```

```

t = t/(inMov_FrameRate);

if(steps_cnt < 5)

    newR = eye(3);

else

    newR = Rs(:, :, steps_cnt-1);

end

if(( abs(z) > 0.1 || abs(x) > 0.1 || abs(y) > 0.1) && steps_cnt > 1)

    cor_cnt = cor_cnt+1;

    newT= ts(steps_cnt-1, :);

    Tr(:, :, steps_cnt) = Tr(:, :, steps_cnt-1) / [newR, -newT'; 0, 0, 0, 1];

    R = newR;

    t = newT;

else

    tempTr = Tr(:, :, steps_cnt-1) / [R, -t'; 0, 0, 0, 1];

    Tr(:, :, steps_cnt) = tempTr;

end

total_frame_cnt = total_frame_cnt + 1;

end

```